Contributions to improve the technologies supporting unmanned aircraft operations

by

Juan Pedro Llerena Caña

A dissertation submitted by in partial fulfillment of the requirements for the degree of Doctor of Philosophy in

Computer Science and Technology

Universidad Carlos III de Madrid

Thesis Advisors:

Prof. Jesús García
Prof. José Manuel Molina

July 2023

Juan Pedro Llerena Caña
j.p.llerena.cana@gmail.com
Applied Artificial Intelligence Group (GIAA)
Computer Science and Engineering Department
Universidad Carlos III de Madrid
Avd. de Gregorio Peces-Barba Martínez, 22
28270 Colmenarejo, Madrid, (Spain).

Dedicado a mis padres y hermanos por su apoyo incondicional.

Dedicado a la energía,
a lo incontrolable, al esfuerzo,
a la sorpresa, al desafío,
al pasado, presente y futuro,
a la alegría, paciencia y belleza,
dedicado a Ana.

Acknowledgements

Throughout the almost four years that this thesis has lasted, I have been fortunate to have the support of many good people who have undoubtedly been an indispensable part of this adventure.

First of all, I would like to thank my supervisors Jesús and José Manuel, for giving me the opportunity to be involved in this project and particularly for their trust in me. You always motivated me with new challenges and when chaos invaded my work, your advice helped me to order it and make sense of it.

To the members of the Applied Artificial Intelligence Group, Miguel Angel, Antonio, and Javier, thank you for many months of dinner talks, your willingness to help and your warm welcome in this family. To my colleagues Daniel and David, thank you for your help, support, and generosity, without you several of my works would not have been possible. To my laboratory colleagues: Álvaro, Cristina, Lázaro, Laura, Pablo, how good it has been to share the laboratory with you. Covadonga, thank you for your wonderful cooking, company, and friendship during these years. I will always remember the thousands of coffees and discussions when the hardest part of the pandemic was over, and we could return to the university.

I would like to thank the more than thirty-seven UC3M and international students who relied on my proposals to complete their final projects degree or research internships. All these proposals have allowed me to learn a lot from all of you.

To Daniel Arias for open me the doors of the German Aerospace Center and introduce me to the great team of Nautical Systems of the Institute of Communications and Navigation. Without a doubt, your team is a reference to follow. Thanks to Ralf, Stefan, Christoph, Xiandong, Lars, Niklas, Alonso, Andrea, and Lukas for hosting me. Thanks to Filippo for your coffee rituals, coffee shop talks, for your wonderful community bike and for showing me the best of Neustrelitz.

To my friends Alejandro Martínez, Ignacio Olabarría, Juan de la Torre, Carlos Outón, Miguel Olías, Víctor García, Alejandro Cremades; thank you for your talks, advice, climbs, hikes and in general for your time and friendship, which year after year have led me, among many other things, to write these lines. Dear friends, you always have been, are and will be a reference for me.

Finally, I would like to thank the Ministry of Science and Innovation for granting me the funding with reference PRE2018-086793, associated to the project TEC2017-88048-C2-2-R, which provide me the opportunity to carry out all my PhD. activities, including completing an international research internship.

Published and submitted content

All the contributions of this doctoral thesis have been published or is in production in journals, conferences, and textbooks. This section enumerates the involved publications and relates them to the main chapters and sections, specifying the way of inclusion.

Journal Articles

- (August 2021) J. P. Llerena, J. García, and J. M. Molina, "An approach to forecasting and filtering noise in dynamic systems using LSTM architectures," *Neurocomputing*, vol. 500, pp. 637–648, 2022.

  DOI: 10.1016/j.neucom.2021.08.162
  Impact factor: 4.99/ Q1 (2020) Q2 (2021)
  Role: First author.
  Statement: the content from this publication is fully included in Chapter 6

- (March 2021) J. P. Llerena, J. García, and J. M. Molina, "Forecasting nonlinear systems with LSTM: Analysis and comparison with EKF," *Sensors*, vol. 21, no. 5, pp. 1–29, 2021.

  DOI: 10.3390/s21051805
  Impact factor: 4.05/ Q2 (2021)
  Role: First author.
  Statement: the content from this publication is fully included in Chapter 7.

- (May 2022) J. P. Llerena, J. García, and J. M. Molina, "Error Reduction in Vision-Based Multirotor Landing System," *Sensors 2022, Vol. 22, Page 3625*, vol. 22, p. 3625, 2022.

  DOI: 10.3390/s22103625
  Impact factor: 4.05/ Q2 (2021)
  Role: First author.
  Statement: the content from this publication is fully included in Chapter 4.

Conference Articles

- (August 2020) J. P. Llerena, J. García, and J. M. Molina, "An Approach to Forecasting and Filtering Noise in Dynamic Systems Using LSTM Architectures," 15th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2020). Advances in Intelligent Systems and Computing., 2020, vol. 1268 AISC, pp. 155–165.

  DOI: 10.1007/978-3-030-57802-2_15

Ratings/Class[1]: A+/Class1

Role: First author.

Statement: the content from this publication is partially included in Chapter 6,7.

- (September 2021) J. P. Llerena, J. García, and J. M. Molina, “LSTM vs CNN in Real Ship Trajectory Classification,” 16th International Conference on Soft Computing Models in Industrial and Environmental Applications, SOCO 2021, Advances in Intelligent Systems and Computing, 2021, vol. 1268 AISC, pp. 58–67.

  DOI: 10.1007/978-3-030-87869-6_6

  Impact factor: A+/Class1

  Role: First author.

  Statement: the content from this publication is fully included in Chapter 8.

- (July 2022) J. P. Llerena, J. G. Herrero and J. M. M. López, "Error reduction in autonomous multirotor vision-based landing system with helipad context," 2022 25th International Conference on Information Fusion (FUSION), Linköping, Sweden, 2022, pp. 1-8.

  DOI: 10.23919/FUSION49751.2022.9841325

  Impact factor: B-/Class3

  Role: First author.

  Statement: the content from this publication is partially included in Chapter 4

Submitted Works

- (Book in production) J. García, J. M. Molina, J. P. Llerena, D. Amigo and D. Sánchez, “Engineering UAS Applications: Sensor Fusion, Machine Vision, and Mission Management” In production by Artech House,

  Role: Third author.

  Statement: the content from this publication is partially included in Chapter 1-3.

- (Journal Article) J. P. Llerena, J. García, and J. M. Molina, “LSTM vs CNN in Real Ship Trajectory Classification” extension Work submitted, accepted, and awaiting publication in Logic Journal of the IGPL, by Oxford Journals.

  ISSN: 1368-9894

  Impact factor: 0.861/Q1

  Role: First author.

[1] https://scie.lcc.uma.es:8443/gii-grin-scie-rating/ratingSearch.jsf

Statement: the content from this publication is fully included in Chapter 8

- (Conference Article) J. P. Llerena, J. García, J. M. Molina and Daniel Arias “Tuning process noise in INS/GNSS fusion for drone navigation based on evolutionary algorithms,” 18th International Conference on Soft Computing Models in Industrial and Environmental Applications, SOCO 2023, Advances in Intelligent Systems and Computing, 2023.

  Impact factor: A+/Class1
  Role: First author.
  Statement: Not Including

Other Research Merits

Several additional contributions have been made throughout this thesis. These include completed research that is in the process of finding a suitable journal, additional conferences, lecture organization, student supervision and participation in research projects of the research group.

Pending publication

- (Journal Article) R. Móstoles, J.P. Llerena, A. Losada, J. García y J.M. Molina “A review of image datasets for human emotions recognition with ML techniques”

  Status: manuscript in search for a journal.
  Role: Second author.
  Statement: Not Including

Other conferences

- (September 2021) R. Móstoles, J.P. Llerena, A. Losada, J. García y J.M. Molina “Sistemas empotrados para el reconocimiento cognitive de conductores”, XIX Conference of the Spanish Association for Artificial Intelligence, Málaga, Spain.

  Role: Second author.
  Statement: Not Including

- (Submitted) Lukas Hösch, Alonso Llorente, Xiangdong An, Juan Pedro Llerena, Daniel Medina "High Definition Mapping for Inland Waterways: Techniques, Challenges and Prospects" 26th IEEE International Conference on Intelligent Transportation Systems.

  Impact factor: A-/Class2
  Role: Third author.
  Statement: Not Including

- (Submitted) D. Sánchez, D. Amigo, J. G. Herrero, J. M. M. López and J.P. Llerena "UAV airframe classification based on trajectory data in UTM collaborative environments," 2023 26th International Conference on Information Fusion (FUSION).

  Impact factor: B-/Class3
  Role: Fifth author.
  Statement: Not Including

Conference sessions chair / partner

- Committee member of the workshop, "Artificial Intelligent Applied to Intelligent Transport Systems" (AI-SIT 2021[2]) conferences of the "*Asociación Española para la Inteligencia Artificial*" (CAEPIA), Málaga (Spain), September 22-24 of 2021.
- Organization of lectures by international students in the Applied Artificial Intelligence Group (GIAA) in September 2020 and July 2021, 2022.
- Member of the Flying Labs[3,4] network, from 2019.

Advised Students projects

Having been trusted by the students, who found all the suggestions interesting, we were able to explore the following works.

UC3M Polytechnic School

1) (Álvaro Archilla Franco, 2020) "*Diseño e implementación de un sistema de estabilización híbrida de imágenes para la captura de video*".
2) (Jia Wei Qi, 2020) "*Evaluación del sistema de fusión sensorial de navegación de un UAV en entorno simulado*".
3) (Daniel García Sousa, 2020) "*Data science explotación y análisis de datos de calidad de aire en la ciudad de Madrid*".
4) (David Molina Morejón, 2020) "*Mapping para UAV*".
5) (Víctor Moreno Azofra, 2021) "*Detección de obstáculos mediante técnicas de visión artificial en UAVs*".
6) (Dilmer Vilca Maguiña, 2021) "*Procesado de imágenes en estación de tierra*".

[2] https://caepia20-21.uma.es/w_ia_sit.html
[3] https://flyinglabs.org/
[4] https://flyinglabs.org/spain/

7) (Alberto Bringas Gil, 2021) *"Reconocimiento por visión artificial y aterrizaje autónomo en helisuperfícies".*

8) (Darío López Salamanca, 2021) *"Sistema de seguimiento de objetos estabilizado embarcable en drones".*

9) (Alejandro Martínez Hermoso, 2021) *"Captura y transmisión telemétrica de datos de sensores embarcados".*

10) (Alberto Ramos González, 2021) "*Detección de objetos en movimiento con sensor LIDAR".*

11) (Alejandro Santana Duro, 2021) *"Optimización de trayectorias de vuelos de UAVs".*

12) (Pablo San Gil Satrústegui, 2021) "*Artificial Intelligence in the Aid to the Diagnosis of Pulmonary Diseases".*

13) (Adrián Gómez Montoro, 2021) *"Detección de obstáculos con LIDAR 2D para Navegación de drones".*

14) (Eduardo Andrés Castro Attías, 2021) *"Redes neuronales para detección de objetos computada en GPU para navegación autónoma de UAVs".*

15) (Laura Danielsson Borrasca, 2022) *"Redes LoRaWAN para el despliegue ágil de sensores con drones".*

16) (Enrique Ortiz Ibáñez, 2022) *"Detección y seguimiento de un objeto mediante visión artificial desde un dron de bajo coste comercial"*.

17) (Marcos Castillo Vélez, 2022) *"Drone object tracking system in hyper-realistic vision simulator".*

18) (Álvaro Martino Ortiz, 2023) *"Optimización de parámetros del sistema de navegación de un dron"*. (In progress)

19) (Samuel Halstead Aldea, 2023) *"Optimización de parámetros del sistema de navegación de un dron en SITL"*. (In progress)

20) (Jaime Martínez, 2023) *"Segmentación semántica, en videos altamente ocluidos"*. (In progress)

21) (Natalia Cores, 2023) *"Segmentación semántica de videos con aprendizaje por refuerzo".* (In progress)

22) (Kai Ye, 2023) *"Seguimiento multiobjetivo con aprendizaje profundo"*. (In progress)

23) (Ramsés Contreras, 2023) *"Memorias Atkinson-Shiffrin en seguimiento multiobjetivo"*. (In progress)

International Advised Students

1. *"Obstacle detection with computer vision"*, Tristan Cotte, INSA, Strasbourg, France.
2. *"Obstacle detection and mapping with LIDAR From UAVS"*, Mustapha EL YOUSSOUFY, INSA, Strasbourg, France.
3. *"Data Minining in the datasets of COVID-19"*, Alexoudi Panagiota, Aristoteles University of Thessaloniki, faculty of Science, Department of mathematics, Tesalónica, Greece.
4. *"Computer Vision for Drones Landing Problems"*, Bouja Mehidi, Thelecom Physique Strasbourg, France.
5. *"Obstacle Detection and Avoidance through Computer Vision for UAV"*, Javier Pérez, Thelecom Physique Strasbourg, France.
6. *"Evaluation of Follow-Me flight mode to track a target with drone"*, Louis Brulebois, Thelecom Physique Strasbourg, France.
7. *"Smooth path planning in constrained environments for unmanned aerial vehicles"*, Yassine Lambarki, Thelecom Physique Strasbourg, France.
8. *"Computer Vision for Drones navigation and detection Problems"*, Reda Choukri, Thelecom Physique Strasbourg, France.
9. *"Start-up of a sensor fusion system optimization platform"*, Hanae Tazi, Thelecom Physique Strasbourg, France.
10. *"Start-up LoRaWAN networks to fast deployment of sensors embedded in drones"*, Vagnona Andrianandrasana, Thelecom Physique Strasbourg, France.
11. *"On Classical and Contemporary Tracking Algorithms"*, Austin Yu[5], Johns Hopkins University, Maryland, USA.
12. *"Image segmentation in urban environment"*, Rohan Mukundhan, Johns Hopkins University, Maryland, USA. (In progress)
13. *"Deep learning and multi-object tracking"*, Marc-Antoine CHARLES, Thelecom Physique Strasbourg, France. (In progress)

[5] https://www.uc3m.es/C3IS/researchlabs

14. "*Visual-Inertial Odometry in AirSim*", Emran Mustafa, Trinity College Dublin, Ireland. (In progress)

Alternative dissemination activities

1. De Python al análisis de datos[6]. Scientific-dissemination program of UC3M aimed at high schools.
2. Drones, aplicaciones y futuro presente[7]. Scientific-dissemination program of UC3M aimed at high schools.
3. Artificial Intelligence and Computer vision, Seminar in "Cooperativa de enseñanaza José Ramón Otero", Madrid, Spain.

Applied Artificial Intelligence Group Projects

- SIMBAT PROJECT (https://giaa.uc3m.es/simbat-project/)
    - Status: in progress
- HADA - Investigación de Herramientas de Anotación de Datos Avanzadas
    - Status: in progress

[6] https://www.uc3m.es/ss/Satellite/Secundaria/es/TextoDosColumnas/1371299683214/

[7] https://www.uc3m.es/ss/Satellite/Secundaria/es/TextoDosColumnas/1371299682768/#profesorado

Abstract

Unmanned Aerial Vehicles (UAVs), in their smaller versions known as drones, are becoming increasingly important in today's societies. The systems that make them up present a multitude of challenges, of which error can be considered the common denominator. The perception of the environment is measured by sensors that have errors, the models that interpret the information and/or define behaviors are approximations of the world and therefore also have errors. Explaining error allows extending the limits of deterministic models to address real-world problems. The performance of the technologies embedded in drones depends on our ability to understand, model, and control the error of the systems that integrate them, as well as new technologies that may emerge.

Flight controllers integrate various subsystems that are generally dependent on other systems. One example is the guidance systems. These systems provide the engine's propulsion controller with the necessary information to accomplish a desired mission. For this purpose, the flight controller is made up of a control law for the guidance system that reacts to the information perceived by the perception and navigation systems. The error of any of the subsystems propagates through the ecosystem of the controller, so the study of each of them is essential.

On the other hand, among the strategies for error control are state-space estimators, where the Kalman filter has been a great ally of engineers since its appearance in the 1960s. Kalman filters are at the heart of information fusion systems, minimizing the error covariance of the system and allowing the measured states to be filtered and estimated in the absence of observations. State Space Models (SSM) are developed based on a set of hypotheses for modeling the world. Among the assumptions are that the models of the world must be linear, Markovian, and that the error of their models must be Gaussian. In general, systems are not linear, so linearization are performed on models that are already approximations of the world. In other cases, the noise to be controlled is not Gaussian, but it is approximated to that distribution in order to be able to deal with it. On the other hand, many systems are not Markovian, i.e., their states do not depend only on the previous state, but there are other dependencies that state space models cannot handle.

This thesis deals a collection of studies in which error is formulated and reduced. First, the error in a computer vision-based precision landing system is studied, then estimation and filtering problems from the deep learning approach are addressed. Finally, classification concepts with deep learning over trajectories are studied. The first case of the collection

studies the consequences of error propagation in a machine vision-based precision landing system. This paper proposes a set of strategies to reduce the impact on the guidance system, and ultimately reduce the error. The next two studies approach the estimation and filtering problem from the deep learning approach, where error is a function to be minimized by learning. The last case of the collection deals with a trajectory classification problem with real data. This work completes the two main fields in deep learning, regression and classification, where the error is considered as a probability function of class membership.

Resumen

Los vehículos aéreos no tripulados (UAV) en sus versiones de pequeño tamaño conocidos como drones, van tomando protagonismo en las sociedades actuales. Los sistemas que los componen presentan multitud de retos entre los cuales el error se puede considerar como el denominador común. La percepción del entorno se mide mediante sensores que tienen error, los modelos que interpretan la información y/o definen comportamientos son aproximaciones del mundo y por consiguiente también presentan error. Explicar el error permite extender los límites de los modelos deterministas para abordar problemas del mundo real. El rendimiento de las tecnologías embarcadas en los drones, dependen de nuestra capacidad de comprender, modelar y controlar el error de los sistemas que los integran, así como de las nuevas tecnologías que puedan surgir.

Los controladores de vuelo integran diferentes subsistemas los cuales generalmente son dependientes de otros sistemas. Un caso de esta situación son los sistemas de guiado. Estos sistemas son los encargados de proporcionar al controlador de los motores información necesaria para cumplir con una misión deseada. Para ello se componen de una ley de control de guiado que reacciona a la información percibida por los sistemas de percepción y navegación. El error de cualquiera de estos sistemas se propaga por el ecosistema del controlador siendo vital su estudio.

Por otro lado, entre las estrategias para abordar el control del error se encuentran los estimadores en espacios de estados, donde el filtro de Kalman desde su aparición en los años 60, ha sido y continúa siendo un gran aliado para los ingenieros. Los filtros de Kalman son el corazón de los sistemas de fusión de información, los cuales minimizan la covarianza del error del sistema, permitiendo filtrar los estados medidos y estimarlos cuando no se tienen observaciones. Los modelos de espacios de estados se desarrollan en base a un conjunto de hipótesis para modelar el mundo. Entre las hipótesis se encuentra que los modelos del mundo han de ser lineales, markovianos y que el error de sus modelos ha de ser gaussiano. Generalmente los sistemas no son lineales por lo que se realizan linealizaciones sobre modelos que a su vez ya son aproximaciones del mundo. En otros casos el ruido que se desea controlar no es gaussiano, pero se aproxima a esta distribución para poder abordarlo. Por otro lado, multitud de sistemas no son markovianos, es decir, sus estados no solo dependen del estado anterior, sino que existen otras dependencias que los modelos de espacio de estados no son capaces de abordar.

Esta tesis aborda un compendio de estudios sobre los que se formula y reduce el error. En primer lugar, se estudia el error en un sistema de aterrizaje de precisión basado en visión por computador. Después se plantean problemas de estimación y filtrado desde la aproximación del aprendizaje profundo. Por último, se estudian los conceptos de clasificación con aprendizaje profundo sobre trayectorias. El primer caso del compendio estudia las consecuencias de la propagación del error de un sistema de aterrizaje de precisión basado en visión artificial. En este trabajo se propone un conjunto de estrategias para reducir el impacto sobre el sistema de guiado, y en última instancia reducir el error. Los siguientes dos estudios abordan el problema de estimación y filtrado desde la perspectiva del aprendizaje profundo, donde el error es una función que minimizar mediante aprendizaje. El último caso del compendio aborda un problema de clasificación de trayectorias con datos reales. Con este trabajo se completan los dos campos principales en aprendizaje profundo, regresión y clasificación, donde se plantea el error como una función de probabilidad de pertenencia a una clase.

Contents

List of Figures

List of symbols

Here, the symbols that appear in the thesis equations are listed. The first three blocks of the symbol list refers to general notational terms of the thesis "sets and spaces", "operators and functions", as well as "vectors and matrices". The remaining blocks of these list are divided into eleven specific sections corresponding to the order of appearance in the chapters of the thesis. The aim of this layout is to make easier for the reader the mathematical sections of these thesis. The blocks into which it is divided are “Navigation refence frames”, “ECEF-Geodetic relations”, “Attitude, representation and kinematics”, “Main Kalman notation”, “Sensor Fusion Variables”, “Camera Refence frames”, “Pinhole model”, “Camera calibration”, “Visual Object Tracking”, “Artificial Neural Networks”, and “Classification/validation”.

Sets and spaces	
$\mathbb{R}, \mathbb{C}, \mathbb{Z}, \mathbb{H}$	The set of Real, Complex, integer and quaternions
$\mathbb{S}^n$	Topological unit sphere, $\mathbb{S}^n = \{x \in \mathbb{R}^{n+1} : x^T x = 1\}$
$SO(n)$	Special orthogonal group, $\mathbb{S}^n = \{\boldsymbol{R} \in \mathbb{R}^{n,n} : \boldsymbol{R^T R} = \boldsymbol{I}_n, det(\boldsymbol{R}) = 1$
$so(n)$	Lie Algebra of $SO(n)$
$\mathbb{R}^n$	The set of real-valued column vectors of size n.
$\mathbb{R}^{n \times m}$	The space of $n \times m$ dimension of Real numbers.
Operators and functions	
$\circ$	Composition operator
$\times$	Cross-product
$\lvert a \rvert$	Absolute value of scalar a.
$\lVert \bar{p} \rVert$	L2 norm for the vector $\bar{p}$, as $\lVert \bar{p} \rVert = \sqrt{\mathrm{x}^T \mathrm{x}}$
e^x	Exponential function
$\sin \alpha, \cos \alpha, \tan \alpha$	Trigonometric functions (sine, cosine, and tangent of alpha)
$\operatorname{atan} a$	Inverse tangent function of a.
$arg\,min\, f\,(x)$	Value that minimizes $f(x)$
$\lim_{x \to n} f(x)$	Limit of function $f(x)$ wen x trend to n.
$\dot{A} = \frac{dA}{dt}$	Time evolution of A
$\nabla f(x)\lvert_a$	Jacobean matrix of f in specific a conditions.
$\frac{\partial f(x,t)}{\partial x}\lvert_a$	Partial x-derivation of function f in specific a conditions
Vectors and matrices	
a	“a” scalar value
$\bar{a}$	“a” column vector
$\bar{a}^*$	Conjugate of $\bar{a}$ vector
$\boldsymbol{q}, \boldsymbol{q_\alpha}$	Quaternion $\boldsymbol{q} \in \mathbb{H}$ and rotation in Euler α −axis expressed in quaternion
q_i	i-component of quaternion $\boldsymbol{q}$
$\boldsymbol{u}^i_{\{1,2,3\}}$	Local unitary vector in $\{i\}$ frame

$\mathcal{B}_i^j \in \mathbb{R}^3$	Orthonormal base of $\{j\}$ frame, expressed in $\{i\}$ coordinates
$R_i^j \in \mathbb{R}^{3\times3}$	Change-of-basis matrix from $\{i\}$ to $\{j\}$ reference frame
A	A matrix
$A_{[i,j]}$	The ith row, jth column element for the matrix A
$\mathbb{I}_{n\times n}$	Identity matrix of dimension $n \times n$.
$0_{n\times m}$	Matrix of dimension $n \times m$ whose all elements are equal to 0
$1_{n\times m}$	Matrix of dimension $n \times m$ whose all elements are equal to 1
A^T	Transpose of matrix A
$\lfloor \bar{a} \rfloor_\times$	Skew-symmetric matrix compose by $\bar{a}$ vector
$\det(A)$	Determinant of A matrix
$\mathrm{tr}(A)$	Trace of A matrix
$\mathrm{diag}(A)$	Diagonal matrix whose diagonal elements are given by x.
Navigation reference frames	
$\{b\}$	Body reference frame
$\{e\}$	Earth-Centered Earth-Fixed frame (ECEF)
$\{g\}$	Geodetic Coordinate system
$\{u\}$	Local Tangent Plane
$\{n\}$	East-North-Down (NED)
$\{en\}$	East-North-Up (ENU)
$\{s\}$	Sensor reference frame
ECEF-Geodetic relations	
x, y, z	Cartesian $\mathbb{R}^3$ coordinates.
λ, φ, h	Geodetic (longitude, latitude and height)
φ'	Geocentric latitude
r_e	Equatorial radium
r_p	Polar radium
$\varepsilon, \varepsilon'$	First and second eccentricity
f	Flattening factor defined
r_λ	Earth curvature
Attitude, representation, and kinematics	
Φ	Attitude representation.
$\boldsymbol{v}$	Rotation of rigid body
$\boldsymbol{u}$	Rotation unit axis
ϕ	Rotation relative to the unit quaternion
R_i^j	Direction Cosine Matrix
ϕ, θ, ψ	Euler angels: $\mathrm{roll}, \mathrm{pitch}, \mathrm{yaw}$
$R(i, \alpha)$	α-Rotation around "i" axis
$\boldsymbol{q}$	Quaternions
$\dot{\boldsymbol{q}}$	Quaternion rate vector
$[\dot{\psi}, \dot{\theta}, \dot{\phi}]^T$	Angular rates expressed as the Euler angles
Δt	Time step
Main Kalman notation	
x_k	Current state vector
z_k	Current measure vector
x_{k-1}	Previous state vector
$\hat{x}_{k\|k-1}$	Current state vector in prediction step
$\hat{x}_{k\|k}$	Current state vector in update step
u	Input system signal/control signal

$f(x,u,w)$	Dynamic System
$h(x,v)$	Stochastic measure function
A	Linear system matrix (continues system)
B	Input matrix (continues system)
H	Observation matrix model
F	State transition matrix (discreet system)
Γ	Input matrix (discrete system)
Q	The covariance of the process noise.
Q_p	Uncertainty in predictions.
Q_u	Projects the errors of the inertial sensors to the state vector.
R	The covariance of the observation measurements
P	Covariance matrix (measure of the estimate accuracy)
K	Optimal Kalman gain
$W_k \sim \mathcal{N}(0, Q_k)$	Process noise
$V_k \sim \mathcal{N}(0, R_k)$	Observation noise
Sensor Fusion Variables	
$\overline{x} \in \mathbb{R}^{IMU+GNSS}$	Fusion INS/GNSS state vector
$\bar{x}_{IMU} \in \mathbb{R}^{IMU}$	INS state vector
$\overline{x}_{GNSS} \in \mathbb{R}^{GNSS}$	GNSS state vector
$\bar{p}^{\{i\}} = (x^i, y^i, z^i)^T$	Position in $\{i\}$ reference frame and 3D-componts
$\bar{r}^e$	Position vector to local origin
$\bar{v} \in \mathbb{R}^3$	Velocity vector
$\dot{E}, \dot{N}, \dot{U}$	Ground Velocity in ENU reference frame
$\boldsymbol{q}$	Attitude representation in quaternions
$\bar{b}_a = \left(b_{a_x}, b_{a_y}, b_{a_z}\right)^T \in \mathbb{R}^3$	Accelerometer bias
$\bar{b}_\omega = \left(b_{\omega_x}, b_{\omega_y}, b_{\omega_z}\right)^T \in \mathbb{R}^3$	Gyroscope bias
$b_h \in \mathbb{R}$	Barometer bias
$\overline{z}_j^{\{i\}} \in \mathbb{R}^3$	Measurements of "j" sensor expressed in $\{i\}$ reference frame
$\bar{\omega} \in \mathbb{R}^3$	Angular rate vector
$\boldsymbol{\omega} \in \mathbb{H}$	Angular rate vector expanding in Hamilton space
$\overline{g} \in \mathbb{R}^3$	Gravity vector
$\overline{m} \in \mathbb{R}^3$	Earth magnetic field vector
$\overline{n}_j \in \mathbb{R}^3$	Gaussian white noise of "j" sensor
$\sigma_{j_i}^2 \in \mathbb{R}^3$	Variance in "i" component error of "j" sensor.
$C_i^j[k]$	$\{i\} \rightarrow \{j\}$ frames conversion matrix at k-time
$A[k]$	Attitude transition matrix INS/GNSS fusion
$U[k]$	Control input- (correction in velocity) INS/GNSS fusion
$\bar{W}[k]$	Observation noise process in loosely coupled architecture.
$\bar{\mathrm{V}}[k]$	System process noise in loosely coupled architecture.
$\mathrm{q}_{\mathrm{am}}^2$	Accelerometer noise for covariance prediction
$\mathrm{q}_{\omega\mathrm{m}}^2$	Rate gyro noise for covariance prediction
$\mathrm{q}_{\mathrm{ap}}^2$	Process noise for IMU accel. bias prediction
$\mathrm{q}_{\omega\mathrm{p}}^2$	Process noise for IMU rate gyro bias prediction
Camera Reference frames	
$\{w\}$	Real World reference frame
$\{c\}$	Camera reference frame
$\{ph\}$	Projective plane
Pinhole model	

A	Intrinsic camera matrix (pinhole model).
$k_1, \dots, k_n \in \mathbb{R}^n$	Set of radials, tangential and prims distortion parameters
$\delta \in \mathbb{R}^{3+n}$	Set of intrinsic camera parameters. Pinhole basis model plus distortions.
θ	External camera parameters.
$\bar{p}^i$	Points-position expressed in $\{i\}$ reference frame
$\Psi(\delta, \theta, \bar{p}^w)$	Camera model. Pinhole extension by intrinsic and extrinsic parameters.
Z	Image plane or focal plane
f_i	Focal distance in i −axis
c_x, c_y	Principal points
$\mathcal{F}^c$	Center of the camera
$(u, v)^T$	Coordinates in 2D projective plane.
${}^jT_i = [R\|\bar{t}] \in \mathbb{R}^{4\times4}$	Homogeneous transformation $\{i\}$ to $\{j\}$ reference frames
$R \in \mathbb{R}^{3\times3}$	Rotation matrix
$\bar{t} \in \mathbb{R}^3$	Translation vector
Camera calibration	
$\hat{E}$	Reprojection error
$O(\bar{p}_i^w) \in \mathbb{R}^2$	Calibration position pattern of element i
$\mathrm{C} \in \mathbb{R}^{2\times\mathrm{n}}$	Corner set of the calibration pattern
Visual Object Tracking	
ST_0	Short-term tracker
ST_1	Short-term tracker with conservative updating
LT_0	Pseudo long-term trackers
LT_1	Re-detecting long-term tracker
Artificial Neural Network	
φ	Activation Function.
σ	Sigmoid function
$\tanh$	Hyperbolic tangent function
$\max(0, z)$	ReLU activation function
$\max(0.01\mathrm{z}, \mathrm{z})$	Leaky ReLU activation function leaky=0.01
w_{kj}	k-neuron weight in j-th synaptic
b_k	k-neuron bias
L_n	n-layer
$W^{(l)}$	Weight matrix of l-layer
$b^{(l)}$	Base matrix of l-layer
$h(t)$	hidden nodes in RNN
$\hat{\mathrm{F}}^+$	Filtering
$\hat{\mathrm{F}}$	Prediction
$\boldsymbol{\Phi}$	Dataset
Φ_i	i-Data package
Z_i	i-Raw data package
X_i	i-Ideal data package
$\mathcal{L}$	Cost function/ learning
$\widehat{\boldsymbol{F}}_{\boldsymbol{\theta}}$	Network function
λ	Regularization factor
Classification validation	
Tp, Tn	True positive and true negative
Fp, Fn	False positive and false negative

Acronyms and Abbreviations

Acronyms	Description
AEE	Average Euclidean Error
AESA	European Union Aviation Safety Agency
AGE	Average Geometric Error
AHRS	Attitude and heading reference system
AI	Artificial Intelligence
AIS	Automatic Identification Systems
ANMS	Attention-based non maximum suppression
ANN	Artificial Neural Networks
API	Application Programming Interfaces
ARMA	Autoregressive Moving Average
ARMAX	Autoregressive-Moving Average with exogenous terms
AUC	Area under the ROC curve
BERT	Bidirectional encoder representations from transformer
BI-LSTM	Bidirectional-LSTM
BRIEF	Binary Robust Independent Elementary Features
CNN	Convolutional Neural Networks
CP	Checkpoints
CSEFMLP	Cost-Sensitive Cross-Entropy Error Function for MLP
CSRT	Channel and Spatial Reliability Tracker
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DCM	Direction Cosine Matrix
DL	Deep Learning
DOF	Degrees of freedom
DPM	Dimension-based Partitioning and Merging clustering
DTW	Dynamic Time Warping
ECEF	Earth-Centered Earth-Fixed frame
EKF	Extended Kalman Filter
EKF1	PX4 extended Kalman filter for position control
EKF2	Specific PX4 INS/GNSS fusion system
EKF3	Specific PX4 INS/Vision fusion system
ENU	East-North-Up
FAO	Food and Agriculture Organization
FAST	Features from Accelerated Segment Test
FLANN	Fast Library for Approximate Nearest Neighbors
Fn	False negative
FOV	Field of view
Fp	False positive

FPS	Frames per second
FSOD	Few-Shot Object Detection
GAN	Generative Adversarial Networks
GCS	Ground Control Station
GIAA	Applied Artificial Intelligence Group
GNSS	Global Navigation Satellite System
GOTURN	Generic Object Tracking Using Regression Networks
GPS	Global Position System
GRU	Gated Recurrent Units
GT	Ground Truth
HITL	Hardware In The Loop
HMM	Hidden Markov Models
HMSE	Half Means Square Error
HOG	Histograms of Oriented Gradients
HTOL	Horizontal Take-Off Landing
ICAO	International Civil Aviation Organization
IMM	Interacting Multiple Model filter
IMU	Inertial measurement unit
INS	Inertial Navigation System
IoU	Intersection Over Union
IP	Internet Protocol
IPPE	Infinitesimal Plane-Based Pose Estimation
IUU	Illegal, Unreported and Unregaled
KCF	Kernelized Correlation Filter
KF	Kalman Filter
KNN	K-Nearest Neighbors
LDS	Linear Dynamic Systems
LIDAR	Light Detection and Ranging o Laser Imaging Detection and Ranging
LSTM	Long-Short Term-Memory
LTP	Local Tangent Plane
MAD	Mean of Absolute Differences
MAPE	Mean Absolute Percentage Error
MARG	Magnetic, Angular rate, and Gravity
MAV	Micro Air Vehicle
MAVLink	Micro Air Vehicle Communication protocol
MHR	Mean Hit Ratio
MIL	Multiple Instance Learning
MIT	Mean Inference Time
ML	Machine Learning
MOSSE	Minimum Output Sum of Squared Error
MOT	Multiple Object Tracking
MSE	Means Square Error
MTT	Multi Target Tracking
MUAV	Micro Unmanned Aerial Vehicle
MVI	Motion Vector Integrator

NAV	Nano Air Vehicle
NCS	Neural Compute Stick
NED	East-North-Down
NED	North-East-Down
NLP	Natural Language Processing
NMS	Non-Maximum Suppression
NN	Neural Network
ORB	Oriented FAST and Rotated BRIEF
OSPA	Optimal Subpattern Assignment Error
PAV	Pico Air Vehicle
PnP	Perspective-n-Point
PSD	Power spectral density
PSP	Post-Synaptic Potential
PWM	Pulse-width modulation
PX4	Open-source flight control software developed by Dronecode foundation
QGC	QGroundControl
RANSAC	Random Sample Consensus
R-CNN	Region-based Convolutional Neural Networks
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
RKF	Robust Kalman filter
RMS	Root Mean Squared
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
ROI	Regions of Interest
RPAS	Remotely Piloted Aircraft Systems
RPI	Raspberry Pi 2 Model B
RTL	Return to lunch
RUS	Random Undersampling
SAD	Sum of Absolute Differences
SAM	Segment Anything Model
SARIMA	Seasonal Autoregressive Integrated Moving Average Model
SD	Standard deviation
SDt	Smart Dust
SESAR	Single European Sky ATM Research
SIFT	Scale-Invariant Feature Transform
SITL	Software in the loop
SLAM	Simultaneous Location And Mapping
SMOTE	Synthetic Minority Over-sampling Technique
SOT	Single Object Tracking
SSD	Single-Shot Detector/Sum of Squared Differences
SSM	State Space Models
SURF	Speeded-Up Robust Features

SVM	Support Vector Machin
TCP	Transmission Control Protocol
T-FoT	Trajectory Functions on Time
TLD	Tracking, Learning, Detection
Tn	True negative
Tp	True Positive
UAS	Unmanned aerial system/ Unmanned Aircraft Systems
UAV	Unmanned Aircraft Vehicle
UC3M	Carlos III University of Madrid
UDP	User Diagram Protocol
UKF	Unscented Kalman Filter
URM	Uniform Rectilinear Motion
VMS	Vessel Monitoring Systems
VO	Visual Odometry
VOT	Visual Object Tracking
VPU	Vision Processing Unit
VTOL	Vertical Take-Off and Landing
WCE	Weighted Cross-Entropy
WGS84	World Geodetic System 84
XML	Extensible Markup Language
YOLO	You Only Look Once
EP	Equilibrium Point

Introduction

The common domain of this thesis is error reduction in drone operations support systems. But what is the meaning of error in the context of drone operations support technologies? According to the Oxford dictionary[8], an error is "*a mistake, especially one that causes problems or affects the outcome of something*". This definition is too general, but it is useful to conceptualize in the fields of mathematics, science, and technology. In these fields, two types of error are distinguished: systematic error and random or stochastic error. The first, systematic error, is defined in statistics as the error that occurs in the same way in all measurements. It is also known as bias. On the other hand, random error is the error that cannot be precisely predicted and is outside the rules of determinism and requires alternative mathematical tools. This type of error is widely studied from different fields such as signal theory or control theory, but always under the umbrella of statistical tools.

This is approached in three different ways. On the one hand, the study of the error of a precision landing system based on the integration of information from a vision system over the drone navigation system. On the other hand, the estimation problem for trajectory tracking. Finally, the error in trajectory classification problems.

As far as a precision landing system is concerned, it is important to note that landing is undoubtedly one of the most common, essential, and critical maneuvers of any aircraft, including drones. Machine vision systems play a fundamental role in new landing strategies. Although there are a variety of applications that address this problem, we will focus on applications that rely on integrated aircraft systems. Undoubtedly, the most direct solution is to identify the target and let the aircraft's guidance and control system do the work to get the landing area. But what happens if the target (landing area) estimation is noisy? Inevitably, this error will propagate through the system and affect landing accuracy, but how?

From the point of view of estimation theory, the study of Kalman filter (KF) estimators allows the identification of the limits of these systems. The limitations of KFs are mainly associated with the starting assumptions of these models. This motivates the study of alternative approach based on data. Machine learning seeks to extract knowledge from the information contained in the data. With the knowledge acquired from the data, models are generated to explain its behavior. For this purpose, tracking problems based on state space

[8] https://www.oxfordlearnersdictionaries.com/

models are addressed from a machine learning approach. The aim is to generate dynamic models capable of estimating and filtering measurements. On this side, Deep Learning (DL) has shown in the last decades that it is able to address the modeling of nonlinear, non-Markovian and non-Gaussian systems. This provides an ideal opportunity to address the limitations of traditional Kalman filter-based estimation systems.

Finally, machine learning data classification problems are strongly supported by information theory, where concepts such as cross-entropy play a fundamental role in the definition of error for deep learning-based approaches. Furthermore, the evaluation of these models requires methods that quantify the error in classification, thus requiring new error definitions.

Motivation and research questions

The first contribution of this thesis, (Chapter 4), focuses on the study of the error of a vision-based precision landing system and its impact on the trajectory. The vision system is embedded in a drone with a zenithal field of view and takes information from the navigation system to estimate a global position of the landing area. These estimates are sent to the guidance system to reach a target. The estimation of the position of the landing platform is done with the context information of the platform to land on. A pinhole camera model is used to perform this estimation. The pinhole camera model is a projective model that can be adapted to the non-linear reality of non-paraxial optics by using aberration models of different types. The estimation of the landing zone position represents an error model to be modeled. But what is the influence on the landing and the trajectory? Also, knowing the influence of this error, is it possible to propose alternative landing strategies? These are the questions addressed in this first paper.

The second contribution of this thesis, Chapter 6, faces the challenge of generating an estimation and filtering model that can be compared to a Kalman filter (KF) using Deep Learning (DL). It presents a case where theory says that the best possible estimator is the KF. This work identifies the challenges of modeling with neural networks with regression problems in which it is desired to introduce nonlinear and non-Markovian components. For this reason, networks with long and short term memory, known by the acronym LSTM (Long Short Term Memory), play a fundamental role in this work.

LSTM networks are capable of modeling non-Markovian temporal behaviors, in other words, with long-term dependencies. Some particularities of LSTM cells, such as the forgetting gate inside each of their units, allow to ignore new measurements against estimations, which can be understood as a filtering since the LSTM cell will give more weight to the estimation than to the measurement. In a way, we can say that these networks are able to learn which measurements are important for the estimation and which are not according to previous states.

Artificial Neural Networks (ANN) operate in a bounded space with the training information, so it is essential to study the solution space. This space is bounded by the training information. However, real systems may not be bounded. One of the first questions that comes up is: What happens outside the boundary, how can we estimate outside the boundary space? To understand what happens at the borders of this space is the subject of the first article. The first article presents a recursive methodology of solution space transformation that allows one to tackle estimation problems of higher complexity from the classical point of view. For classical tracking systems based on Kalman filters, it is common to test the systems in the absence of measurements, so the question is how will LSTM networks behave in the absence of new measurements? These questions are also the subject of this second study.

The third contribution, Chapter 7, deepens on the formulation of the estimation and filtering problem. Specifically, from the deep learning approach. In addition, it delves into the understanding and formulation of LSTM cells and deep networks composed of these cells. The learning process of the networks is a critical moment, so a good definition of the learning process as an optimization process is crucial for a correct behavior. This motivates the study of the optimization processes and addresses classical challenges such as overfitting.

Throughout the third research article, three case studies of state estimation are compared in which the last one refers to a highly nonlinear system. The three case studies are approached as tracking systems, where dynamic models define the behavior of the trajectories of the system states. The goal is to estimate and filter such states even when measurement is lost. In the cases of linear systems, it is known that KF is the optimal estimator, but can the networks be compared? and in the cases where the trajectories are

governed by nonlinear behavior, what happens? These questions will be answered in the second article.

From the classical approach, process noise in fusion systems gives the Kalman filter in general and the estimation model in particular some flexibility in the face of uncertainty in the behavior of the dynamics of the state vector to be filtered. Regarding estimating vehicle kinematics, this means that if a vehicle describes a uniformly accelerating motion and the estimation model corresponds to a uniformly linear motion, the process noise can give the fusion system (KF), some flexibility to follow it successfully. This raises questions such as: what happens if the trajectories change with respect to the estimator model? what is the limit of the model change of the trajectories where the Kalman filter continues to work? and for the networks, what is its limit? These questions try to evaluate the robustness of the proposed neural estimators with respect to classical approaches.

In the final contribution, Chapter 8, classification systems are studied in the face of highly unbalanced real data sets. In the real data domain, it is common to find information bias, i.e. more data belonging to one class than to others. This drastically affects classical machine learning strategies by biasing the classification models. Although data balancing strategies exist, they modify the observations space, so the search for learning-based strategies represents a great opportunity. The classification approaches that stand out in the current literature focus on deep learning, where the main trends are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), with LSTMs being the most mentioned. But how does this bias influence the main neural architectures? can it be inferred in the learning process to address the information bias? is there a difference in classifying kinematic data with LSTMs and CNNs? which ones? These are some of the questions that this research aims to answer.

In summary, this thesis proposes as a principal objective:

1) Define and prototype data fusion-based navigation systems.
2) Define and prototype video analysis subsystems.
3) Design and prototype interpretation/reasoning subsystems.

4) Design and prototype deep learning methodologies for estimation and filtering.

To achieve these objectives, a series of secondary objectives are proposed:

I. Study indoor-outdoor UAV navigation techniques and mission planning.

II. To review in the literature problems associated with modeling dynamic systems with deep learning approaches.

III. To study estimation and filtering techniques.

IV. Explore advanced classification techniques with real data.

V. Explore hyper-realistic Software/Hardware In The Loop (SITL/HITL) simulation environments for experimentation with UAVs.

VI. Define models and systems that allow extracting information from the flight context.

VII. To study sensor fusion and new deep learning architectures.

Methodology

This thesis is the result of two different methodological processes. On the one hand the methodology of training the doctoral student to acquire the necessary skills for research and on the other hand the research methodologies that have been used to achieve the research objectives.

The research training methodology includes a rigorous scientific training, which is framed within the PhD Program in Computer Science and Technology of the University Carlos III de Madrid (UC3M). While working on the research project, the training associated with the PhD program in Computer Science and Technology of Universidad Carlos III de Madrid, aimed at improving research skills and ensuring the scientific quality of the research work, has been followed, including several types of training:

Specific education

A set of seminars and attendance to research conferences have been taken. On the one hand, there are the courses organized by the Computer Science and Engineering Department of University Carlos III de Madrid, on relevant research topics within the area of Computer

Engineering, given by prestigious visiting professors, especially from international universities. In addition, PhD students have participated in the research results presentation conferences, in which they present the state of development of their theses to other PhD students and researchers of the UC3M Computer Science and Engineering Department.

Transversal education

The transversal training complements the previous training through the acquisition of common skills for the development of scientific skills and for the improvement of the future professional career. This training is composed by different activities (short courses, seminars, etc.), which have been recognized by the academic committee. Among the courses and transversal trainings carried out are courses in time management and mentoring.

In addition, as a complementary activity it has participated in teaching activities such as teaching support in the subjects of "Data analysis", "Physical principles of computer science". Also, it has been involved in the direction of different final degree projects and in the development of projects associated with international students' stays. This provides a different approach that helps the sintering of knowledge to be explained and therefore a powerful tool in the work of scientific dissemination.

Regarding research, the methodology is framed in the scientific method in which the formulation of research problems is motivated by the special interest in DL paradigms under the context of the associated main project, using quantitative methods, generally experimental, analytical, or descriptive.

To achieve each of the objectives involved to the aim project "*Contribuciones a las tecnologías Habilitadores para la gestión de aeronaves no tripuladas y soporte a las operaciones*" and to elaborate the reference contributions, a methodological structure of 6 stages was followed: 1) Review of the state of the art to know and understand the work of other authors. 2) Problem formulation. 3) Proposal. 4) Experimental design. 5) Result evaluation. 6) Conclusions.

Thesis structure

In addition to the introduction and conclusions, a backbone of the thesis is the central block that divided into two parts. The first part, "Part I: Drones, Navigation and Vision-based precision landing" is composed by four chapters, the first three introduce the reader to concepts and tools that culminate with the publication in Chapter 4. The second part, "Part II: Deep Learning, forecasting, filtering and classification", is composed by other four chapters, in which the first one, Chapter 5, is an introduction to neural networks, the next two, Chapter 6 and Chapter 7, correspond to two contributions related to filtering and state estimation from the deep learning approach and the last one, Chapter 8, address de classification problem as the last contribution.

The aim of this structure is to provide the reader with the concepts and tools necessary to understand the referenced contributions. In addition, the Chapters 4, 6, 7 and 8 associated with the four reference contributions keep the structure and order of the original publications in order to highlight these works to the reader.

Part I: Drones, Navigation and Vision-based precision landing

Throughout this first part, general concepts about drones, navigation, computer vision, and finally precision landing are introduced. Specifically:

- **Chapter 1: Overview on drones**

Any study or development of technologies first requires knowledge of their history, terminology, classification, fundamentals, legislation and finally a starting point for the study and/or development. Although there are many commonalities between the different types of UAVs, this chapter considers the flight controller as the starting point. The flight controller system requires several essential subsystems such as guidance or navigation to function. In order to develop new applications based on the above-mentioned subsystems, it is necessary to have development and validation strategies that minimize risk, cost, and time. Hyper-realistic simulation is a powerful tool to minimize previous goals and accelerate the development of new applications.

Throughout this chapter the above topics are presented in 4 sections: Section 1.1. gives a general introduction from UAVs to drone terminology including drone definition and classification. Section 1.2. introduces the flight controller together with a brief introduction of control theory concepts. Section 1.3 can be considered a continuation of the previous sections but focused on vehicle guidance. Section 1.4 focuses on hyper-realistic simulation systems and different simulation platforms for drones.

- **Chapter 2: UAS INS/GNSS Navigation**

Inside the flight controllers the navigation system is responsible to estimate the necessary states required by the control system, guidance system, or other secondary subsystems. To do this, the navigation system uses sensors to sense the environment and algorithms to estimate the desired states. Position, velocity, acceleration, and orientation are the most common states that need to be estimated by the navigation system. The information fusion systems are presented as one of the most widespread technologies thanks to its angular piece, the Kalman filter. However, these systems require common reference frames for the information to be coherent, it is not always possible to observe the desired states, the measurements and models are not deterministic or simply the measurements of the different sensors are asynchronous.

This chapter introduces the fundamentals and mathematical tools necessary to understand, develop, and research on the basic INS/GNSS navigation systems embedded in drones. The chapter is divided into four sections, Section 2.1: Introduction to navigation, Section 2.2: Reference frame systems, Section 2.3: Attitude representation and mathematical tools, Section 2.4: Fusion of Inertial Navigation Systems and Global Navigation Satellite System, and the fundamentals of estimation and filtering with Kalman filters are introduced.

- **Chapter 3: Machine Vision Systems of UAS**

Although it is difficult to select a set of branches related to computer vision-based UAS applications, this chapter provides an overview of fundamentals and strategies that were considered outstanding for the development of the reference publications of this thesis, as well as for the further development of new vision-based drone applications. The AirSim simulation environment has a key role, so information related to the configuration of this system is included.

The chapter is divided into five sections. Section 3.1. introduces machine vision systems for UAS. Section 3.2. presents the fundamentals of modeling a camera and calibration. Section 3.3. explores the image stabilization problem from mechanical and computational approaches. Section 3.4. focuses on object detection in images, with a review of the state of the art in this branch of computer vision, describing the classical problems faced by researchers, the most widely used evaluation metrics, and a brief example of deep learning-based detection. Finally, Section 3.5. introduces the reader to the visual object tracking problem.

- **Chapter 4: Error Reduction in Vision-Based Multirotor Landing System**

This chapter corresponds to the third reference contribution of this thesis. Therefore, the chapter keep the original structure of the article. The fundamentals detailed in the previous chapters have been used for this chapter. The chapter show a scientific article structure in

which it initially shows the authors, an abstract and keywords. It is then organized as follows: First, Section 4.1. contains an introduction, Section 4.2. shows the problem formulation of helipad position estimation by monocular computer vision system. Section 4.3. describes the landing strategy proposal and the global estimation module. The correction module design, the analysis of the complete landing system, and a description of the test environment can be found in Section 4.4. Finally, the conclusions are presented in Section 4.5. References for this work are given in Section 4.6.

Part II: Deep Learning, forecasting, and filtering.

This second part is composed of an introductory chapter and two specific chapters associated with the first two references of this thesis.

- **Chapter 5: Artificial Neural Networks.**

The study of Artificial Neural Networks (ANN) involves a variety of biological, psychological, mathematical, physical, and computational foundations. The purpose of this chapter is to provide an overview of neural networks, their foundations, the current trends in the literature, and finally to introduce the reader to the terminology used in the following chapters.

This chapter is divided into 3 sections: The first section, Section 5.1, provides a historical introduction to contextualize the progress. Section 5.2. presents the bioinspired mathematical concept and its formulation. Section 5.3. introduces the network concept, the topology, and the potential of the most prominent current networks. Finally, the references are given.

- **Chapter 6: An approach to forecasting and filtering noise in dynamic systems using LSTM architectures.**

This chapter corresponds to the first reference contribution of this thesis. The original structure of the journal article has been kept except for the authors' biographies, which have been deleted. The fundamentals detailed in the previous chapters and Part I, have been used here. The chapter presents a scientific article structure in which it initially shows the authors, an abstract and keywords. It is then organized as follows: First, Section 6.1. contains an introduction, Section 6.2. introduce the mathematical problem formulation. Section 3 shows the database, structure, and pre-processing. Section 6.4. shows the LSTM neuro-estimator model, general process, and training parameters. The description and results of the numerical experiments are summarized in Section 6.5. Finally, the conclusions are presented in Section 6.6. followed by references.

- **Chapter 7: Forecasting nonlinear systems with LSTM: Analysis and comparison with EKF.**

This chapter corresponds to the second reference contribution of this thesis. The original structure of the journal article has been kept. The concepts detailed in the previous Part I, chapters and publications have been used for this chapter. The chapter presents a scientific article structure in which it initially shows the authors, an abstract and keywords. It is then organized as follows: First, Section 7.1. introduction, Section 7.2. defines the problem and introduces how to approach the problem from a classical observer point of view, as well as reviewing possible solutions to estimation and filtering problems from deep learning paradigms. Section 7.3. is a description of the study proposal, the methodology for its realization, and a rigorous mathematical definition. Section 7.4. details the three case studies based on the proposed approach and the proposed experiments. Finally, Section 7.5. presents the conclusions. References for this work are given in Section 7.6.

- **Chapter 8: LSTM vs CNN in real ship trajectory classification**

This chapter corresponds to the fourth reference contribution together with the journal article that is currently in production. Therefore, the chapter keep the original structure of the articles. The chapter show a scientific article structure in which it initially shows the authors, an abstract and keywords. It is then organized as follows: First, Section 8.1. contains an introduction, Section 8.2. provides an overview of similar work on ship trajectory classification. Section 8.3. describes the methodology, data structures, classical approaches deep leaning approaches and learning problem. The experiments and results are presented in Section 8.4. Finally, the conclusions are presented in Section 8.5.

Part I: Drones, Navigation and Vision-based precision landing

Chapter 1: Overview on drones

1.1. Introduction

Unmanned Aerial Vehicles (UAVs) emerged in the context of World War I (1914-1918), where they were used for air defense training by the British Army. These early vehicles demonstrated their high military potential, but the lack of sensing, navigation, and communications technologies at the time limited their applications. During the interwar period (1918-1939), advances in telecommunications made remote control possible. In addition, new video technologies enabled the first on-board video systems and infrared sensing. All these new technologies were used during World War II (1939-1945), where the most widespread concept was the remote control of aircraft behind enemy lines. The new control theories allowed the development of the first autopilot systems. In the second half of the 20th century, space exploration, the Cold War, and various wars such as Vietnam and Iran-Iraq motivated the use of these vehicles for espionage, nuclear testing, and precision warfare. At the end of the 20th century and especially at the beginning of the 21st century, the applications of UAVs ceased to be exclusively military and began to be used in the civilian area. Some of the UAV applications are focused on search and rescue, fire control, agriculture, geology, archaeology, or recreational use.

An Unmanned Aerial System (UAS) is a system consisting of the UAV, a ground station, communications systems, and the pilot who performs the missions. The inclusion of the pilot in the UAS ecosystem allows them to be referred to as Remotely Piloted Aircraft Systems (RPAS). Depending on the application, these aircraft can vary widely in size, weight, and shape, but as a minimum requirement they must be reusable and capable of flying remotely or autonomously; for these two reasons, ballistic missiles, projectiles, and torpedoes cannot be considered UAVs [1]. However, loitering munitions [2] or kamikaze drones have the possibility of being recovered if they do not find a target, so in a sense they are a hybrid concept between a vehicle and a munition. According to the International Civil Aviation Organization (ICAO), UAVs weighing less than 25 kg are called "drones".

UAVs can be classified by wingspan and weight, type of lift, type of wing, and the applications for which they are designed (military, civil, commercial, or recreational). Other metrics used for classification according to [3] are range, endurance, and altitude. As for wingspan and weight in [4], they are UAV, maximum wingspan of 61m and a maximum weight of 15.000 kg; Micro Unmanned Aerial Vehicle (MUAV), with a wingspan of 1-2m and a weight

of 2-5Kg; Micro Air Vehicle (MAV), 15cm-1m and 50g-2Kg; Nano Air Vehicle (NAV), 2cm-15cm and 3g-50g; Pico Air Vehicle (PAV), 0. 25-2.5cm and 0.5-3g; finally, vehicles between 1mm-0.5cm and a weight of 0.005-0.5g are known as Smart Dust (SDt).

As for the classification by wing type, there are two main types, fixed-wing, and rotary-wing drones. The first type of drones have their wings attached to the rest of the aircraft and the propulsion system is independent of them. For takeoff and landing, they require longitudinal runways where the vehicle can vary its speed and the wings generate lift. They are also known as Horizontal Take-Off Landing (HTOL). These vehicles typically have a greater range than rotorcraft, but they cannot land vertically, and they cannot perform static flights; they must always be in motion for the wings to keep them in the air. This is a limitation to the range of operations in complex environments. Rotary wing drones rely on the motion of the wings, coupled to the propulsion system, to provide lift. This gives them the ability to take off and land vertically, known as Vertical Take-Off and Landing (VTOL). They are characterized by having several rotors that counteract the torque produced by each of them. In addition, they can be subdivided into a main rotor and a tail rotor, two rotors in a coaxial configuration, or multirotor. Multirotor drones usually have three or more rotors. The most common multirotor are the quadcopters, hexacopters, and octocopters. Increasing the number of rotors increases the payload capacity of the drone, but also increases its weight and power consumption. In addition to the number of rotors, there are different geometric configurations in which the rotors can be in the same or different planes. These configurations are typically designed to increase payload, stability, and reliability in the event of motor failure. Combining the advantages of both types of drones is possible through hybrid or convertiplanes models [5]. These types of hybrids or convertiplanes generally seek to integrate VTOL with the payload capacity and autonomy of fixed-wing drones.

The continuous emergence of drone applications, motivated by new business opportunities in the context of smart cities, is driving the regulation of the vehicles and their operation. European countries have opted for a set of regulations, concepts and systems called U-Space [6]. This regulation is led by the European program SESAR (Single European Sky ATM Research). For example, according to AESA (European Union Aviation Safety Agency) [7], the current regulations stipulate that drone flights must always be within the pilot's field of vision, must not exceed a height of 120 m, and must not fly over people or vehicles. They also establish flight exclusion zones to ensure the safety of infrastructure, other vehicles, and people. For example, it is forbidden to fly within 8 km of an airport or airfield. In addition, persons and vehicles wishing to operate in European airspace must be registered with the EASA. This legislation exempts NAVs if they are not equipped with cameras.

1.2. Flight controller

In the survey on open-source flight platforms by E. Ebeid et al. [8], the challenges faced by civil drone applications based on UAV platforms are mentioned. Among them are the difficulties in standardizing flight controller architectures. One of the emerging platforms is the Pixhawk series [9]. The integrated software corresponds to PX4 [10]. It is only one part of the DroneCode collaborative project [11], which also includes the ground station QGround Control [12].

Flight controllers, such as PX4, are systems that consist of aircraft control, navigation, and guidance. They may also include other systems such as avoidance. Aircraft control is responsible for keeping the aircraft stable in the air. It is based on control theory, which studies the stability of a dynamic model of the vehicle. The most general description of vehicle dynamics is based on state space models. This allows the stability of the system to be translated into a desired situation. For this, the flight controller needs a control law for the system, a reference input to specify e.g. the desired position, a navigation system to locate the aircraft states, and a control output to tell the engines the force to apply. In addition, control models typically introduce a disturbance model into their control loop. A disturbance model can make the controller more robust to situations such as drag from air currents or changes in air density among others. Fig. 1.1 shows a general schematic of a UAS control system.

To accomplish the required mission, the flight controller works continuously during a flight. In order for these algorithms to work, an environmental perception system is required. Several navigation sensors are used to provide the vehicle's position, velocity, and orientation, as well as a dynamic model of the sensors' behavior and estimation. The sensors typically operate at different frequencies so that the dynamic model of the navigation system allows for estimation when no measurements of the vehicle states are being taken. When new measurements are received, the navigation system merges them with the estimates and filters out the noise.

The general concept of control theory is to minimize the error of the system states. To do this, the error of the desired reference is calculated relative to the current states. One of the classical ways to do this is to study the dynamics of the error. There are a variety of strategies, with the study of the eigenvalues (poles of the system) being one of the most widely used methods for linear systems. Knowing the error dynamics allows to identify the influence of the reference on the stability and therefore to apply gains that move the poles of the system to positions of desired stability. An exhaustive development of these control theories can be consulted in classic references such as K. Åström and B. Wittenmark [13].

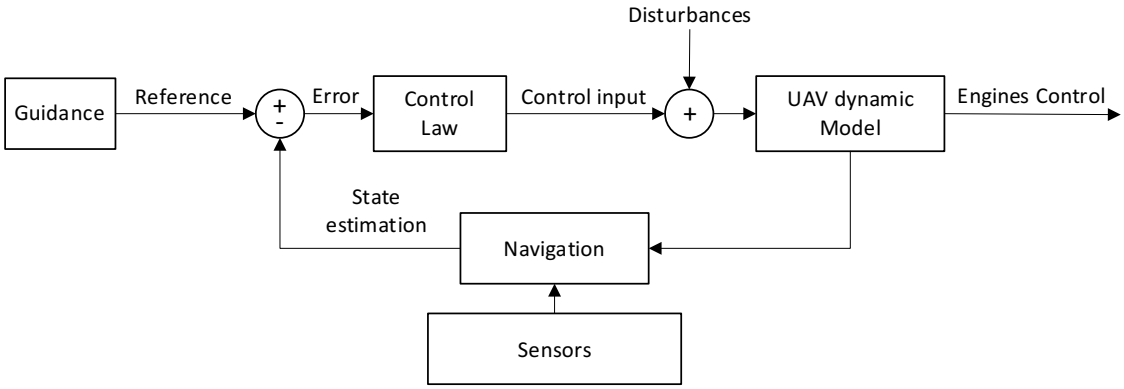


Fig. 1.1. Basic UAV control system.

To facilitate the study of dynamic systems and their differential equations, the Laplace transform [14] is generally used to transform a real variable function such as time "t" into a complex variable "s". This transform allows the n-derivative of t, $\frac{d^n y}{dt^n}$, to be represented by s^n and thus helps to study the dynamics of linear systems in a phase space.

To define stability is essential for control systems, therefore the most generalized concepts of stability are those associated with the Routh stability [15] and, in a more generalized form, the Lyapunov stability theorem [15]. The Routh stability criteria study the eigenvalues of the dynamical system belonging to a complex space in which the abscissa axis defines the real part and the ordinates the complex part. According to the Routh theorem, a system is stable if all its eigenvalues belong to the left half-plane. In addition, the influence on the behavior of the system will be different according to the value of the eigenvalues of its components. As for the Lyapunov stability theorem, the general idea is to prove that the solutions of the difference equation for initial conditions x_0 remain close to x_0 for all subsequent times. This requires finding a positive definite function known as the Lyapunov function. Although control and stability theory is out the scope of this thesis, it is worth mentioning that the Lyapunov stability theorem includes Routh stability for the case of linear systems.

1.3. Guidance

The guidance system is responsible for providing the control system with the reference signal to execute a given trajectory to complete a mission. In the context of drone flight controllers, guidance can consist not only of reaching a position, but also of finding a threshold distance to that position, a speed equal to that of the target, or even intercepting a target. There are three main phases to any drone mission: take-off, flight, and landing. Takeoff and landing are essential maneuvers and therefore have their own guidance systems, generally

minimalist unless precision landing systems are available. On the other hand, during the flight phase, the guidance system is essential as it is responsible for achieving or executing the maneuvers previously planned at the ground station. Some of the maneuvers in which the guidance system is involved include reaching a specific area, executing a specific trajectory such as a search pattern, following a target, and others. In the PX4 flight controller, the embedded guidance system is known as L_1 and is governed by a nonlinear control law based on virtual targets introduced by Park et al. [16].

1.4. Simulation

UAS are powerful systems that face difficult challenges when operating in highly complex environments. For example, operating in an urban environment requires ensuring the integrity of the aircraft, other vehicles, infrastructure, and the safety of people. To develop new applications such as logistics, surveillance, transportation, and others, higher levels of autonomy are needed for long-range flights, with requirements for endurance, reliability, and fault tolerance [8]. To ensure a safe transition to the new technologies, simulation systems are needed to contextualize the missions and validate the applications. Software In The Loop (SITL) and Hardware In The Loop (HITL) simulations are a safe and reliable way forward. The basic idea of SITL and HITL is to replace the information provided by the real environment, the real vehicle, and the real sensors with simulations to test algorithms or develop new applications [17], [18]. Powerful simulation engines like JSBSim [19], JMAVsim[20], FlightGear [21], Matlab UAV Toolbox [22], Gazebo [23] or AirSim [24] are used for this purpose.

If the physics of the environment, vehicle, and sensors are already done on the simulator, the flight controller algorithms can be included in the simulation loop. When the algorithms are run on the simulation computer, it is called SITL. When the algorithms of the flight controller or system under test are embedded in external physical devices, such as Pixhawk, the simulation is in HITL.

Since the environments where the simulation and the algorithms are executed are different, communication protocols such as UDP (User Diagram Protocol) and TCP (Transmission Control Protocol) or IP (Internet Protocol) are required.

The main idea of simulation in SITL and HITL is to provide a safe transition of new applications to the final physical device. Therefore, the ground station is often included in the simulation ecosystem.

The efforts of the scientific community to standardize communication with UAS in small drones have led to the development of the MAVLink protocol (Micro Air Vehicle Communication protocol) [25]. This allows the development of multiple platforms under a common communication framework. To help develop new applications using these protocols,

there are Application Programming Interfaces (API), such as the MAVSDK [26], that make this task easier.

1.5. References

[1] U. MoD, "Joint doctrine note 2/11 the UK approach to unmanned aircraft systems," *UK MoD The Development, Concepts and Doctrine Centre, SWINDON, Wiltshire, 2011*.

[2] M. Voskuijl, "Performance analysis and design of loitering munitions: A comprehensive technical survey of recent developments," *Def. Technol.*, vol. 18, no. 3, pp. 325–343, Mar. 2022, doi: 10.1016/J.DT.2021.08.010.

[3] N. Elmeseiry, N. Alshaer, and T. Ismail, "A Detailed Survey and Future Directions of Unmanned Aerial Vehicles (UAVs) with Potential Applications," *Aerosp. 2021, Vol. 8, Page 363*, vol. 8, no. 12, p. 363, Nov. 2021, doi: 10.3390/AEROSPACE8120363.

[4] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Prog. Aerosp. Sci.*, vol. 91, pp. 99–131, May 2017, doi: 10.1016/J.PAEROSCI.2017.04.003.

[5] G. J. J. Ducard and M. Allenspach, "Review of designs and flight control techniques of hybrid and convertible VTOL UAVs," *Aerosp. Sci. Technol.*, vol. 118, p. 107035, 2021, doi: 10.1016/j.ast.2021.107035.

[6] "U-Space | droneuropa." https://www.droneuropa.com/U-Space/ (accessed May 20, 2023).

[7] "Guide - drone operators | EASA." https://www.easa.europa.eu/en/light/topics/guide-drone-operators (accessed May 20, 2023).

[8] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, and U. P. Schultz, "A survey of Open-Source UAV flight controllers and flight simulators," *Microprocess. Microsyst.*, vol. 61, pp. 11–20, Sep. 2018, doi: 10.1016/J.MICPRO.2018.05.002.

[9] Pixhawk, "Pixhawk series." https://pixhawk.org/products/ (accessed May 20, 2023).

[10] "Open Source Autopilot for Drones - PX4 Autopilot." https://px4.io/ (accessed Feb. 22, 2022).

[11] "The Dronecode Foundation - We are setting the standards in the drone industry with open-source - Join the Community!" https://www.dronecode.org/ (accessed May 20, 2023).

[12] "QGC - QGroundControl - Drone Control." http://qgroundcontrol.com/ (accessed Mar. 29, 2022).

[13] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, Third Edit. Mineola, New York: Dover Publications.

[14] M. Olivi, "The Laplace transform in control theory," *Lect. Notes Control Inf. Sci.*, vol. 327, pp. 193–209, Mar. 2006, doi: 10.1007/11601609_12/COVER.

[15] J. Zabczyk, "Mathematical Control Theory," Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-44778-6.

[16] S. Park, J. Deyst, and J. P. How, "A New Nonlinear Guidance Logic for Trajectory Tracking".

[17] AirSim, "AirSim SITL PX4." https://microsoft.github.io/AirSim/px4_sitl/

[18] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” *Springer Proc. Adv. Robot.*, vol. 5, pp. 621–635, 2018, doi: 10.1007/978-3-319-67361-5_40.

[19] “JSBSim Open Source Flight Dynamics Model.” https://jsbsim.sourceforge.net/ (accessed May 20, 2023).

[20] “jMAVSim with SITL | PX4 User Guide.” https://docs.px4.io/main/en/simulation/jmavsim.html (accessed May 20, 2023).

[21] “FlightGear Flight Simulator – sophisticated, professional, open-source.” https://www.flightgear.org/ (accessed May 20, 2023).

[22] MATLAB, “UAV Toolbox.” https://es.mathworks.com/products/uav.html (accessed Apr. 25, 2023).

[23] “Gazebo.” https://gazebosim.org/home (accessed May 20, 2023).

[24] “Home - AirSim.” https://microsoft.github.io/AirSim/ (accessed Feb. 22, 2022).

[25] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, “Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey,” *IEEE Access*, vol. 7, pp. 87658–87680, 2019, doi: 10.1109/ACCESS.2019.2924410.

[26] “Introduction · MAVSDK Guide.” https://mavsdk.mavlink.io/main/en/index.html (accessed Mar. 13, 2022).

Chapter 2: UAS INS/GNSS Navigation

2.1. Introduction

Paul D. Groves [1] defines navigation *"as any of several methods of determining or planning a ship's or aircraft's positions and course by geometry, astronomy, radio signals, etc*". Meanwhile, the Oxford Learner's Dictionary [2], define tracking as *"to find somebody/something by following the marks, signs, information, etc., that they have left behind them"* or *"to follow the movements of somebody/something, especially by using special electronic equipment".* In some ways, navigation can be considered as a self-tracking system. Regarding navigation systems, P. Groves [1] refer them as "a device that determines position and velocity automatically". The methods to determine these kinematic variables or states of some body/something are defined as navigation techniques [1].

The most common UAS navigation technique is based on sensor information fusion [1], [3], which rely on the concept of improving the estimation of a variable/state using information from different sources instead of a single one. To use this technique, it is necessary to address previous challenges:

First, the sensors used to make the observations provide data which can be expressed in different reference systems (frames), so it is required to have a common reference frame where all the information is coherent. An example can be found in the information provided by a Global Navigation Satellite System (GNSS) and a distance sensor embedded on the same vehicle. Both systems provide vehicle position measurements, but in the first case, GNSS, refers to a global reference frame (the Earth global position) and the second to a local one.

The second challenge is the representation of the orientation (or attitude) of the vehicle with respect to the common reference system. For this, it is necessary to define the attitude, how to express it and its temporal evolution. In physics this problem is studied in classical mechanics under the name of estimating the pose of a "rigid body" [4], [5]. The third challenge for information fusion, focuses on modelling the errors inherent in the measurements of every sensor, since the measurement processes present a stochastic behavior (noise) and systematic deviations (sensor bias) that requires statistical treatment and modeling.

The technique to join the information which is detailed in this chapter is based on the estimation theory of stochastic processes, specifically on the Kalman filters (KF) and its variant for nonlinear systems, the extended Kalman filter (EKF). These techniques are extensions of

deterministic methods (estimation theory) based on SSM that model the dynamic behavior of a system (in our case the motion of a UAS) and minimize the state error using a feedback loop. In the case of KF, the covariance of the system is minimized, so it is an optimal method as explained in J.P. Llerena et al. [6]. This information fusion technique aims to obtain a vector with the vehicle states, which can be at a higher frequency than the sensor updates.

As mentioned, a problem that needs to be solved by navigation systems is the estimation of sensor bias. The measurements of local sensors on-board such as angular velocity or acceleration in the body frame present biases that are projected onto the global reference frame by nonlinear transformations. Thus, the state vector initially composed of kinematic states such as position and velocity, needs to be extended to estimate the bias of the sensors to correct them from the measurements.

Finally, these techniques of sensor fusion have a multitude of adjustable parameters that modify the filter operation and impacts the navigation result. Therefore, it is essential to fine tuning the KF parameters (basically the models assumed for the errors appearing both in the measurements and predictions) to achieve the best navigation for the used sensors. This adjustment is beyond the scope of this chapter but is addressed in recent submitted papers.

2.2. Reference Frame Systems

The observations (measurements) of states such as position or velocity are always made from the sensor's reference frame. However, in the navigation problem it is necessary to know the states in the vehicle's reference frame, which is usually called body, denoted with the letter $\{b\}$. In order to be able to transform state measurements between different reference frames, mathematical tools are required to relate measurements between reference frames.

It is possible to classify two types of reference systems: local and global. Global reference systems are those that allow an object to be unequivocally identified in the whole representation of the planet, in our case the Earth.

Two of the most popular global reference systems are the Earth-Centered Earth-Fixed frame (ECEF) $\{e\}$ and the Geodetic Coordinate system $\{g\}$. First one, ECEF, is formed by a set of three orthogonal axes located at the center of the Earth (geocentric coordinates) where the z_e axis is aligned with the geographic north and the x_e,y_e axes define the plane of the equator. The second global system is supported by the approximation of the earth's surface to a geodetic geometry where the positions are unequivocally characterized by the angles with respect to the ECEF axes $\{x^e, y^e, z^e\}$, expressed as longitude, " λ", latitude "φ" and height "h" above the geodetic surface.

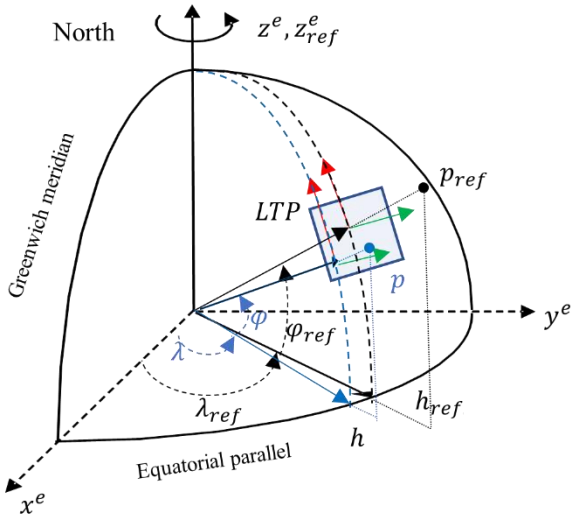


Fig. 2.1. Relation between, local and global reference frame and relation between geodetic latitude, height and ECEF coordinates.

Local reference frames are those that express measurements in specific regions. Generally, Cartesian frames are used to define them on specific regions of the Earth's surface, using tangent planes defined on the geodetic surface called Local Tangent Plane coordinate system (LTP) $\{u\}$. According to the orientation of the axes, the most common reference frames systems are East-North-Down (NED) $\{n\}$ and East-North-Up (ENU) $\{en\}$. As their names indicate, the Cartesian axes correspond to each of the geographical directions. The tangent point on the geoid is called the reference point $\bar{p}_{ref} = \left(\lambda_{ref}, \varphi_{ref}, h_{ref}\right)^T$ and is expressed in global coordinates. In navigation systems reference point, it is usually taken as the starting point of missions. Fig. 2.1 is included to help the reader.

Finally, sensors also have their own reference frames called Sensor reference frame $\{\mathrm{s}\}$. There are different type of sensors so specific nomenclature is needed for each one, such as $\{\mathrm{giro}, \mathrm{acc}, \mathrm{baro}, \ldots\}$. The sensors are usually referred to the vehicle main axes, also named as body frame.

Throughout this section we summarize the mathematical expressions that allow switching between local and global reference systems to generate a common reference frame. For navigation problems the common reference frame is usually considered the vehicle's gravity center $\{\mathrm{b}\}$ which corresponds to a local reference frame.

2.2.1. Global frames (WGS84 and ECEF) and local frame at tangent point ENU and NED

The geodetic frame $\{\mathrm{g}\}$ expresses the position with respect to the reference ellipsoid WGS84 (World Geodetic System 84) [3], containing the coordinates: geodetic longitude, geodetic latitude, and geodetic height $(\lambda, \varphi, \mathrm{h})$.

WGS84 model is a simple ellipsoidal model, whose parameters are its semi-axes r_e (Equatorial radium), r_p (Polar radium) and eccentricity, defined as $\varepsilon = \sqrt{1 - \frac{{r_p}^2}{r_e^2}}$, and ε' second eccentricity of the ellipsoid. Alternatively, the flattening factor defined as $f = \frac{r_e - r_p}{r_e}$. The reference constants for these values are $r_e = 6378137$ meters, $f = 1/298.257223563$.

The ECEF frame $\{e\}$ has the center in the ellipsoid, Z axis parallel to polar axis and X, Y axes included in the equatorial plane, pointing to meridians respectively at longitude of 0 and $\pi/2$ radians. ECEF is used as intermediate frame from WGS-84 to local Cartesian frame at Earth surface.

It is important to differentiate the geocentric coordinates, referred to the ECEF system, from the geodetic coordinates, referred to as the geodetic model (WGS84). This difference is provided by the geodetic model (datum) and is represented in the Fig. 2.2, where φ' refers to geocentric latitude and φ refers to geodetic latitude.

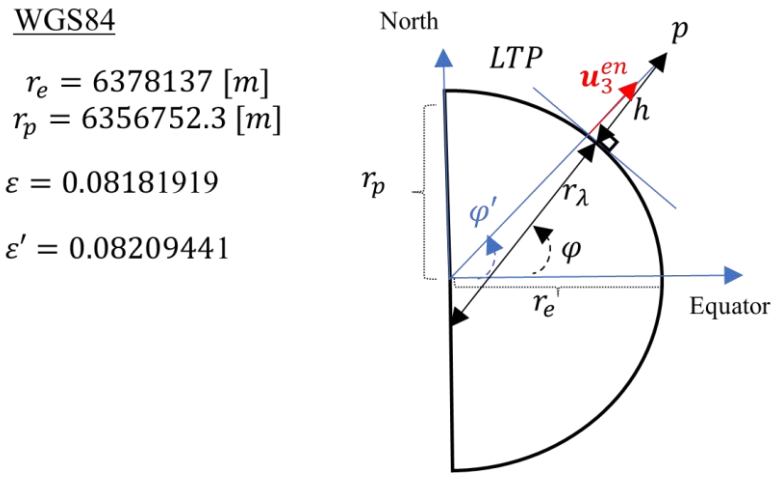


Fig. 2.2. Relationships among geocentric and geodesic coordinates.

The geodetic latitude and geodetic height are defined with respect to LTP, not with respect to the center of ellipsoid, requiring the appropriate conversions from geocentric to geodetic magnitudes.

2.2.2. Geodetic to ECEF transformation

Consider a position $p^g = (\varphi, \lambda, h)^T$ in WGS84 geodetic coordinates $\{g\}$ that we want to transform to the reference frame $\{en\}$. Considering the parameters of the WGS84 model, the transformation is defined as follows. In the first place, we compute the Earth curvature parameter for transformation:

$$r_\lambda = \frac{r_e}{\sqrt{1 - \varepsilon^2 \sin^2 \varphi}} \tag{2.1}$$

and the local vertical vector, $\boldsymbol{u}_3^{en}$, expressed in ECEF coordinates:

$$\boldsymbol{u}_3^{en} = \begin{bmatrix} \cos\varphi \cos\lambda \\ \cos\varphi \sin\lambda \\ \sin\varphi \end{bmatrix} \tag{2.2}$$

So, the transformed ECEF coordinates are given by:

$$\bar{p}^e = \begin{bmatrix} x^e \\ y^e \\ z^e \end{bmatrix} = \begin{bmatrix} r_\lambda \cos\lambda \\ r_\lambda \sin\lambda \\ r_\lambda(1-\varepsilon^2)\sin\varphi \end{bmatrix} + h\boldsymbol{u}_3^{ne} = \begin{bmatrix} (r_\lambda + h)\cos\varphi\cos\lambda \\ (r_\lambda + h)\cos\varphi\sin\lambda \\ \left((1-\varepsilon^2)r_\lambda + h\right)\sin\varphi \end{bmatrix} \tag{2.3}$$

2.2.3. ECEF to geodetic transformation

Considering a position $\bar{p}^e = (x^e, y^e, z^e)^T$ expressed in the ECEF reference frame $\{e\}$, it is desired to know this position $\bar{p}^g = (\varphi, \lambda, h)^T$ expressed in the geodetic global reference frame $\{g\}$.

Unfortunately, the conversion from ECEF to geodetic coordinates is not analytic, because the geodetic height is defined over the LTP, but at the same time the tangent point (geodetic latitude and longitude) depends on the local vertical axis. Here we use an approximation in order to avoid an expensive iterative process to compute this transformation.

First, geodetic latitude is approximated as a correction over geocentric latitude, φ' which has a direct analytical expression:

$$\begin{aligned} R &= \sqrt{(x^e)^2 + (y^e)^2 + (z^e)^2} \\ D &= \sqrt{(x^e)^2 + (y^e)^2} \\ \varphi' &= atan\frac{z^e}{D} \end{aligned} \tag{2.4}$$

Then, the geodetic latitude, φ, is approximated with the following sequence of terms:

$$
\begin{aligned}
f &= 1 - \varepsilon^2 \\
x_a &= \frac{(1-f)r_e}{\sqrt{(tan^2(\varphi) + (1-f)^2)}} \\
y_a &= (1-f)({r_e}^2 - {x_a}^2)^{\frac{1}{2}} \\
\mu_a &= atan\frac{({r_e}^2 - {x_a}^2)^{\frac{1}{2}}}{(1-f)\ x_a} \\
r_a &= \frac{x_a}{cos(\varphi)} \\
l &= R - r_a \\
\delta\lambda &= \mu_a - \varphi' \\
h &= l \cdot cos(\delta\lambda) \\
\rho_a &= \frac{(1-f)r_e}{\sqrt{1-(2f-f^2)\,sin^2(\mu_a)}} \\
\varphi &= \mu_a - atan\frac{l \cdot sin(\delta\lambda)}{\rho_a + h} \\
r_\lambda(\varphi) &= \frac{r_e}{\sqrt{1-\varepsilon^2\,sin^2\,\varphi}}
\end{aligned}
\tag{2.5}
$$

And, finally, the other coordinates, height, h and longitude, λ, are computed as usual, using the local ellipsoid curvature r_λ:

$$
\begin{aligned}
h &= \frac{D}{cos(r_\lambda)} - r_\lambda \\
\lambda &= atan\frac{y^e}{x^e}
\end{aligned}
\tag{2.6}
$$

2.2.4. ECEF to local Cartesian (ENU and NED) transformation

In navigation system, its typical consider by local Cartesian reference system the ENU $\{en\}$ and the NED $\{n\}$ reference frame system. Each of these reference systems has a different global orientation. In this section we consider the conversion of a position expressed in the reference frame ECEF $\{e\}$ coordinates $\bar{p}^e = (x^e, y^e, z^e)^T$ to local Cartesian coordinates $\bar{p}^{en/n} = (x, y, z)^T$ centered at reference origin $\bar{p}^g_{ref} = \left(\varphi_{ref}, \lambda_{ref}, h_{ref}\right)^T$. It uses the unitary the orthogonal vectors, $\boldsymbol{u}_i^{en}$, that make up the orthonormal base $\mathcal{B}_e^{en} = \{\boldsymbol{u}_1^{en}, \boldsymbol{u}_2^{en}, \boldsymbol{u}_3^{en}\}^e$ of local ENU frame, expressed in ECEF coordinates:

$$\boldsymbol{u}_1^{en} = \begin{bmatrix} -sin(\lambda_{ref}) \\ cos\,\lambda_{ref} \\ 0 \end{bmatrix}; \boldsymbol{u}_2^{en} = \begin{bmatrix} -\sin\varphi_{ref}\cos\lambda_{ref} \\ -\sin\varphi_{ref}\sin\lambda_{ref} \\ cos\,\varphi_{ref} \end{bmatrix}; \boldsymbol{u}_3^{en} = \begin{bmatrix} cos\,\varphi_{ref}\cos\lambda_{ref} \\ cos\,\varphi_{ref}\sin\lambda_{ref} \\ \sin\varphi_{ref} \end{bmatrix} \quad (2.7)$$

The relationship between the local ENU and NED systems is bidirectional and can be defined by the following transformation $R_{en}^{n} = R_{n}^{en} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$.

The orthogonal basis $\mathcal{B}_e^n \in \mathbb{R}^3$ of the local cartesian NED reference frame expressed in ECEF $\{e\}$ is defined as $\mathcal{B}_e^n = \{\boldsymbol{u}_1^n, \boldsymbol{u}_2^n, \boldsymbol{u}_3^n\}^e$. Specifically, each one of the vectors that make up the base are $\{\boldsymbol{u}_1^n = \boldsymbol{u}_2^{en};\ \boldsymbol{u}_2^n = \boldsymbol{u}_1^{en};\ \boldsymbol{u}_3^n = -\boldsymbol{u}_3^{en}\}$.

Then, the position vector with respect to local origin is computed,$\bar{r}^e$, as the vector difference between both positions in ECEF frame:

$$\bar{p}_{ref}^{e} = \begin{bmatrix} r_\lambda(\varphi_{ref})\ cos\,\lambda_{ref} \\ r_\lambda(\varphi_{ref})\ sin\,\lambda_{ref} \\ r_\lambda(\varphi_{ref})(1-\varepsilon^2)\ sin\,\varphi_{ref} \end{bmatrix} + h_{ref} \cdot \boldsymbol{u}_3^{en/n} \quad (2.8)$$

$$\bar{r}^e = \bar{p}^e - \bar{p}_{ref}^{e} \quad (2.9)$$

Finally, the local coordinates can be obtained with the projections of the position vector on each local unitary vector, even though all magnitudes are expressed in ECEF coordinates:

$$\bar{p}^{en} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}^e \cdot \boldsymbol{u}_1^{ne/n} \\ \boldsymbol{r}^e \cdot \boldsymbol{u}_2^{ne/n} \\ \boldsymbol{r}^e \cdot \boldsymbol{u}_3^{ne/n} \end{bmatrix} = R_e^{en/n} \cdot \bar{r}^e \quad (2.10)$$

where $R_e^{en/n}$ are the change-of-basis matrices (R_e^{en} and R_e^n) constructed with the eigenvectors of the means of the $\mathcal{B}_e^{en/e}$orthogonal base of ENU $\{en\}$ or NED $\{n\}$ reference frame, expressed in ECEF $\{e\}$ reference frame. These R_e^{en} and R_e^n matrices can be considered as an $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ transformation.

2.2.5. Local Cartesian (ENU or NED) to ECEF transformation

Given a position in local coordinates $\bar{p}^{en/n} = (x, y, z)^T$, in ENU $\{en\}$ or NED $\{n\}$ frame, referred to the plane reference origin $\bar{p}_{ref}^{g} = \left(\varphi_{ref}, \lambda_{ref}, h_{ref}\right)^T$ it is desired to know the coordinates of the position $\bar{p}^{en/n}$ in global coordinates ECEF, $\bar{p}^e$. If the reference origin plane is not expressed in global coordinates, it is necessary to determine the latitude, longitude, and height coordinates, (φ, λ, h), by applying the transformations expressed in (section "ECEF to

Geodetic"). The first step, is use the unitary orthogonal vectors, $\boldsymbol{u}_i^{en}$, (2.7) that make up the base $\mathcal{B}_e^{en}\{\boldsymbol{u}_1^{en}, \boldsymbol{u}_2^{en}, \boldsymbol{u}_3^{en}\}^e$ of local ENU frame, expressed in ECEF coordinates.

Then, the ellipsoid curvature is computed, at the geodetic latitude of local origin $r_\lambda(\varphi_{ref})$, as show in the end of collection (2.5).

So, the local origin is expressed in ECEF coordinates as show in (2.8).

Finally, the ECEF position, $\bar{p}^e$, is directly obtained with vectorial sum of local coordinates to origin, once everything is expressed in the global coordinates:

$$\begin{aligned} \bar{p}^e &= \bar{p}_{ref}^e + x\boldsymbol{u}_1^{en} + y\boldsymbol{u}_2^{en} + z\boldsymbol{u}_3^{en} \\ \bar{p}^e &= R_{en/n}^e \cdot \bar{p}^{en/n} + \bar{p}_{ref}^e \end{aligned} \tag{2.11}$$

$$\begin{aligned} R_{en/n}^e &= \left[\boldsymbol{u}_1^{en/n}, \boldsymbol{u}_2^{en/n}, \boldsymbol{u}_3^{en/n}\right] \\ R_{en}^e &= \begin{pmatrix} -\sin\lambda_{ref} & -\sin\varphi_{ref}\cos\lambda_{ref} & \cos\varphi_{ref}\cos\lambda_{ref} \\ \cos\lambda_{ref} & -\sin\varphi_{ref}\sin\lambda_{ref} & \cos\varphi_{ref}\cos\lambda_{ref} \\ 0 & \cos\varphi_{ref} & \sin\varphi_{ref} \end{pmatrix} \\ R_n^e &= \begin{pmatrix} -\sin\varphi_{ref}\cos\lambda_{ref} & -\sin\lambda_{ref} & -\cos\varphi_{ref}\cos\lambda_{ref} \\ -\sin\varphi_{ref}\sin\lambda_{ref} & \cos\lambda_{ref} & -\cos\varphi_{ref}\cos\lambda_{ref} \\ \cos\varphi_{ref} & 0 & -\sin\varphi_{ref} \end{pmatrix} \end{aligned} \tag{2.12}$$

where $R_{en/n}^e \in \mathbb{R}^{3\times3}$, it is the change-of-basis matrix from $\{en\}$ or $\{n\}$ to $\{e\}$ reference frame.

2.3. Attitude mathematical concepts

To describe the situation of a rigid body in 3D, classically six degrees of freedom are required: three describing the position of the center of masses and another three rotational degrees to describe the attitude (orientation of its axes).

Being able to express the attitude of a body in different reference systems is essential for navigation systems to find a common reference frame, since it is possible to have measurements that are made from one observer but need to be expressed in another reference system.

The mathematical basis on which the concept of attitude in navigation systems is based and developed are both Lie groups and Lie algebras, specifically the group of rotations SO(3) [4], [5]. This algebraic structure is framed in differential topology and can be understood as a group of transformations on a vehicle that is approximated as a rigid body (commonly referenced as differentiable manifold in navigation).

Throughout this section different types of attitude representation and its associated kinematics are described, emphasizing the representation with quaternions since it is used later in the sensor fusion Section 2.4.

2.3.1. Attitude representation

The attitude is the orientation of an object described mathematically with respect to a reference system. This mathematical representation refers to a set of parameters and transformations that associate the orientation of one reference system with another. Naming the reference systems A and B as F^A and F^B respectively, the attitude Φ expresses the orientation of one system relative to another:

$$\Phi: F^A \leftrightarrow F^B \tag{2.13}$$

The attitude of B relative to A can be represented as $\Phi_1: F^A \rightarrow F^B$ while the reverse $\Phi_2: F^B \rightarrow F^A$, so $\Phi_2 = \Phi_1^{-1}$.

There are different representations of the attitude, however there are 3 main representations that are the most extended in the literature navigation applications:

- Direction Cosine Matrix (DCM): Represents a transformation between two reference frames, so its algebraic sense is the change-of-basis matrix between two reference frames. For $\mathbb{R}^3$ spaces the dimension of the matrix is 3×3.
- Euler Angles: Describes the orientation of one reference system relative to another using a set of three rotations parameterized by three different angles. The typical notation used to define these angles in navigation systems is $\{\phi, \theta, \psi\} = \{roll, pitch, yaw\}$ [6].
- Quaternion: Mathematically are vectors with four components belonging to the Hamilton space, $\mathbb{H}$, (extension of the real space $\mathbb{R}^3$ like the complex numbers), which allows a compact representation of the object's attitude.

Each of these representations can be deduced under the concepts of Lie groups [4], [5] and their algebras. More information can be found in specific texts such as D. Sattinger [4] or F. Lachello [5].

2.3.1.1. Direction cosine matrix (DCM)

Consider a vehicle with its own reference frame $\{b\}$ that it is desired to align with north to work in the NED $\{n\}$ local reference frame. Also, consider that the sensor is placed at the gravity center of the vehicle where an angle with respect to north ψ is measured.

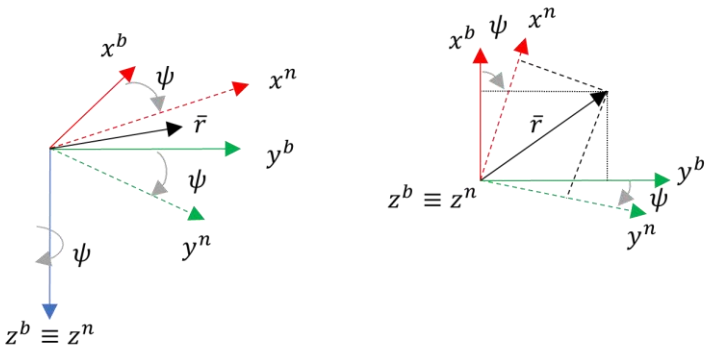


Fig. 2.3. Rotation of the z-axis. Geometric relation between the reference frame $\{b\}$ and $\{n\}$.

Let $\bar{r}^b = (x^b, y^b, z^b)^T$ and $\bar{r}^n = (x^n, y^n, z^n)^T$ the representations of the vector $\bar{r}$ expressed in $\{b\}$ or $\{n\}$, known ψ, the relationship between both reference frames can be geometrically deduced from Fig. 3.3 as:

$$\bar{r}^n = \begin{bmatrix} x^n \\ y^n \\ z^n \end{bmatrix} = \begin{bmatrix} x^b \cos\psi + y^b \sin\psi \\ -x^b \sin\psi + y^b \cos\psi \\ z^b \end{bmatrix} \tag{2.14}$$

Matrix can be expressed as:

$$\bar{r}^n = \begin{bmatrix} x^n \\ y^n \\ z^n \end{bmatrix} = \overbrace{\begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{R_b^n} \begin{bmatrix} x^b \\ y^b \\ z^b \end{bmatrix} \tag{2.15}$$

where R_b^n is a rotation matrix and is called "Direct Cosine Matrix" (DCM). This matrix can be interpreted algebraically as a change-of-basis matrix, where the columns are the unit vectors of the transformation, i.e., the projection of the axes of reference frame $\{n\}$ on $\{b\}$. The DCM matrix is composed by sines and cosines that relates the orientations between two reference systems. Specifically, (2.15) expresses a rotation of the z-axis and also can be expressed as $R(z, \psi)$. Taking three rotations to define the complete orientation of the body in $\mathbb{R}^3$ space, it can be used using the same geometric approximation as above together with the right-hand rule to define the rotation with the remaining x and y axes.

$$R_b^n(x, \phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}; \; R_b^n(y, \theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.16}$$

The properties of these matrices can be found in detail in R.M. Rogers [7]. The most important are:

- The columns of the DCM correspond to a set of vectors that form an orthonormal basis.
- The vector transformed about the rotation axis is invariant (Fig. 2.3 z-axis).
- The elements of which DCM is composed are sine and cosine functions.
- The cosines are placed on the principal diagonals of the DCM matrix.

The important mathematical properties of DCM are:

$$det(R) = 1; R^T = R^{-1}; R^T R = I \tag{2.17}$$

2.3.1.2. Euler angles

The Euler angles $\{\phi, \theta, \psi\}$, Fig. 2.4, is a parameterization form of the R rotations previously used in the DCM representation. Each of the angles denote the rotation about each $\{x, y, z\}$ axes and generally expressed in radians. In navigation these angles usually take the names "roll", “pitch” and “yaw".

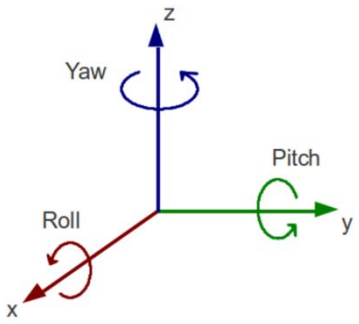


Fig. 2.4. Euler angles in ENU reference frame.

SO(3) is the group of transformations in which R belongs and is non-abelian [4], [5]. This means that it has no commutative property so the order of multiplication cannot be varied and corresponds to:

$$R_b^n = R(x, \phi)R(y, \theta)R(z, \psi) \tag{2.18}$$

In the grouped form the set of rotations is as follow:

$$R_b^n = \begin{bmatrix} cos\,\theta\, cos\,\psi & cos\,\theta\, sin\,\psi & -sin\,\theta \\ -cos\,\phi\, sin\,\psi + sin\,\phi\, sin\,\theta\, cos\,\psi & cos\,\phi\, cos\,\psi + sin\,\phi\, sin\,\theta\, sin\,\psi & sin\,\phi\, cos\,\theta \\ sin\,\phi\, sin\,\psi + cos\,\phi\, sin\,\theta\, cos\,\psi & -sin\,\phi\, cos\,\psi + cos\,\phi\, sin\,\theta\, sin\,\psi & cos\,\phi\, cos\,\theta \end{bmatrix} \tag{2.19}$$

In navigation, Tai-Bryan angles [6] are generally used under the name of Euler angles. The difference is that, from the vehicle reference system, a zero-inclination angle with respect to

the horizon is assigned in place of $\pi/2$. The equation is carried out in the following order of operation:

$$R_b^n = R(z,\psi)R(y,\theta)R(x,\phi) \tag{2.20}$$

Regardless of the convention used, the resulting matrix has the following properties:

- Rotation matrices are orthogonal.
- The determinant of the R matrix is unitary.
- Rotation matrices are not commutative (non-abelian group) $R_b^c R_a^b \neq R_a^b R_b^c$

To obtain the Euler angles starting from DCM, the transformation is obtained directly from the components of the matrix R as:

$$\theta = -sin^{-1} R_{13}\,, \phi = tan^{-1}\frac{R_{23}}{R_{33}}, \psi = tan^{-1}\frac{R_{12}}{R_{11}} \tag{2.21}$$

where the sub-indices i, j of $R_{i,j}$ means the rows and columns of the rotation matrix R.

However, if the angle of the intermediate rotation $R(y, \theta = \pm\pi\backslash 2)$ is fixed in (2.18) and (2.19):

$$R(\theta = \pi\backslash 2) = \begin{bmatrix} 0 & 0 & -1 \\ sin(\phi-\psi) & cos(\phi-\psi) & 0 \\ cos(\phi-\psi) & -sin(\phi-\psi) & 0 \end{bmatrix} \tag{2.22}$$

an indetermination called “Gimbal Lock” or “kinematic singularity” appears, in which a degree of freedom is lost. The value of $\phi - \psi$ can be known, but not the values of each of them independently, that's why it is a singularity and that's why a degree of freedom is lost.

2.3.1.3. Quaternions

Let the rotation $\boldsymbol{v}$ of a rigid body with an angle θ, such that $\theta = \|v\| \in \mathbb{R}$ expressed in radians around a unit axis $\boldsymbol{u} = \left[u_x, u_y, u_z\right]^T$, can be expressed as $\boldsymbol{v} = \boldsymbol{u} \cdot \phi$, where $\phi = \theta/2$ denotes the rotation relative to the unit quaternion [3], [8]. In exponential map format the series can be expanded according to Euler's formula:

$$e^{\boldsymbol{v}} = e^{\boldsymbol{u}\phi} = \sum_{k=0}^{\infty} \frac{1}{k!}(u\phi)^k = \underbrace{\left(1 - \frac{\phi}{2!} + \frac{\phi^4}{4!} + \cdots\right)}_{cos\,\phi} + \underbrace{\left(\boldsymbol{u}\phi - \frac{\boldsymbol{u}\phi^3}{3!} + \frac{\boldsymbol{u}\phi^5}{5!} + \cdots\right)}_{\boldsymbol{u}\,sin\,\phi} \tag{2.23}$$

Thus, the representation of the rotation $\boldsymbol{v}$ can be expressed as a 4-vector $\boldsymbol{q} \in \mathbb{H}$, Hamilton space [9], which is called a quaternion. From the previous expression it can be deduced that the 4-vector quaternion, can be expressed as:

$$\boldsymbol{q}_\theta = \left[cos\left(\frac{\theta}{2}\right), sin\left(\frac{\theta}{2}\right)\boldsymbol{u}\right]^T \tag{2.24}$$

If the expression (2.24) is compared with the Euler's formula, $e^{i\theta} = cos\,\theta + i\,sin\,\theta$, it is seen the quaternion $\boldsymbol{q}$ is a vector extension of Euler's formula. Thus, quaternions are another attitude representation and, as will be seen throughout this section, avoids the problem of the "gimbal lock" comment in Euler angle subsection.

A quaternion $\boldsymbol{q} = [q_0, q_1, q_2, q_3]^T$ is composed by four components $\boldsymbol{q}_i, i = \{0,1,2,3\}$. These four components typically divide in two sides $\boldsymbol{q} = [q_0, q_{1:3}]^T$. First component $\boldsymbol{q}_0 \in \mathbb{R}$ means the scalar side of the quaternion and the other three components $\boldsymbol{q}_{1:3} \in \mathbb{R}^3$ with the vector side.

Unitary quaternion is a quaternion whose norm $\|\boldsymbol{q}\| = 1$:

$$\begin{gathered}\|\boldsymbol{q}\| = \sqrt{\boldsymbol{q}\cdot\boldsymbol{q}^*} = \sqrt{q_0^2 + q_{1:3}\cdot q_{1:3}} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1 \Leftrightarrow \\ \Leftrightarrow q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1\end{gathered} \tag{2.25}$$

Topologically, the set of unit vectors in $\mathbb{R}^4$ denote a sphere $\mathbb{S}^3$, so a unit quaternion defines a rotation, $R = Rot(\theta, \boldsymbol{u})$, of the group $SO(3)$ [8], where $\boldsymbol{u} = \left[u_x, u_y, u_z\right]^T$ is the unit vector parallel to the rotation axis:

$$R = Rot(2\,cos^{-1}\,q_0\,, \frac{q_{1:3}}{\|q_{1:3}\|}) \tag{2.26}$$

The unit vector side $\frac{q_{1:3}}{\|q_{1:3}\|} = \boldsymbol{u}$ means a vector parallel to the axis of rotation, while the scalar side defines the rotation θ relative to the rotation axis.

As show in [10] the representation of the DCM matrix from the quaternion $\boldsymbol{q}$ is:

$$R(\boldsymbol{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{2.27}$$

If the quaternion is unitary, the previous expression can be simplified as:

$$\begin{gathered} -q_2^2 - q_3^2 = q_0^2 + q_1^2 - 1 \\ -q_1^2 - q_3^2 = q_0^2 + q_2^2 - 1 \\ -q_1^2 - q_2^2 = q_0^2 + q_3^2 - 1 \end{gathered} \tag{2.28}$$

In this way, the trace of the rotation matrix $R(\boldsymbol{q})$ is reduced:

$$R(\boldsymbol{q}) = 2\begin{bmatrix} q_0^2 + q_1^2 - 1/2 & q_1q_2 + q_0q_3 & q_1q_3 - q_0q_2 \\ q_1q_2 - q_0q_3 & q_0^2 + q_2^2 - 1/2 & q_2q_3 + q_0q_1 \\ q_1q_3 + q_0q_2 & q_2q_3 - q_0q_1 & q_0^2 + q_3^2 - 1/2 \end{bmatrix} \tag{2.29}$$

The inverse transformation, DCM to $\boldsymbol{q}$, can be deduce from previous matrix (2.29). Firstly, from the trace of the matrix (2.29) the scalar part of the quaternion q_0 can be solved.

$$\begin{aligned} Trace(R) &= \sum_{i=1}^{3} R_{ii} = R_{11} + R_{22} + R_{33} = 4q_0^2 - 1 \\ |q_0| &= \frac{1}{2}\sqrt{Trace(R) + 1} \end{aligned} \tag{2.30}$$

From the term R_{11} can be solved q_1:

$$\begin{aligned} R_{11} = q_0^2 + q_1^2 - \frac{1}{2} &= 2\left(\frac{1}{2}\sqrt{Trace(R) + 1} + q_1^2 - \frac{1}{2}\right) \\ |q_1| &= \sqrt{\frac{R_{11}}{2} + \frac{1 - Trace(R)}{4}} \end{aligned} \tag{2.31}$$

Using the same logic for the rest of the trace elements of $R(\boldsymbol{q})$ the remaining terms of q are solved:

$$\begin{aligned} |q_2| &= \sqrt{\frac{R_{22}}{2} + \frac{1 - Trace(R)}{4}} \\ |q_3| &= \sqrt{\frac{R_{33}}{2} + \frac{1 - Trace(R)}{4}} \end{aligned} \tag{2.32}$$

The relation between the Euler angles and quaternions can be found with the DCM-Euler relation (2.21):

$$\begin{aligned} \theta &= -sin^{-1}(2q_1q_3 - 2q_0q_2) \\ \phi &= tan^{-1}\left(\frac{2q_2q_3 + 2q_0q_1}{q_0^2 + q_1^2 - q_2^2 - q_3^2}\right) \\ \psi &= tan^{-1}\left(\frac{2q_1q_2 + 2q_0q_3}{q_0^2 + q_1^2 - q_2^2 - q_3^2}\right) \end{aligned} \tag{2.33}$$

Since the Euler angle representation can be expressed as an ordered composition of 3 rotations about the axes of the reference frame, (2.20), and since a rotation $R(\boldsymbol{q})$ can be expressed as a quaternion $\boldsymbol{q}$, the rotation matrix with Euler angles can be expressed as a composition of quaternions:

$$R_a^b = R_a^b(z, \psi)R_a^b(y, \theta)R_a^b(x, \phi) = \boldsymbol{q}_\psi \circ \boldsymbol{q}_\theta \circ \boldsymbol{q}_\phi \tag{2.34}$$

Were "∘" means the composition operator. This property says that the composition of two quaternions, $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ corresponds to a multiplication operation [4], [5]:

$$\boldsymbol{q} = \boldsymbol{q}_a \circ \boldsymbol{q}_b = (q_{a,1}q_{b,1} - q_{a,1:3} \cdot q_{b,1:3};\ \ q_{a,1}q_{b,1:3} + q_{b,1}q_{a,1:3} - q_{a,1:3} \times q_{b,1:3}) \tag{2.35}$$

Each of the quaternions in exponential map:

$$\boldsymbol{q}_i = \left[cos\left(\frac{i}{2}\right), sin\left(\frac{i}{2}\right)\boldsymbol{u_i}\right]^T = \left[cos\left(\frac{i}{2}\right) \quad 0 \quad 0 \quad sin\left(\frac{i}{2}\right)\right]^T | i = \{\phi, \theta, \psi\} \tag{2.36}$$

where the vectors u_i means the canonical Euler rotation basis vectors, i.e:

$$B_u = \{u_\phi, u_\theta, u_\psi\} = \begin{Bmatrix} u_x & u_y & u_z \\ \overset{\frown}{1} & \overset{\frown}{0} & \overset{\frown}{0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix} \tag{2.37}$$

Finally, multiplying the three quaternions associated to the Euler rotations using the expression (2.35), the values of the 4-vector quaternion $\boldsymbol{q}$ are given:

$$\boldsymbol{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} cos\left(\frac{\psi}{2}\right)cos\left(\frac{\theta}{2}\right)cos\left(\frac{\phi}{2}\right) + sin\left(\frac{\psi}{2}\right)sin\left(\frac{\theta}{2}\right)sin\left(\frac{\phi}{2}\right) \\ cos\left(\frac{\psi}{2}\right)cos\left(\frac{\theta}{2}\right)sin\left(\frac{\phi}{2}\right) - sin\left(\frac{\psi}{2}\right)sin\left(\frac{\phi}{2}\right)cos\left(\frac{\phi}{2}\right) \\ cos\left(\frac{\psi}{2}\right)sin\left(\frac{\theta}{2}\right)cos\left(\frac{\phi}{2}\right) + sin\left(\frac{\psi}{2}\right)cos\left(\frac{\phi}{2}\right)sin\left(\frac{\phi}{2}\right) \\ -cos\left(\frac{\psi}{2}\right)sin\left(\frac{\theta}{2}\right)sin\left(\frac{\phi}{2}\right) + sin\left(\frac{\psi}{2}\right)cos\left(\frac{\theta}{2}\right)cos\left(\frac{\phi}{2}\right) \end{bmatrix} \tag{2.38}$$

2.3.2. Attitude Kinematics

Navigation systems consider the kinematics of the vehicle reference systems, i.e., the temporal evolution of the vehicle states including attitude. The aim of this subsection is to focus on the temporal study of the different representations of the attitude kinematics, DCM, Euler angles and quaternions, as the basis of inertial navigation systems (INS). As a consequence of the "gimbal lock" with the Euler angles representation of the attitude, throughout the derivation of this representation the problem propagates and manifests itself as a mathematical singularity. The quaternion representation of the attitude solves this problem and therefore is of great relevance for navigation systems. For this reason, in this subsection, we will expand on the kinematics of quaternions as a basis for navigation systems in discrete time, as used in the fusion section.

2.3.2.1. DCM Kinematics

Taking the DCM representation of the attitude defined by the rotation matrix R, where R changes with time (angles vary with time) and considering the property $R \cdot R^T = \mathbb{I}$ of the SO(3) rotation group:

$$\frac{dI}{dt} = \frac{d(R \cdot R^T)}{dt} = \frac{dR}{dt} \cdot R^T + R \cdot \frac{d(R^T)}{dt} = \overbrace{\dot{R} \cdot R^T}^{\Theta} + \left(\dot{R} \cdot R^T\right)^T = 0 \tag{2.39}$$

From the previous equation it is given $\Theta^T = -\Theta$ so Θ is a skew-symmetric matrix.

$$\begin{aligned} &\dot{R} \cdot R^T = \Theta \\ &\dot{R} \cdot \overbrace{R^T \cdot R}^{\mathbb{I}} = \Theta \cdot R \\ &\dot{R} = \Theta \cdot R \end{aligned} \tag{2.40}$$

It can be deduced that Θ is the cross-product operator, $\times$, between the angular rate vector $\bar{\omega} = \left[\omega_x, \omega_y, \omega_z\right]^T$ and the orthonormal basis B^i of the starting reference frame, $\{i\}$.

$$\Theta(\bar{\omega}) = \bar{\omega} \times B^i = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix} \tag{2.41}$$

So, the time evolution $\dot{R}$, can be calculated with the vector of angular rates $\bar{\omega}$ and the rotation matrix R. From the navigation system point of view, it implies that to determine the time evolution of the attitude with DCM it is necessary to have a sensor to provide the angular rates. This information is given from a gyroscope.

2.3.2.2. Euler Angles

As for the representation with Euler angles, starting from a fixed reference frame which presents relative angular rates expressed as the Euler angles as $\left[\dot{\psi}, \dot{\theta}, \dot{\phi}\right]^T$ and using the angular velocity addition theorem [1], [7] it is possible to deduce the angular velocities in different reference frames. The angular velocity addition theorem says that, for angular velocity vector in a common reference frame, the angular velocity resulting from the rotations is a simple sum of the rotations it contributes.

$$\omega_Z^A = \omega_B^A + \omega_C^B + \cdots \omega_Z^Y \tag{2.42}$$

From (2.20) where the rotation matrix is the ordered composition of three orientations about a common reference frame and considering that each of the Euler angles vary in time, it can be calculated the angular velocity vector $\bar{\omega} = \left[\omega_x, \omega_y, \omega_z\right]^T$ as if it is measured in the rotation frame [3].

$$\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = R_a^b(y,\theta)R_a^b(x,\phi)\begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} + R_a^b(x,\phi)\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + \begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix} \tag{2.43}$$

Grouping the previous equation:

$$\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = \begin{bmatrix}1 & 0 & -\sin\theta\\ 0 & \cos\phi & \sin\phi\cos\theta\\ 0 & -\sin\phi & \cos\phi\cos\theta\end{bmatrix}\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix} \tag{2.44}$$

The inverse transformation is:

$$\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix} = \frac{1}{\cos\theta}\begin{bmatrix}\cos\theta & \sin\phi\sin\theta & \sin\theta\cos\phi\\ 0 & \cos\phi\cos\theta & -\sin\phi\cos\theta\\ 0 & \sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}\omega_x\\\omega_y\\\omega_z\end{bmatrix} \tag{2.45}$$

In the case of $\theta = \pm\frac{\pi}{2}$ the "gimbal-lock" singularity of Euler angles is again manifested, so generally for attitude kinematics another type of attitude representation is used in place of Euler angles.

2.3.2.3. Quaternions

Quaternion propagation refers to the accumulation of attitude over time in quaternion form and is found by integrating the $\boldsymbol{q}$ differential equation. Since there is no closed solution, approximate methods such as discrete numerical methods are used.

Δt is the time step which defines the discrete time as $t_n = n\Delta t$, where $n = 1,2,\ldots$ is a discrete set. Furthermore, it is generalized that the angular velocities measured by the gyroscope are also in discrete time $\bar{\omega}(t_n) = \left[\omega_x, \omega_y, \omega_z\right]^T$ in rad/s so the numerical approximation is generalized. The attitude in the local reference frame with quaternions at instant $t + \Delta t$ is defined as $\boldsymbol{q}(t + \Delta t)$, this means a variation of the quaternion $\Delta\boldsymbol{q}$ during the time variation Δt. The rotation is around the instantaneous axis $\boldsymbol{u} = \frac{\bar{\omega}}{\|\bar{\omega}\|}$, where the rotated angle is $\theta = \|\bar{\omega}\|\Delta t$. Thus, the variation of the quaternion can be expressed as:

$$\Delta\boldsymbol{q} = \cos\frac{\theta}{2} + \boldsymbol{u}\sin\frac{\theta}{2} = \cos\frac{\|\bar{\omega}\|\Delta t}{2} + \frac{\bar{\omega}}{\|\bar{\omega}\|}\sin\frac{\|\omega\|\Delta t}{2} \tag{2.46}$$

$$\|\bar{\omega}\| = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \tag{2.47}$$

Taking $\boldsymbol{q}(t + \Delta t) = \Delta\boldsymbol{q}\boldsymbol{q}(t)$*:*

$$\begin{aligned}\boldsymbol{q}(t+\Delta t)-\boldsymbol{q}(t) &= \left(cos\frac{\|\bar{\omega}\|\Delta t}{2}+\frac{\bar{\omega}}{\|\bar{\omega}\|}sin\frac{\|\bar{\omega}\|\Delta t}{2}\right)q-q \\ &= \left(-2sin^2\frac{\|\bar{\omega}\|\Delta t}{4}+\frac{\bar{\omega}}{\|\bar{\omega}\|}sin\frac{\|\bar{\omega}\|\Delta t}{2}\right)q\end{aligned} \tag{2.48}$$

Taking the definition of temporal derivative:

$$\begin{aligned}\dot{\boldsymbol{q}} &= \lim_{\Delta t\to 0}\frac{\boldsymbol{q}(t+\Delta t)-\boldsymbol{q}(t)}{\Delta t} \\ &= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left(-2sin^2\frac{\|\bar{\omega}\|\Delta t}{4}+\frac{\bar{\omega}}{\|\bar{\omega}\|}sin\frac{\|\bar{\omega}\|\Delta t}{2}\right)\boldsymbol{q} \\ &= \frac{\bar{\omega}}{\|\bar{\omega}\|}\lim_{\Delta t\to 0}\frac{1}{\Delta t}\left(sin\frac{\|\bar{\omega}\|\Delta t}{2}\right)\boldsymbol{q} \\ &= \frac{\bar{\omega}}{\|\bar{\omega}\|}\frac{d}{dt}\left(sin\frac{\|\bar{\omega}\|}{2}t\right)\Big|_{t=0}\boldsymbol{q} \\ &= \frac{1}{2}\begin{bmatrix}-\omega_x q_1-\omega_y q_2-\omega_z q_3\\ \omega_x q_0+\omega_z q_2-\omega_y q_3\\ \omega_y q_0-\omega_z q_1+\omega_x q_3\\ \omega_z q_0+\omega_y q_1-\omega_x q_2\end{bmatrix} \\ &= \frac{1}{2}\boldsymbol{\omega q}\end{aligned} \tag{2.49}$$

The product of the angular velocity $\bar{\omega}\in\mathbb{R}^3$ and quaternion $\boldsymbol{q}\in\mathbb{H}$ can be done expanding the angular velocity $\bar{\omega}$ in Hamilton space as a quaternion, $\boldsymbol{\omega}=\left[0,\omega_x,\omega_y,\omega_z\right]^T\in\mathbb{H}$.

Operator $\Omega(\bar{\omega})$ is defined as an extension of $\Theta(\bar{\omega})$:

$$\Omega(\bar{\omega})=\begin{bmatrix}0 & -\bar{\omega}^T\\ \bar{\omega} & \lfloor\bar{\omega}\rfloor_\times\end{bmatrix}=\begin{bmatrix}0 & -\omega_x & -\omega_y & -\omega_z\\ \omega_x & 0 & \omega_z & -\omega_y\\ \omega_y & -\omega_z & 0 & \omega_x\\ \omega_z & \omega_y & -\omega_x & 0\end{bmatrix} \tag{2.50}$$

where $\lfloor\bar{\omega}\rfloor_\times=\Theta(\bar{\omega})$ is an skew-symmetric matrix. Finally, the notation is:

$$\dot{\boldsymbol{q}}=\frac{1}{2}\Omega(\bar{\omega})\boldsymbol{q} \tag{2.51}$$

Note that the $\Omega(\bar{\omega})$ operator can be interpreted as the offset between the body and sensor reference frame [11], [12]. On the other hand, in sensor fusion systems typically $\Omega(\bar{\omega})$ terms are expressed with the signs changed so that (2.51) can be expressed as:

$$\dot{\boldsymbol{q}}=-\frac{1}{2}\Omega(\bar{\omega})\boldsymbol{q} \tag{2.52}$$

Finally, another generalized matrix form to express the time evolution can be deduced from expression (2.9) as follows:

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} 0 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2.53}$$

2.4. Fusion of the INS and GNSS

According to Oxford Dictionary [2], fusion "is the process or result of joining two or more things together to form one". In data sensor context, is the process of integrating information from several sensors into a single joint output. Regarding navigation systems, inertial navigation systems (INS) and global navigation satellite systems (GNSS) are two of the main techniques used in real environments for UAS navigation. Each of these systems aim to estimate different states of a vehicle such as position, velocity, attitude among others. INS are able to obtain states relative to attitude and positions in local coordinates, while GNSS systems such as the Global Position System (GPS), Glonass, or Galileo [13], allow to obtain the global position of a receiver by triangulating the signal from different satellites of the same GNSS family or even combining several systems [14]. Although INS can estimate local position and velocities, they present bias in their estimation that depending on the quality of the system propagates and diverges to a lesser or greater extent over time [15].

The integration of sensor information or fusion can be done with different strategies. Works such as D.L. Hall, [16], J. Llinas [17] or Paul D. Groves [1], show several classifications and strategies according to the form of information integration. In navigation systems, cascade integration and centralized integration are two of the most widespread strategies. As shown by Paul D. Groves [1] in both cases an integration algorithm is used. The main difference is that in the cascade architecture, each sensor is accompanied by its own estimation process and then all the results are integrated over the fusion algorithm. In the case of centralized fusion, the raw information from the sensors is fed into the fusion algorithm to generate a complete state vector.

Typical INS/GNSS fusion solutions [18]–[22] are based on a loosely coupled architecture (GNSS is an independent information source that provides a single position measurement), which uses GNSS position and velocity measurements to aid the INS. In this way, the IMU sensors are used to extrapolate position, velocity, and attitude at high frequency (50 Hz), while updates from GNSS measurements at lower frequency (1 Hz) allows the update of kinematic estimates and inertial sensor biases. Other INS/GNSS centralized integration is a tightly coupled integration, however the main problem of the tightly coupled architecture means

that there is no stand-alone GNSS solution, which is the reason why this solution is often discarded from real systems. In addition, other alternatives appear when independent modules with subsets of sensors are available to estimate attitude and kinematics with parallel filters [23], a robust solution but only available when the number of sensors is enough to group then in independent modules.

Regarding fusion algorithms, Kalman filter (KF) is the most widely used because of its optimal solution in terms of asymptotic convergence of the covariance to zero for the ideal case of linear systems with Gaussian noise. However, nonlinear transformations required by local-global transformations require methods to handle the nonlinearities. For this case extended Kalman filters (EKF) are one of the principal solutions used. These algorithms are based on the linearization of the nonlinear processes to locally simplify the problem and to be able to apply the classical KF strategy.

Other proposals for multisensor navigation besides EKF are unscented Kalman (UKF) and Interacting Multiple Model (IMM) filters. Besides, the problem of designing complex sensor fusion systems has been addressed from the point of view of machine learning. An approach to contextual aspects of GNSS/INS sources is presented in [24] present the use of dynamic Neural Networks to build models of INS errors before combination with GPS data to facilitate adaptation with time-varying errors. Recently, works such as the one by J.P. Llerena et al. [25] focused on the filtering and estimation of highly nonlinear systems using neural networks with Long-Short Term-Memory (LSTM) cells have shown very encouraging results for the tracking solution of complex systems, opening the possibility to using as centralized information fusion systems in navigation problems.

In any case, independently of the selected sensor fusion algorithm, the estimated state vector resulting in the output for the GNSS/INS filter usually contains the attitude 4-vector q, 3D positions, 3D velocity and 3D biases corrections for acceleration and angular rate in body frame, respectively $\overline{b}_a$ and $\overline{b}_\omega$.

$$\overline{x} = [\underbrace{\lambda, \varphi, h}_{Global\ Pos.}, \underbrace{\dot{E}, \dot{N}, \dot{U}}_{Ground\ velo.}, \underbrace{q_0, q_1, q_2, q_3}_{Attitude}, \underbrace{b_{a_x}, b_{a_y}, b_{a_z}}_{Accel.\ bias}, \underbrace{b_{\omega_x}, b_{\omega_y}, b_{\omega_z}}_{Gyros.\ bias}]^T \tag{2.54}$$

The attitude is usually computed with respect to the reference origin point taken at the mission arming point when engines are started. The position and velocity are usually expressed in the inertial ENU frame, and the sensor biases expressed in the body frame. The state vector can be extended, $\overline{x}^E$, to include bias in barometric height, (2.55), if a barometer is available, to integrate also this source of measurements.

$$\overline{x}^E = [\underbrace{\overline{x}^T}_{State\ vector}, \underbrace{b_h}_{Barometer\ bias}]^T \tag{2.55}$$

In this section the complete INS/GNSS fusion process to obtain the complete state vector (2.54) is described. First, the EKF estimation algorithm is presented as the basis of the state estimation process. Then, the state vectors provided by INS and GNSS are described to finally describe the specific process of centralized fusion based on the loosely coupled architecture.

2.4.1. State estimation

In estimation theory in stochastic processes, the Kalman filter is said to be the optimal solution since it minimizes the covariance of the system [26]. If we consider a stochastic nonlinear dynamic system (2.56), the first approximation derived from the KF, is the Extended Kalman Filter (EKF).

$$\begin{aligned} \dot{x} &= f(x, u, w) \\ z &= h(x, v) \end{aligned} \tag{2.56}$$

As in the linear Kalman filter [26]–[28], w shows the noise process and v the measurement noise. Note that the system and measurement model can be nonlinear. The EKF idea is built around the linearization system over the estimated states $\hat{x}$. This means $f(.)$ and $h(.)$ must be derived with respect to the states x, the model noise w, measurement noises v and the input signal u. To simplify the explanation an autonomous system is considered:

$$\begin{aligned} A &= \nabla f(x, 0,0)|_{(\hat{x},u,0)} \\ W &= \nabla f(0,0,w)|_{(\hat{x},u,0)} \\ H &= \nabla h(\boldsymbol{x}_k, 0)|_{(\hat{x},u,0)} \\ V &= \nabla h(0, v)|_{(\hat{x},u,0)} \end{aligned} \tag{2.57}$$

The first parenthesis in (2.57) denote the terms with respect the functions are derived from the system and measurement, while the second parenthesis, $(\hat{x}, u, 0)$, means the values to be substituted in the jacobian matrix.

When the continuous system has been linearized, the next step is to discretize and apply the same process as in the linear KF.

Kalman filters and EKF is divided into two steps, prediction and update. To identify these steps and the temporary state, Kalman notation uses a sub-index in the form $x_{A|B}$. The first, A, refers to the temporal state (current=k, previous=$k-1$) and the second, B, refers to the filter step (prediction=$k-1$, update=k).

The Kalman filter steps formulation is formulated as follows when the system doesn't have noise in estimation process and is autonomous $\Gamma = 0$ or when control signal $u_k = 0$.

Prediction step:

$$\begin{aligned} \hat{x}_{k|k-1} &= F\hat{x}_{k-1|k-1} \\ P_{k|k-1} &= FP_{k-1|k-1}F^T + Q_k \end{aligned} \tag{2.58}$$

Update step:

$$\begin{aligned} K_k &= P_{k|k-1}H^T(HP_{k|k-1}H^T + R_k)^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1}) \\ P_{k|k} &= (I - K_kH)P_{k|k-1} \end{aligned} \tag{2.59}$$

In KF, the forecast is made on the current state k, so it is usually called prediction in place of forecast. Firstly, a state space models (SSM) predict the current time state $\hat{x}_{k|k-1}$ and then, the prediction is improved $\hat{x}_{k|k}$ with the current measure z_k.

All Kalman notation is summarized under Table 2.1.

Table 2.1. Main Kalman notation definition.

Symbol	Definition
x_k	Current state vector
z_k	Current measure vector
x_{k-1}	Previous state vector
$\hat{x}_{k\|k-1}$	Current state vector in prediction step
$\hat{x}_{k\|k}$	Current state vector in update step
u	Input system signal/control signal
$f(x,u,w)$	Dynamic System
$h(x,v)$	Stochastic measure function
A	Linear system matrix (continues system)
B	Input matrix (continues system)
H	Observation matrix model
F	State transition matrix (discreet system)
Γ	Input matrix (discrete system)
Q	The covariance of the process noise
R	The covariance of the observation measurements
P	Covariance matrix (measure of the estimate accuracy)
K	Optimal Kalman gain
$W_k \sim \mathcal{N}(0, Q_k)$	Process noise
$V_k \sim \mathcal{N}(0, R_k)$	Observation noise

2.4.2. INS State vector

An Inertial Navigation System (INS) is composed of a set of sensors, processors and mathematical methods that process the information coming from the sensors in order to estimate physical states such as position, orientation and velocity without need of an external

reference. Thus, the INS aim to estimate a vector of states $\bar{x}$, composed of the attitude, expressed in quaternions $\boldsymbol{q}^b = [q_0, q_1, q_2, q_3]^T$, rotation rate expressed in the body reference frame $\{b\}$ $\bar{\omega}^b = \left[\omega_x, \omega_y, \omega_z\right]^T$ the gyroscope bias $\bar{b}_\omega = \left[b_{\omega_x}, b_{\omega_y}, b_{\omega_z}\right]^T$ and accelerometer biases $\bar{b}_a = \left[b_{ax}, b_{ay}, b_{az}\right]^T$.

$$\bar{x}_{IMU} = \left[\boldsymbol{q}^{b^T}, \bar{\omega}^{b^T}, \bar{b}_\omega^T, \bar{b}_a^T\right]^T \tag{2.60}$$

To estimate the above states, the standard INS uses measurements from a 6 or 9-DOF inertial measurement unit (IMU) consisting of 3-axis gyro (angular velocity), 3-axis accelerometer (accelerations) and several times in addition use 3-axis magnetometer (NED orientations) [11].

Gyroscope measurements:

The gyroscope provides information of the angular velocity. Considering that the sensor is placed at the center of the vehicle reference frame $\{b\}$, this sensor provides measurements $\bar{z}_\omega^b$ of the angular velocity states $\bar{\omega}^b$, which can be modeled according to (2.61).

$$\bar{z}_\omega^b = \bar{\omega}^b + \bar{b}_\omega + \bar{n}_\omega \tag{2.61}$$

where $\bar{b}_\omega = \left[b_{\omega_x}, b_{\omega_y}, b_{\omega_z}\right]^T$ is an additive error also called gyro bias vector and $\bar{n}_\omega = \left[\sigma_{\omega_x}^2, \sigma_{\omega_y}^2, \sigma_{\omega_z}^2\right]^T$ is a Gaussian white noise with power spectral density (PSD) σ_{gi}^2 associated with each of its $i - axis$.

- ***Accelerometer measurement***

Acceleration measurements $\bar{a}^b$ in the body reference frame $\{b\}$, can be modeled as $\bar{z}_a^b$:

$$\bar{z}_a^b = \bar{a}^b - \bar{g}^b + \bar{b}_a + \bar{n}_a \tag{2.62}$$

where $\bar{g}^b = \left[g_x^b, g_y^b, g_z^b\right]^T = R_p^b \cdot \bar{g}^p + b_{ag}$ is the gravity vector expressed in the reference frame $\{b\}$, R_p^b is the rotation matrix between the sensor platform reference frame $\{p\}$ and the sensor-vehicle center $\{b\}$ and b_{ag} is the bias in position between both reference systems. On the other hand $\bar{b}_a = \left[b_{ax}, b_{ay}, b_{az}\right]^T$ is the systematic error of the sensor and $\bar{n}_a = \left[\sigma_{ax}^2, \sigma_{ay}^2, \sigma_{az}^2\right]^T$ corresponds to a Gaussian white noise with PSD σ_{ai}^2 associated to each of its $i - axes$. In this case it considers the bias in the accelerometer triad to be insignificant.

- ***Magnetometer measurement***

The Earth magnetic field in the body reference frame is defined as $\overline{m}^b$ and its measurements z_m^b can be modeled as the sum of the field value m, a bias b_m and Gaussian white error n_m with PSD σ_m^2. Although the bias b_m could be very large its variation is very slow so calibration strategies such as the one shown in the work of J.F.Vasconcelos [29] can be used.

$$\overline{z}_m^b = \overline{m}^b + \overline{b}_m + \overline{n}_m \tag{2.63}$$

To obtain the attitude over time it is necessary to integrate the differential equation (60) along time.

It is outside the scope of this subsection the integration in the global reference frame of the Attitude. This requires information from the magnetometer. More information on this solution can be found in [1], [10], [29], [30].

To integrate (60), can be applied a polynomial linearization method of $\boldsymbol{q}(t+\Delta t)$ over time t, namely the Taylor series [31].

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t \Delta t + \frac{1}{2!}\ddot{\boldsymbol{q}}_t \Delta t^2 + \frac{1}{3!}\dddot{\boldsymbol{q}}_t \Delta t^3 + \ \ldots \tag{2.64}$$

Using the definition of $\dot{\boldsymbol{q}}$ (2.51), the new orientation $\boldsymbol{q}_{t+1}$ can be written [32] as:

$$\begin{aligned}\boldsymbol{q}_{t+1} = \left(\mathbb{I}_4 + \frac{1}{2}\Omega(\bar{\omega})\Delta t + \frac{1}{2!}\left(\frac{1}{2}\Omega(\bar{\omega})\Delta t\right)^2 + \cdots\right)\boldsymbol{q}_t + \frac{1}{2}\dot{\Omega}(\bar{\omega})\Delta t^2\boldsymbol{q}_t \\ + \left(\frac{1}{12}\dot{\Omega}(\bar{\omega})\Omega(\bar{\omega}) + \frac{1}{24}\Omega(\bar{\omega})\dot{\Omega}(\bar{\omega}) + \frac{1}{12}\ddot{\Omega}(\bar{\omega})\right)\Delta t^3\boldsymbol{q}_t + \cdots\end{aligned} \tag{2.65}$$

Considering that the angular velocity is constant in period $[t, t+1]$, the derivative of the angular velocity $\dot{\bar{\omega}} = 0^{3\times1}$, being able to discard the derivative of the operator Ω by reducing the series in:

$$\boldsymbol{q}_{t+1} = \left(\mathbb{I}_4 + \frac{1}{2}\Omega(\bar{\omega})\Delta t + \frac{1}{2!}\left(\frac{1}{2}\Omega(\bar{\omega})\Delta t\right)^2 + \cdots\right)\boldsymbol{q}_t \tag{2.66}$$

For high terms of the series, the error of the approximation vanishes quickly, in the same way when $\Delta t \to 0$, however the higher integration terms, of the series (2.64), improve our approximation, especially for high sampling times.

Looking at (2.66) it can be seen how the series is equivalent to the exponential map $e^{\frac{\Delta t}{2}\Omega(\omega)}$, where generally is truncated in the first term. Some of the reasons for applying this simple truncation are the simplicity of the architecture that facilitates its implementation in

embedded systems or that if the sensor signal is not filtered the estimation will not converge to a good result and will even get worse using higher order terms.

$$\boldsymbol{q}_{t+1} = \left(\mathbb{I}_4 + \frac{1}{2}\Omega(\bar{\omega})\Delta t\right)\boldsymbol{q}_t = e^{\frac{\Delta t}{2}\Omega(\bar{\omega})}\boldsymbol{q}_t \tag{2.67}$$

Applying exponential map (30):

$$\boldsymbol{q}_{t+1} = \underbrace{\left[cos\left(\frac{\|\bar{\omega}\|\Delta t}{2}\right)\mathbb{I}_4 + \frac{1}{\|\bar{\omega}\|}sin\left(\frac{\|\bar{\omega}\|\Delta t}{2}\right)\Omega(\bar{\omega})\right]}_{A(t)}\boldsymbol{q}_t \tag{2.68}$$

where the term inside the bracket that multiplies q_t is an orthogonal rotation that preserves the normalization of the quaternions, so it is not necessary to normalize q_{t+1} if q_t is normalized, although in many works it is recommended to do so to avoid rounding errors inherent to embedded systems. Generally, this matrix is known as the transition matrix of the attitude where its internal terms depend on the time instant "t" and can be expressed as $A(t)$.

Since there are different reference systems that are indicated by a subscript, to define the temporal state of the attitude, future $t+1$ and current t, hereinafter defined as $\boldsymbol{q}[k+1$ and $\boldsymbol{q}[k]$ respectively:

$$\boldsymbol{q}[k+1] = A[k]\boldsymbol{q}[k] \tag{2.69}$$

2.4.3. GNSS State vector

Global navigation systems provide absolute global positions and velocities relative to the Earth's surface, so the GNSS state vector is:

$$\bar{x}_{GNSS} = \left[\underbrace{\lambda, \varphi, h}_{Global\ Position}\ ,\ \underbrace{\dot{E}, \dot{N}, \dot{U}}_{Ground\ velocity}\right]^T \tag{2.70}$$

However, it should be noted that the observations, "$\bar{z}_{GNSS}$" provided by the global navigation system refer exclusively to the position $\bar{p}_{GNSS} = [\lambda, \varphi, h]^T$, and the velocity relative to the Earth's surface $\bar{v}_{en} = \left[\dot{E}, \dot{N}, \dot{U}\right]^T$. In information fusion systems, global coordinates are usually transformed to local ENU positions applying the set of geodetic-ECEF (2.3) and ECEF-ENU transformations. In this way the observations provided by the GNSS system, $\bar{z}_{GNSS} = \bar{p}_{en}$, refer to the ENU position, using as a reference point the origin of the mission. Furthermore, the measurements $\bar{z}_{GNSS}$, are modeled as the state value plus a Gaussian white noise $\bar{n}_{GNSS}$ compose by horizontal and vertical PSD σ_h^2, σ_v^2 respectively. So, the measurement model can be expressed as:

$$\bar{z}_{GNSS} = \bar{p}_{en} + \bar{n}_{GNSS} = \begin{bmatrix} x_{en} \\ y_{en} \\ z_{en} \end{bmatrix} + \begin{bmatrix} \sigma_h^2 \\ \sigma_h^2 \\ \sigma_v^2 \end{bmatrix} \tag{2.71}$$

GNSS are not very accurate in height measurement, so usually GNSS receivers have integrated or are combined with a barometer measurement, z_{bar}, to improve the deficiencies in GNSS height measurement. The barometric sensor can be modeled as:

$$z_{bar} = z + b_{bar} + n_{bar} \tag{2.72}$$

where z_{bar} is the barometric height, b_{bar} is the height bias and n_{bar} means a Gaussian noise white noise with PSD σ_{bar}^2. To define the current time instant "k" or future "$k+1$" of $j = \{ \bar{p}_{ne}, \bar{v}_{ne}, z_{GNSS}, z_{bar}\}$, hereafter it is expressed in square brackets as $j[k]$, and $j[k+1]$ respectively.

2.4.4. Fusion of the INS and GNSS

Although there are different INS/GNSS fusion strategies, this sub-section focuses on the centralized loosely couple integration using an Extended Kalman filter (EKF) as an approach to the EKF2 fusion system of PX4 flight controller.

A representative dynamic model is selected that integrates local sensed inputs (acceleration and gyros), absolute positions (GNSS data) and sensor dynamics biases, with uncertainty models. The position and velocity coordinate frame selected for this solution is the ENU frame with respect to the tangent plane with origin defined by the arming point at the start of the mission.

The EKF fusion algorithm processes all available measurements in a centralized module: GPS, barometric height sensors, and the input of IMU readings with local body-frame sensed acceleration and angular rates. As mentioned above, the GNSS and barometer inputs ($\bar{z}_{GNSS}, z_{bar}$) are considered, within Kalman inference mechanism, as "observations", while IMU inputs ($\bar{z}_a, \bar{z}_\omega$) (2.61) and (2.62) are considered as "control inputs", $\bar{u}[k]$. The input control contains the ideal magnitudes of 3D accelerations and angular rates expressed in the body-fixed local frame, respectively $\bar{a}_b[k]$ and $\bar{\omega}_b[k]$:

$$\bar{u}[k] = [\bar{a}_b[k]^T, \bar{\omega}_b[k]^T]^T \tag{2.73}$$

This input control is related to available IMU measurements at k-time ($\bar{z}_a[k], \bar{z}_\omega[k]$), which must be corrected with the respective estimated biases in the state vector.

$$\begin{aligned} &\bar{a}_b[k] = \bar{z}_a[k] - \bar{b}_a[k] \\ &\bar{\omega}_b[k] \equiv [\omega_x[k] \quad \omega_y[k] \quad \omega_z[k]]^T = \bar{z}_\omega[k] - C_b^{en}[k]\bar{\omega}_{en} - \bar{b}_\omega[k] \end{aligned} \tag{2.74}$$

Besides, in the case of body angular rate, also the Coriolis effect, $\bar{\omega}_{en}$, is subtracted to generate the corrected input control, projected through the Body-to-ENU ($\{b\} \rightarrow \{en\}$) frames conversion matrix, $C_b^{en}[k]$. This projection matrix is direclty obtained from the vehicle attitude expressed in quaternion vector as show in (2.29). The EKF requires a system model described by the state vector and a dynamic stochastic model to describe its evolution with time:

$$\dot{\bar{x}}(t) = \frac{\bar{x}(t)}{dt} = y(t, \bar{x}, \bar{u}, \bar{V}(t)) \tag{2.75}$$

being $\bar{u}(t)$ the control input (deterministic) and $\bar{V}(t)$ is the plant noise process (unobservable noise). The first term $\bar{u}(t)$ is related to the observations provided by inertial sensors as indicated above, which include a certain noise error, while $\bar{V}(t)$ is an additional noise model to take into account deviations from the predictions. The prediction equations resulting from the model are obtained after integration of differential equation, a well-known model [37] is obtained for this problem with the following (non-linear) equations:

$$\begin{gathered}\bar{x}_{en}[k+1] = f(\bar{x}_{en}[k], \bar{u}(t)) = F(\bar{x}_{en}[k], \bar{u}(t))\bar{x}_{en}[k] + \bar{u}(\bar{x}_{en}[k]) \\ \begin{bmatrix} \bar{p}_n[k+1] \\ \bar{v}_n[k+1] \\ \boldsymbol{q}_n[k+1] \\ \bar{b}_a[k+1] \\ \bar{b}_\omega[k+1] \end{bmatrix}_{en} = \begin{bmatrix} I_3 & \Delta t I_3 & 0_{3x4} & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & I_3 & 0_{3x4} & 0_{3x3} & 0_{3x3} \\ 0_{4x3} & 0_{4x3} & A[k] & 0_{4x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x4} & I_3 & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x4} & 0_{3x3} & I_3 \end{bmatrix} \begin{bmatrix} \bar{p}_n[k] \\ \bar{v}_n[k] \\ \boldsymbol{q}_n[k] \\ \bar{b}_a[k] \\ \bar{b}_\omega[k] \end{bmatrix} + \Delta t \begin{bmatrix} 0_{3x1} \\ U[k] \\ 0_{4x1} \\ 0_{3x1} \\ 0_{3x1} \end{bmatrix}\end{gathered} \tag{2.76}$$

where $A[k]$ is the attitude transition matrix as show in (2.68) which depends on the components of corrected angular velocity in body frame, $\bar{\omega}_b[k] = [\omega_x[k] \quad \omega_y[k] \quad \omega_z[k]]^T$.

Finally, $U[k]$ is the correction in the velocity computed from the control input, corresponding to acceleration vector, expressed in inertial frame and projected to ENU frame, affected by the gravitational effect, $\bar{g}$.

$$U[k] = C_{en}^b[k]\ \bar{a}_b[k] + \bar{g} \tag{2.77}$$

With this dynamic model, the EKF approximates the predictions and their covariance matrix with a first-order approximation for the non-linear functions of prediction model, $f(.)$, and projection to the measurement space, $h(.)$:

$$\begin{aligned} \bar{x}_{en}[k|k-1] &= f(\bar{x}_{en}[k], \bar{\mathrm{u}}[k], \bar{\mathrm{V}}[k]) \\ z_{en}[k] &= h(\bar{x}_{en}[k], \bar{W}[k]) \end{aligned} \tag{2.78}$$

being $\bar{W}[k]$ the observation noise process, and $\bar{\mathrm{V}}[k]$ the system process noise to characterize uncertainty in the predictions. The prediction equations of Extended Kalman Filter are used to propagate the state vector and covariance matrix:

$$\hat{x}[k|k-1] = f(\hat{x}[k-1|k-1], \hat{u}[k-1])$$
$$P[k|k-1] = F[k]P[k-1|k-1]F^T[k] + V[k]Q_u[k-1]V^T[k-1] + Q_p[k-1] \tag{2.79}$$

The matrices F, V, H are computed with the Jacobean operators applied to the model functions f and h:

$$F_{[i,j]} = \frac{\delta f_{[i]}}{\delta \bar{x}_{[j]}}(\bar{x}(t), \boldsymbol{q}(t), \bar{u}(t), t) \tag{2.80}$$

$$V_{[i,j]} = \frac{\delta f_{[i]}}{\delta \bar{u}_{[j]}}(\bar{x}(t), \boldsymbol{q}(t), \bar{u}(t), t) \tag{2.81}$$

$$H_{[i,j]} = \frac{\delta h_{[i]}}{\delta \bar{x}_{[j]}}(\bar{x}(t), w(t)) \tag{2.82}$$

The covariance matrix for process noise, Q, is separated in two terms: Q_p, corresponding to uncertainty in predictions and Q_u, projecting the errors of inertial sensors to the state vector.

$$Q[k] = V[k]Q_u[k-1]V^T[k-1] + Q_p[k-1]$$
$$Q[k] = V[k]\begin{bmatrix} 0_{10x10} & 0_{10x3} & 0_{10x3} \\ 0_{3x10} & q_{am}^2 I_3 & 0 \\ 0_{3x10} & 0 & q_{\omega m}^2 I_3 \end{bmatrix} V^T[k] + \begin{bmatrix} 0_{10x10} & 0_{10x3} & 0_{10x3} \\ 0_{3x10} & q_{ap}^2 I_3 & 0 \\ 0_{3x10} & 0 & q_{\omega p}^2 I_3 \end{bmatrix} \tag{2.83}$$

being $q_{am}^2, q_{\omega m}^2, q_{ap}^2, q_{\omega p}^2$ the model parameters used to tune the system response by adaptation of the plant-noise process $Q[k]$. The first two, $q_{am}^2, q_{\omega m}^2$, are used to model the uncertainty of IMU sensors, projected through V matrix to state vector, and the last two, $q_{ap}^2, q_{\omega p}^2$, are direct models of prediction uncertainty for the assumed model of smooth variation of inertial sensor biases.

These parameters can be tuned as shown in [38] and fine-tuned to improve the navigation system.

Finally, given the prediction model detailed above, the "update" phase of EKF is given by the classical KF (2.59) detailed for this particular case as:

$$K[k] = P[k|k-1]H[k]^T(H[k]P[k|k-1]H[k]^T + R[k])^{-1}$$
$$\hat{x}[k|k] = \hat{x}[k|k-1] + K[k](z[k] - h(x[k|k-1])) \tag{2.84}$$
$$P[k|k] = P[k|k-1](I - K[k]H[k])$$

where the $R[k]$ matrix in EKF update equations consider the noise in measurement observations:

$$R_{GNSS} = \begin{bmatrix} \sigma_h^2 & 0 & 0 \\ 0 & \sigma_h^2 & 0 \\ 0 & 0 & \sigma_v^2 \end{bmatrix};\ R_{BAR} = \sigma_{bar}^2 \tag{2.85}$$

Remembering that σ_h^2, σ_v^2 are the variance in horizontal and vertical errors of GNSS positions, respectively, and σ_{bar}^2 the variance in barometric height.

Therefore, the EKF filter used for sensor fusion depends on two sets of parameters which are sensor noise and plant noise. The terms in (2.83) correspond to the plant noise, used to stabilize the filter, and avoid becoming too confident in its own predictions with respect to measurements. Both the estimation of cinematic parameters and sensor biases depend on choosing appropriate parameters characterizing noise in sensor data and uncertainty in prediction (process noise). Especially, process noise parameters affect to the predicted error covariance and have critical impact in the weights given to the sensor observations with respect to the predicted estimates. A higher value for these parameters implies higher values of predicted covariance and so higher gain to observations (since the confidence on prediction decreases). Conversely, lower values imply lower gain to observations (higher confidence on predictions). For example, if GNSS position noise parameters are set to very small values compared to INS prediction errors, it will produce frequent changes of position and attitude during vehicle trans state. In the same way, low values for GNSS velocity noise will cause the filter roll and pitch angles to be noisy, probably affecting to the vehicle motion up and down.

2.5. References

[1] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Navigation Systems*. Boston, MA, USA; London, UK: Artech House, 2013.

[2] "Oxford Learner's Dictionaries | Find definitions, translations, and grammar explanations at Oxford Learner's Dictionaries." https://www.oxfordlearnersdictionaries.com/ (accessed Jul. 26, 2022).

[3] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of unmanned aerial vehicles*. Springer Dordrecht Heidelberg New York London : Springer, 2015. doi: 10.1007/978-90-481-9707-1.

[4] D. H. Sattinger and O. L. Weaver, *Lie groups and algebras with applications to physics, geometry, and mechanics*. Berlin Heidelberg New York Tokio: Springer-Verlag, 1986. Accessed: Jul. 30, 2022. [Online]. Available: https://books.google.es/books?hl=es&lr=&id=RVzhBwAAQBAJ&oi=fnd&pg=PA3&dq=lie+algebra+to+mechanic&ots=6xq_Fkhsw8&sig=ldwkuVbvy-aGEAh0hCdk3VZ40Pg

[5] F. Lachello, *Lie algebras and applications*. Heidelberg New York Dordrecht London: Springer Heidelberg , 2006. Accessed: Jul. 30, 2022. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-662-44494-8.pdf

[6] J. Gallardo-Alvarado, "Appendix 1: A Simple Method to Compute the Rotation Matrix," *Kinematic Anal. Parallel Manip. by Algebr. Screw Theory*, pp. 355–356, 2016, doi: 10.1007/978-3-319-31126-5_16.

[7] R. M. Rogers, *Applied mathematics in integrated navigation systems*, 3rd ed. Reston Virginia: American Institute of Aeronautics and Astronautics, 2007.

[8] J. Solà, "Quaternion kinematics for the error-state Kalman filter," 2017, doi: 10.48550/arxiv.1711.02508.

[9] T. Y. Lam, "HANDBOOK OF ALGEBRA, VOL. 3," in *Handbook of Algebra*, Elsevier S., vol. 3, M. Hazewinkel, Ed. Amsterdam ; New York: Elsevier, 2003, pp. 429–454. doi: 10.1016/S1570-7954(03)80068-2.

[10] R. G. Valenti, I. Dryanovski, and J. Xiao, "Keeping a good attitude: A quaternion-based orientation filter for IMUs and MARGs," *Sensors*, vol. 15, no. 8, pp. 19302–19330, 2015, doi: 10.3390/s150819302.

[11] R. Munguía and A. Grau, "A Practical Method for Implementing an Attitude and Heading Reference System Regular Paper," *Int. J. Adv. Robot. Syst.*, 2014, doi: 10.5772/58463.

[12] X. Kong, "INS algorithm using quaternion model for low cost IMU," *Rob. Auton. Syst.*, vol. 46, no. 4, pp. 221–246, 2004, doi: 10.1016/j.robot.2004.02.001.

[13] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle, "GNSS-Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more," *Choice Rev. Online*, vol. 45, no. 11, pp. 45–6185, 2007, doi: 10.5860/choice.45-6185.

[14] C. Cai and Y. Gao, "Modeling and assessment of combined GPS/GLONASS precise point positioning," *GPS Solut.*, vol. 17, no. 2, pp. 223–236, Apr. 2013.

[15] A. Hasan, K. Samsudin, A. Rahman bin Ramli, and S. Ismaeel, "A Review of Navigation Systems (Integration and Algorithms)," *Aust. J. Basic Appl. Sci.*, vol. 3, no. 2, pp. 943–959, 2009.

[16] D. L. Hall and J. Llinas, "An introduction to multi-sensor data fusion," in *Proceedings of the IEEE (Volume:85, Issue: 1)*, 2016, pp. 6–23. Accessed: Jul. 30, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/554205/

[17] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White, "Revisiting the JDL data fusion model II," in *Proceedings of the Seventh International Conference on Information Fusion, FUSION 2004*, 2004, vol. 2, pp. 1218–1230.

[18] R. Sun, Q. Cheng, G. Wang, and W. Y. Ochieng, "A novel online data-driven algorithm for detecting UAV navigation sensor faults," *Sensors*, vol. 17, no. 10, 2017.

[19] Y. Yao and X. Xu, "A RLS-SVM Aided Fusion Methodology for INS during GPS Outages," *Sensors 2017, Vol. 17, Page 432*, vol. 17, no. 3, p. 432, Feb. 2017, doi: 10.3390/S17030432.

[20] E. Martí, J. García, J. M.-I. C. on H. Artificial, and undefined 2010, "A simulation framework for UAV sensor fusion," *Springer*, vol. 6077, no. PART 2, pp. 460–467, 2010, doi: 10.1007/978-3-642-13803-4_57.

[21] J. Besada, A. Soto, G. De Miguel, J. García, and E. Voet, "ATC trajectory reconstruction for automated evaluation of sensor and tracker performance," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 28, no. 2, pp. 4–17, 2013, doi: 10.1109/MAES.2013.6477864.

[22] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Inf. Fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[23] T. Layh and D. Gebre-Egziabher, "Design for graceful degradation and recovery from GNSS interruptions," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 32, no. 9, pp. 4–17, 2017, doi: 10.1109/MAES.2017.160194.

[24] E. D. Martí, D. Martín, J. García, A. de la Escalera, J. M. Molina, and J. M. Armingol, "Context-aided sensor

fusion for enhanced urban navigation," *Sensors*, vol. 12, no. 12, pp. 16802–16837, 2012, doi: 10.3390/s121216802.

[25] J. P. L. Llerena, J. G. Herrero, and J. M. M. Molina, "Forecasting nonlinear systems with lstm: Analysis and comparison with ekf," *Sensors*, vol. 21, no. 5, pp. 1–29, 2021, doi: 10.3390/s21051805.

[26] K. Åström and B. Wittenmark, *Computer-controlled systems: theory and design*. Mineola, New York: Dover Publications, 2013.

[27] F. L. Lewis, *Optimal Control*, 3rd ed. Hoboken, New Jersey: John Wiley & Sons, 2012.

[28] P. Zarchan and H. Musoff, *Fundamentals of Kalman Filtering: A Practical Approach, Third Edition*. Reston Virginia: American Institute of Aeronautics and Astronautics, 2000.

[29] J. F. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira, and B. Cardeira, "Geometric approach to strapdown magnetometer calibration in sensor frame," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 47, no. 2, pp. 1293–1306, 2011, doi: 10.1109/TAES.2011.5751259.

[30] R. P. G. Collinson, *Introduction to Avionics Systems*. Dordrecht: Springer Netherlands, 2011.

[31] F. Tronarp and S. Särkkä, "Continuous-Discrete Filtering and Smoothing on Submanifolds of Euclidean Space," Apr. 2020, Accessed: Jul. 29, 2022. [Online]. Available: http://arxiv.org/abs/2004.09335

[32] J. R.Wertz, *Spacecraft attitude determination and control*, vol. 73. Dordrecht / Boston / London: Springer Science & Business Media, 2012.

Chapter 3: Machine Vision Systems of UAS

3.1. Introduction

For several years now, the low cost of vision sensors [1] and the large number of studies related to image processing have made vision systems essential systems for any unmanned aircraft. Some of the main vision-based applications found in the literature focus on navigation systems, obstacle detection and avoidance, visual servoing subsistems, as well as autonomous/precision landing, autonomous surveillance missions, infrastructure inspection or autonomous refueling, among others.

As navigation is concerned, there are two main groups, outdoor and indoor navigation. As for outdoor navigation, INS/GNSS fusion systems are widespread as described in the previous chapter. However, there are applications that improve the accuracy of the navigation system by adding to the loosely coupled architecture relative displacement velocity information through optical flow identification [2]. Other applications such as presented by G. Conte and P. Doherty [3] and complemented by the work of J. R. G Braga et al. [4], focus on avoiding GPS signal loss problems by geolocating the position of an aircraft by correlating georeferenced satellite images with images taken from the aircraft. With this proposal they replace in the INS/GNSS navigation system the information coming from GNSS with a fusion system between visual odometry and image correlation that is integrated in a 12-state loosely couple fusion architecture.

For indoor navigation visual odometry (VO) and simultaneous localization and mapping (SLAM) [5] are the main trends. With this technology small low-cost drones can be used for applications such as shown in S. Krul et al. [6] research, in which this navigation technology is used to enable a low-cost commercial drone to orient itself inside greenhouses and farms using a low-cost monocular vision system. In addition, this technology can be used with swarms of drones as if they are considered as agents as shown in the work of D. Zou et al. [7].

Obstacle detection and avoidance is another area of great interest for aircraft safety applications and the environment itself. Approaching the sense and avoid problem from a computer vision point of view presents serious challenges. Focusing the problem on sensing,

a single conventional camera cannot infer the depth of a scene directly. Classically to solve this problem, stereo systems with epipolar geometry have shown good results. Since several years ago with the rise of deep learning, some deep learning researches such as S. Saleh et al. [8]have been able to perceive depth in scenes with monocular images. In this case the authors use semi-supervised learning using a subset of the KITTI database containing LIDAR information and images in urban environments. With this information the authors manage to train a deep neural network that infers depth from a 2D image in real time ranges. Another classic strategy in scene sensing is based on motion perception. In the work of Tom van Dijk [9] the problem of sonorization and avoidance for UAVs is analyzed.

There are a multitude of applications with vision and drones, in the review by A. Al-Kaff et al. [10] another wide range of them can be found, including visual servoing, border surveillance, infrastructure inspection, agriculture, among others.

All these applications, in one way or another, require aircraft to perform a fundamental and critical maneuver, landing. One of the most common strategies in this area is to use vision and context information to identify a landing surface and/or improve the accuracy in terms of the relative positions between the aircraft and the landing surface. In recent works such as J.P. Llerena and his colleagues from the Applied Artificial Intelligence Group at Universidad Carlos III de Madrid [11] evaluate the relative position estimation error of a helipad and propose a combination of cylindrical space bias correction and a novel descent function. This work is discussed in detail in the following chapter.

Many of the mentioned applications are not only limited to machine vision in the visible electromagnetic spectrum, but digital image processing allows exploiting the potential of multispectral images. Some examples of success in this field are infrastructure inspection using thermographic vision [12] or crop analysis [13].

The deployment of the mentioned applications requires physical elements where the algorithms and strategies can be executed. There are two main approaches: onboard and offboard. Onboard systems are characterized for using companion computers that provide a higher degree of independence from ground stations, however, the performance of the equipment is reduced since the payload of the aircraft and its autonomy is affected. Offboard deployments allow use of large computing resources, including cloud computing, however, this requires the aircraft must be in communication with the ground station where the technology is deployed, increasing its external dependence.

3.2. Computer Vision

As a Cambridge dictionary [14] “vision” is “an idea or mental image of something”, on the other hand, “Image” is a “a picture in your mind or an idea of how someone or something is”,

and picture “the way that something or someone is thought of by other”. In this way, the principal goal of computer vision is to extract information from the real-world using computer-embedded algorithms.

Vision systems involve a multidisciplinary set of tasks such as raw information extraction, processing and analysis that can be placed under the scope of artificial intelligence fields.

Having a mathematical model that relates the perception of objects/things from a physical world to what is perceived by a sensor is fundamental for an intelligent system to be able to interpret the surrounding environment.

Considering the information acquisition problem decoupled and solved, computer vision applications are based on algorithms and strategies for the extraction of relevant information from the raw image data. Free software tools such as OpenCV [15] integrate a variety of algorithms and applications to solve computer vision problems. These free software tools are supported by a large community of developers and scientists that facilitate the use of computer vision strategies.

Testing vision applications on drones is no easy task, but hyper-realistic image simulators such as AirSim are a great help in testing applications without the additional risk or cost of an undetected design flaw.

The available images information from AirSim include, RGB images, segmentation images, infrared, and deep images [14].

The objective of this section is to provide the basic concepts of a camera model and its calibration in order to be used in complex vision applications.

3.2.1. Pinhole camera

Vision system modeling and calibration is a key issue in a multitude of applications. A main mathematical camera model is the pinhole camera model. This model is based on the concepts of paraxial geometrical optics and projective geometry. The objective of a pinhole camera model is relating the coordinates of the real world in 3D $\{w\}$ to the coordinates of the camera itself {c} and finally to the coordinates of the projective plane, Fig. 3.1, $\{ph\}$.

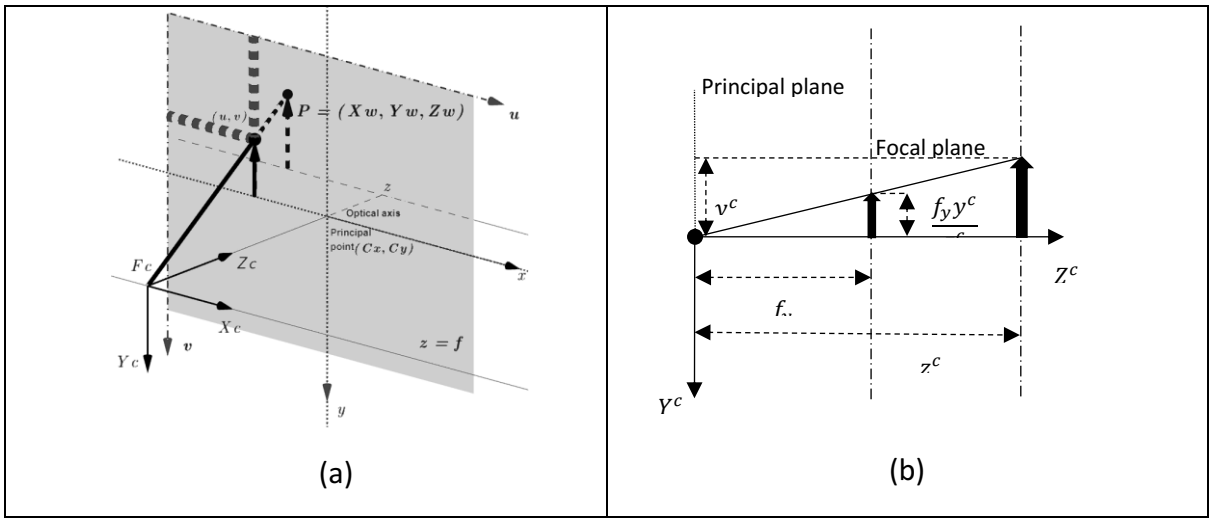


Fig. 3.1. Pinhole geometrical reference frames representation. a) general image, b) tringle of transformation.

Let the center of projection be the origin of a Euclidean coordinate system, and the plane $Z = f$, which is called the image plane or focal plane. The line starting from the center of the camera and perpendicular to the image plane is known as the optical axis or principal axis. The intersection of the principal axis with the image plane is called the principal point (c_x, c_y). The plane through the center of the camera and parallel to the image plane is known as the principal plane of the camera. $\mathcal{F}^c$ is the center of the camera and is where the origin of coordinates is located [15].

In the pinhole model, a point in space p^w expressed in the camera reference frame {c} can be expressed as $p^c = (x^c, y^c, z^c)^T$ and projected onto the image plane using the triangle, Fig. 3.1. Pinhole geometrical reference frames representation. a) general image, b) tringle of transformation.

(b), as shown in equation (3.1), where (f_x, f_y) are the x and y focal lengths of the optical system. Ignoring the depth coordinate z, the central projection mapping from the 3D world, p^c, to the 2D image coordinates, p^{ph}, is:

$$p^{ph} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{f_x x^c}{z^c} \\ \frac{f_y y^c}{z^c} \end{bmatrix} \tag{3.1}$$

Until now it has been supposed the origin of coordinates in the image plane is the same as the principal point, however, in Fig. 3.1 (a), it is observed that this may not be true so that equation (3.1) can be corrected as:

$$p = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{f_x x^c}{z^c} + c_x \\ \frac{f_y y^c}{z^c} + c_y \end{bmatrix} \tag{3.2}$$

The homogeneous coordinates of a point with Cartesian coordinates $(x, y, z)^T$ are defined as (kx, ky, kz, k), where $k \neq 0$ is an arbitrary constant. In the particular case of $k = 0$, the coordinates determine a vector, or in other words a direction. The conversion from homogeneous to Cartesian coordinates is performed by dividing the first three components of the homogeneous coordinates by the fourth.

Taking the assumption that the world and image points are represented in homogeneous coordinates, then the central projection can be expressed as a linear mapping between their homogeneous coordinates in terms of matrix multiplication by (3.3).

$$\begin{bmatrix} u \\ v \\ z^c \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} x^c \\ y^c \\ z^c \end{bmatrix} \tag{3.3}$$

Where the arbitrary constant k of Cartesian coordinate transformation is taken z^c which cancels the denominator of the u and v coordinate terms, equation (3.1) and (3.2). The A matrix is called as intrinsic matrix or calibration matrix [15].

If the coordinates of a point p^w are expressed in other reference frame $\{w\}$ than the camera reference frame {c}, it can be expressed in the camera reference frame, by a homogeneous transformation ${}^cT_w = [R|t]$ composed of a rotation R and a translation t.

$$p^c = {}^cT_w p^w;\ {}^cT_w = [R|t] = \begin{bmatrix} R^{3\times3} & t^{3\times1} \\ 0^{1\times3} & 1 \end{bmatrix} \tag{3.4}$$

The ${}^cT_w = [R|t] \in \mathbb{R}^{4\times4}$ matrix is referred to as the external matrix of the system. Finally, the pinhole camera model can be express as:

$$p^{ph} = A[R|t]p^w \tag{3.5}$$

3.2.2. Camera calibration

Camera calibration means the identification of the intrinsic A-matrix terms of the camera. This problem is usually approached from the estimation of the camera pose with a calibration pattern.

For pose estimation, the Perspective-n-Point (PnP) problem [16] is formulated where the objective is to minimize the reprojection error (3.7) of 3D points in the image plane $\{ph\}$.

Given a point $p^c \in \mathbb{R}^3$ belonging to the knowing pattern located in real-world 3D space and expressed in the camera reference frame $\{c\}$, it can be expressed in the image camera plane reference frame $\{ph\}$ as $\mathrm{p^{ph}} \in \mathbb{R}^2$. The relationship between the two reference frames is provided by the pinhole camera model in (3.5).

The pinhole model can be improved with radial, tangential, or prism distortion corrections, adding n set of k_i parameters to the model [17], [18]. The set of internal parameters of the camera model can be expressed by the vector $\delta = (f_x, f_y, c_x, c_y, k_1, \dots, k_n)$.

If the point is expressed in coordinates of the pattern reference frame p^w, there exists an extrinsic homogenous transformation ${}^cT_w$ to relate the reference frame of the pattern to the camera reference frame Equation (3.4) is used.

As discussed above, the transformation ${}^cT_w = [{}^cR_w|t_w^c]$ is a rototranslation composed of the pattern's orientation ${}^cR_t = R_x(\theta_1)R_y(\theta_2)R_z(\theta_3)$ to the camera and the pattern position vector to the camera $t_w^c \in \mathbb{R}^3$. Thus, the parameter vector to be identified to obtain the camera–pattern relationship is $\theta = (R, t) = (\theta_1, \theta_2, \dots, \theta_6) \in \mathbb{R}^6$.

Finally, the camera model remains as a function (3.6) that projects points $p^w \in \mathbb{R}^3$ to $p^{ph} \in \mathbb{R}^2$ points of the camera image plane.

$$p^{ph} = \Psi(\delta, \theta, p^w) \tag{3.6}$$

Calibration problem is the problem of minimizing the reprojection error Equation (3.7) of the observed calibration pattern features and detecting feature. One of the classic features to identify by computer vision are the corners. If the pattern is known, we know a priori the 3D position of these corners in the reference frame of the pattern.

$$\hat{E} = arg\,min \sum_{p_i^w \in \mathrm{C}} [\Psi(\delta, \theta, p_i^w) - O(p_i^w)]^2 \tag{3.7}$$

where $p_i^w \in \mathrm{C}$ and C is a corner set of the pattern. $O(p_i^w) \in \mathbb{R}^2$ is the corners obtained in the camera plane. Furthermore, since all points p_i^w belong to a pattern plane, the z-component of all corners in the pattern frame will always be 0. This quality allows solving (3.7) using specific methods such as the Infinitesimal Plane-Based Pose Estimation (IPPE) [19].

The estimation of internal camera parameters requires a learning phase modeled in (7) as an optimization problem. In addition, identifying the six parameters to define the transformation ${}^cT_w$ between the pattern calibration and the camera, involves a similar process. Although both processes can be clustered as shown in (3.7), the internal camera parameters δ will be constant for a particular vision system; however, the position of the pattern may change.

3.3. Image stabilization

In the work of J. Dong and H. Liu [20] the authors define video stabilization as "the process of improving video quality by eliminating the effect of fluctuating motions due to wobble". This paper focuses on video stabilization for a real-time system in the infrared spectrum mounted on an unmanned aerial vehicle.

The authors classify the systems into three different groups: sensor and lens based, mechanical tool based and computational tool based. The first ones focus on the way the camera receives the light. The latter aims to eliminate undesirable vibrations during recording. Finally, computational tools use mathematical algorithms to improve video quality. The latter do not require additional software or knowledge of the capture device. In turn, computational methods are divided into delayed and offline.

These classifications can also be found in works such as those of Adeel Yousaf et al. in [21] or M. Ahmed in [22], who also contemplates within the digital methods those that use electronic processing to control the stability of the image, defining them only as software algorithms. In J.Dong et al. in their work [23] they divide video stabilization into mechanical and software based. On the other hand, S. Ertürk et al. in their work [24] differentiates between mechanical-optical stabilizing systems that are those that use gyroscopic sensors or adjust the angle of refraction, mechanical-digital systems that are those in which the movement is corrected with digital processing and mechanical and purely digital actuators that are those systems where the estimation and correction of the movement is done digitally.

Due to the different classifications given by the cited authors, this section focuses on the two broadest types of stabilization: mechanical stabilization and digital or computational stabilization. In several works such as that of X. Cheng et al. [25] the two types of stabilization are employed together to improve the final system.

In the image stabilization process, a series of steps must be carried out in order to calculate the movements acting on the system and correct them. The steps for motion stabilization are based on estimating the movement that is occurring, correcting it and, in the case of digital stabilization, compensating the stabilized image. There are a multitude of techniques to identify motion, in the following lines the main techniques and theoretical foundations found for the measurement of motion in video stabilization systems with mechanical and computational techniques are discussed. In addition, this section presents some of the theoretical foundations that are the basis for understanding the steps necessary for stabilization from mechanical and computational perspectives.

3.3.1. Mechanical stabilization

Mechanical stabilization focused on the camera aims to reduce camera body movements caused by unintentional actions. This type of stabilization can be carried out by purely mechanical devices, or by more complex systems based on mechanical sensors and actuators. Some of the simplest systems used to stabilize a camera body are tripods and monopods. Both consist of external devices that are attached to the camera to prevent camera movement. In the case of UAVs, the goal of the vision systems is the same, however, these vehicles can make very fast movements. This versatility in movement poses a problem in terms of image capture.

In real life, the mechanical stabilization solution for this is the so-called gimbal which is a support system that allows the camera to maintain the desired angle so no matter what the motion is, the direction of shooting remains the same Fig. 3.2. Some other more advanced gimbals also include anti vibration features.

Fig. 3.2. Drone with gimbal stabilizer. Artificial image generated with Stable Diffusion AI [26].

In mechanical stabilization systems used in robotic applications such as drones, there are two phases: One for motion estimation and one for motion correction or smoothing.

As an example of mechanical stabilization in robotics using electromechanical actuators, WS. Rone et al. in [27] propose a system for mechanical stabilization of a robot by adding a robotic tail. To stabilize the robot body at each movement, motions and weight distributions are measured with digital sensors. Then, a response is computed and angles are sent to the electromechanical actuators of each tail segment.

An example of stabilization for a control system is shown by X. Cheng et al. [25] where they use a hybrid stabilization system for a vision robotic platform. It uses the acceleration information in the XYZ axes and transforms them into rotation angles, as well as giving information to the image stabilization unit.

The measurement of motion in mechanical systems is usually performed with sensors that determine certain kinematic states of the vision system such as positions, velocities or accelerations in different representations and coordinate spaces. These sensors, in general, are electromechanical or optoelectronic devices capable of transforming the measurements of the desired states into analog or digital electronic signals. An example of these devices are the inertial measurement units or IMU which integrate a set of sensors that allow the measurement of different states such as accelerations, velocities and attitudes.

To correct the motion in a mechanical system, it must be cancelled by performing the opposite motion. Knowing the direction, sense and modulus of the movement, it is possible to correct this movement by applying a movement with the same direction and modulus, but in the opposite direction. One way to compensate these movements is the use of servomotors. These actuators consist of an electric motor, an encoder that transforms the rotational position into electrical signals, a control system to position the motor by means of electrical pulses and a regulation system to modify the speed and torque of the motor. The positioning control is performed by means of a PWM (pulse-width modulation) signal, which consists of a control signal by means of a periodic pulse train. By modifying the width of the pulse sent to the servomotor, its final position is varied. By sending a position to the servomotor, it works as a closed-loop control system through a feedback based on the encoder information and giving great precision to the final positioning. In this way, by means of sensors that measure the movements and servomotors that compensate them, a system can be mechanically stabilized.

3.3.1.1. Example of mechanical stabilization in AirSim

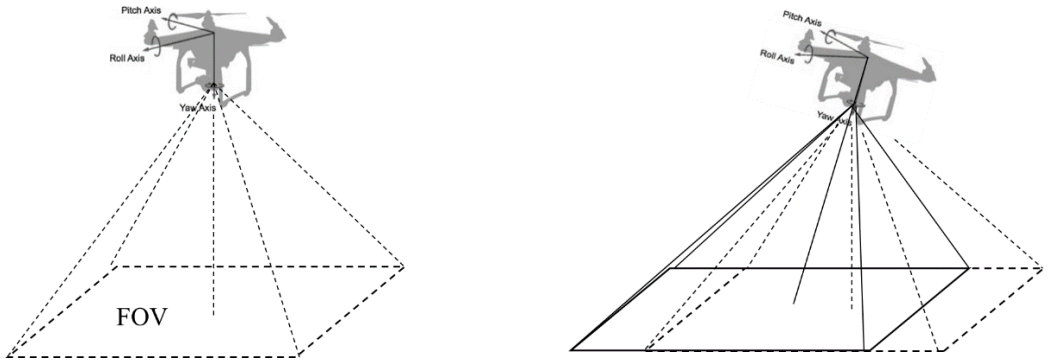
Assume a drone with a camera placed at the bottom to capture images in the zenith plane. If the vehicle is hovering at a certain altitude and stable, the attitude of its yaw axis is perpendicular to the ground plane or image plane Fig. 3.3 (a). However, if the vehicle moves horizontally while keeping the altitude, the field of view no longer corresponds to its perpendicular projection on the surface Fig. 3.3 (b), but a slightly deviated plane.

To correct this effect and/or simulate the behavior of a gimbal in AirSim, it can be defined in the settings of the settings.json file [28].

To keep stable zenithal plane, it is necessary to orient the gimbal to the ground, in other words to rotate -90º the x-direction axis, corresponding to the Euler pitch angle.

Fig. 3.3 shows the case of the movement of a drone in the AirSim environment when the gimbal is without stabilization and when we introduce stabilization. Fig. 3.4 (a) shows from an external observer the position of a drone when it moves horizontally, to do so it acquires a

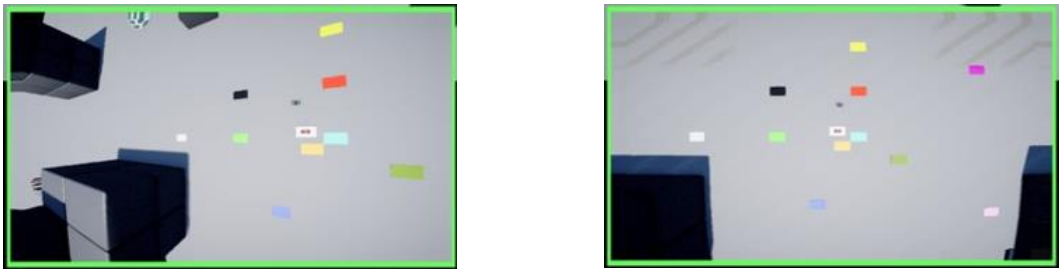
certain inclination in the pitch and roll axes. Fig. 3.4 (a) and (b) show the image captured using the gimbal without stabilization and with stabilization.



(a) (b)

Fig. 3.3. Deformation of the zenithal field of view by clearance of a rotary wing drone. A) Ideal zenithal plane. B) Effect on the field of view.

(a)



(b) (c)

Fig. 3.4. Gimbal simulation in AirSim. a) Image using a gimbal without stabilization, b) Image using a stabilized gimbal.

3.3.2. Computational stabilization

Computational image stabilization systems work on sequences of frames previously captured by the vision system, inferring on the data matrix associated with these images through the use of different algorithms.

Works such as A. Hamza et al. In [29] or WC. Hu, CH. Chen, et al. [30], address the problem of real-time stabilization for fast vehicles using a feature tracker. The method employed is divided into four steps: estimation of the frame motion, classification of the feature points belonging to the background or foreground, calculation of the global motion using the optical flow of the background points, and a filtering using a Kalman filter. Finally, the frame motion is compensated by applying an inverse and smoothed version of the global motion vector. C. Chen et al. in [31] approach the problem in the same way, but a DBSCAN (Density-Based Spatial Clustering) algorithm is used for background point separation.

On the other hand, S. Liu et al. in [32] propose a variation of the optical flow method called “SteadyFlow”. This method enforces spatial coherence by replacing feature trajectories with pixel profiles. Pixel profiles are motion vectors collected from the same pixel over time. In addition, this method can treat spatially variant motion making it suitable for processing scenes with depth. For its initialization, the optical flow technique is used, and discontinuous motions are discarded by spatiotemporal analysis. It ends by recreating the final image with the "as-similar-as-possible" technique. This method achieves high performances, but in offline applications because it is an iterative process.

In the work of SK. Kim et al. [33], authors use only background features points. The perspective projection model is used to describe the motion and using the RANSAC (Random Sample Consensus) algorithm, the fundamental transformation matrix is calculated. Then, the feature points are transformed sing this matrix. The distance between points is calculated with L_2 norm. Finally, a set of correspondences of points exceeding a specific threshold is determined. The process is repeated K-steps and the fundamental matrix that provides the largest number of feature correspondences is selected. A Kalman filter is used for intentional motion estimation.

In this way, computational systems are generally decoupled from the physical hardware elements, providing a robust alternative to changing hardware elements, being invariant to the physical system and easily scalable, which can translate into cost reduction.

By applying digital image processing algorithms, the aim is to reduce vibrations and unintentional movements of the video. The computational stabilization process consists of the following steps: First, motion estimation, followed by motion correction and finally, frame compensation. While the concept of motion estimation and correction are similar to

mechanical stabilization, frame compensation is a new action introduced in this type of stabilization. Specifically, frame compensation refers to the fill that must be introduced into the image once the motion has been corrected so that the image maintains its original dimensionality. It can be said to be a padding action.

In the following sub-sections, the computational stabilization is focused on the three stages mentioned above, which are described in more detail in the following sections.

3.3.2.1. Digital image motion

A video is a sequence of images at different time moments called frames. To measure motion in a video we rely on the premise that the points of light that define an area in an image undergo minimal alterations, either in shape, color, brightness, etc... between adjacent frames. Thus, motion is defined as a correlation of successive images defined by a trajectory or displacement. In the work of F. Xiao et al. [34] the effect of motion in a digital image is shown by capturing a single point of light with long exposure time, thus motion can be observed in digital images.

3.3.2.1.1. Movement models

Based on the way the motion space is represented in the image, a model can be used to represent it in 2D, 3D. Also, the motion transformations can be described by 2D methods, 3D methods or something in between called 2.5D. The 2D motion models define the image space as a two-dimensional environment with two coordinates. The types of affine transformations that can occur in these models include rotations, translations, scaling or the combination of all of them such as projective (Homographic) transformations. The affine transform preserves parallelism and midpoints and satisfies transformations such as rotations, scaling, shearing or reflections. The projective transform is more general than the affine transform.

The advantage of projective models is that they are more robust and computationally less expensive, so they are suitable for real-time applications. The main disadvantage is that they are valid only for flat scenes and do not work well in scenes with a lot of depth. 3D motion models define the image space as a three-dimensional environment with three coordinates. They require defining a structure for the motion that contains many characteristic points. 3D model transformations have the advantage of good performance with deep scenes. As a disadvantage is that they do not work well on frames without depth variation, and it is computationally very expensive, so they are not usually applied for real time solutions [35]–[37].

Some of the methods optimized for 2D motion models perform as well as methods used for 3D models while maintaining the robustness of 2D methods. These intermediate methods

are called 2.5D and reduce the distance between 2D and 3D methods by relaxing the requirements for 3D reconstruction [38], [39].

3.3.2.1.2. Types of approximation for motion estimation in videos

Pixel-based algorithms or feature-based algorithms can be used for motion estimation. Pixel-based algorithms, also called direct methods, check the pixel-by-pixel brightness changes between two adjacent frames. These algorithms can perform image-to-image matching with high accuracy. The main drawbacks are the large computational load on the correspondences and that, if in some scenario it is not executed correctly, the information can drop below a certain level and smear the area (brightness modifications) [40].

Feature-based algorithms compare points of interest or descriptors. Descriptors are points that are considered characteristic and invariant under certain conditions. Typically, these methods are based on identifying and tracking descriptors associated with characteristic points or regions such as corners, edges or textures. In this way, the computational load is focused on regions of the image that contain a large amount of information rather than on the entire image [41]. In the work of S. Battiato et al. [42] the authors use a feature-based approach by employing the SIFT (Scale Invariant Feature Tracker) technique. The trajectories of the points are evaluated between frames to estimate the motion computed by Euclidean distance. A modified version of the iterative least squares method is used to avoid estimation errors. An adaptive motion vector integration filter MVI (Motion Vector Integrator) is used to separate the intentional motion.

3.3.2.1.3. Frame matching

The goal is to compare two consecutive frames and find the matching matrix between them. The correspondence matrix, or transformation matrix between the H-frames, indicates the spatial transformation that the observer has undergone. Finding this transformation matrix is widely known as Perspective-n-Point (PnP) [16]. Depending on the type of approach selected (direct or descriptor methods), different algorithms are used to relate the points between adjacent frames. Traditional correlation methods such as least squares or optimization use all the information contained to calculate the matrix parameters. Algorithms for motion estimation are generally based on techniques of four types: gradient techniques, pixel recursion techniques, block matching techniques, and frequency-based techniques; more details in [22].

The sensitivity of PnP methods to the descriptors or features of one frame being well related to that of the next frame requires an association strategy. Algorithms such as RANSAC (Random Sample Consensusson) [43], are able to compute the parameters of the correspondence matrix from a data set with errors by an iterative process that removes outliers which allows to improve the correspondence between frames and finally the

estimation of the transformation matrix. Different metrics such as SSD (Sum of Squared Differences), SAD (Sum of Absolute Differences), MAD (Mean of Absolute Differences), or correlation are used to calculate the error.

Finally, from any of the above methods, a correspondence/transformation matrix H is obtained, which can be expressed as (3.8).

$$F(u,v,z')^T = H \cdot I(u,v,1)^T;\ H = \begin{bmatrix} R^{2\times2} & t^{2\times1} \\ p^{1\times2} & h_{33} \end{bmatrix} | t = [t_x, t_y] \tag{3.8}$$

where the original image is $I(u,v)^T$ and its homogeneous coordinates are $I(u,v,1)^T$. The resulting image in homogeneous coordinates is $F(u,v,z')^T$. If the type of transformation H is not projective $z' = 1$, $h_{33} = 1$ y $p^{1\times2} = 0^{1\times2}$. The combination of the terms of H, allow us to perform translations, rotations, scalings and perspective transformations. Considering $h_{33} = 1$ and $p^{1\times2} = 0^{1\times2}$, the matrix H corresponds to the set of affine transformations known as homographic matrix. Thus, the terms of t correspond to pixel translations.

The $R^{2\times2}$ matrix allows scaling, shearing, rotations, etc. For scaling $R = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$, where the terms s_x and s_y means the scale factor in x and y axis respectively. If $s_i < 1$ the images are reduced, while if $s_i > 1$, the result is that of zooming in on each of their $i = \{x, y\}$ coordinates. Shearing on the image I is produced by applying an $R = \begin{bmatrix} 1 & -j_x \\ -j_y & 1 \end{bmatrix}$, where the terms $j_{\{x,y\}}$ of the matrix indicate the tilt on each of the coordinates. A rotation $\theta$with the rotation axis perpendicular to the image coordinate origin, upper left corner of the image, can be expressed by $R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$. To change the center of rotation to the coordinates $(c_x, c_y)^T$, the matrix H is the combination of two transformations, a rotation θ and a translation t that keeps the central coordinate $(c_x, c_y)^T$ invariant. This is achieved by $t_x = c_x(1-\cos\theta) - c_y \sin\theta$, $t_y = c_y(1-\cos\theta) + c_x \sin\theta$.

Finally, applying the inverse homographic matrix H on the last frame allows to reduce the variation it has undergone with respect to the previous frame, stabilizing the image. More details on computational stabilization can be found at [44]–[47].

3.3.2.2. Example of computational correction

The objective of this subsection is to illustrate an example of computational correction in which it is intended to computationally determine the transformation undergone by an image and compare it with reality in order to estimate the degree of accuracy of the basic computational correction system. Given two images taken by the same vision system. The

second image is the result of rotating 90º degrees the first one and capture by the vision system.

A feature-based approach is used for motion estimation. To extract the descriptors of the two images to be compared, the ORB (Oriented FAST and Rotated BRIEF) method is used, then a point matching is performed between the two frames and the best correspondences are filtered. Finally, the homographic transformation matrix is calculated from the calculated point correspondences.

The ORB feature extractor is a point detector that combines the FAST method with the performance enhancement offered by BRIEF (Binary Robust Independent Elementary Features). The FAST (Features from Accelerated Segment Test) algorithm consists of discriminating corner points as an invariant element to create a descriptor. To discriminate the feature points, a central pixel P and sixteen pixels in a circle around it are taken. If the intensity of at least twelve consecutive pixels of the sixteen pixels has an intensity equal to the intensity of P plus a threshold or minus that threshold, the point is chosen as the descriptor. The BRIEF (Binary Robust Independent Elementary Features) algorithm trains decision trees to recognize intensity patterns and correlate them between two images. This method is effective correlating images with different perspective but is less sensitive to correlate elements that undergo rotations.

A FLANN (Fast Library for Approximate Nearest Neighbors) matcher is used for the correlation of the minutiae of two images, which contains algorithms that match in a fast and efficient way using a clustering and search in a multidimensional space. Using the KNN (k-nearest neighbors) method, the matched points are filtered. These points are used in the PnP problem to compute the homographic transformation matrix. Fig. 3.5 shows the ORB features extracted over the same scene at two different times Fig. 3.6 shows the correlation of the points by applying the FLANN matching algorithm.

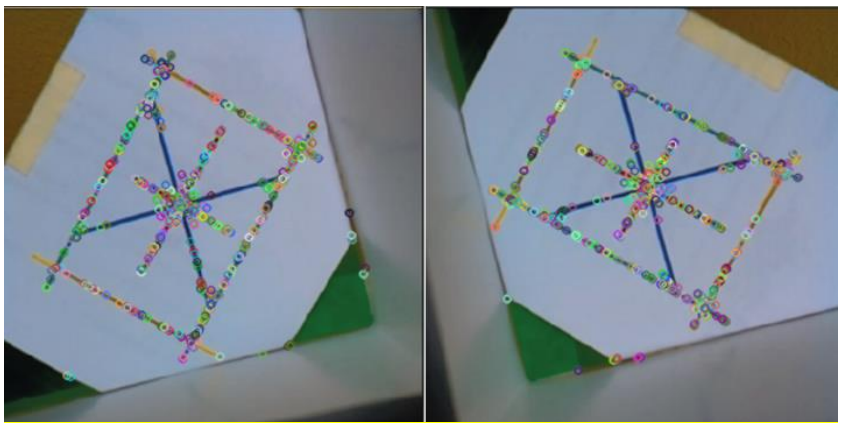


Fig. 3.5. Features identified by the ORB algorithm for two images belonging to the same scene but rotated 90º.

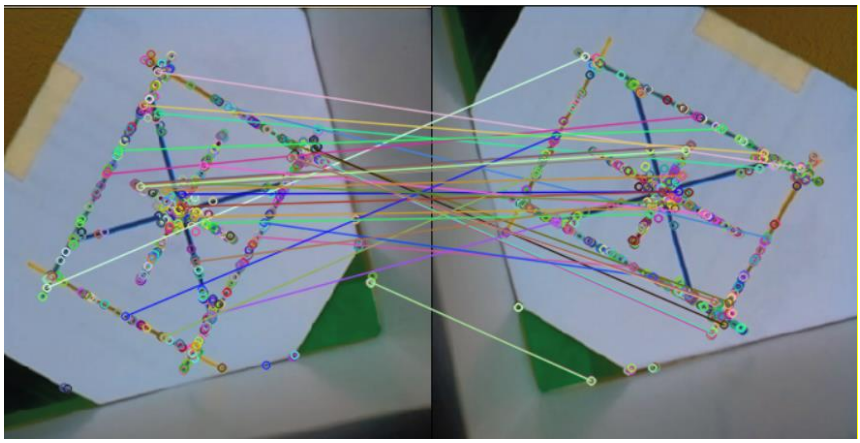


Fig. 3.6. Relationship of feature points using the FLANN matcher for images rotated 90°.

To check the accuracy of the system, four rotation tests are performed.

Fig. 3.7 shows two frames in which the main transformation is a -90º rotation. The inverse homographic matrix is applied and finally the initial frame and the corrected one are compared on the last image.

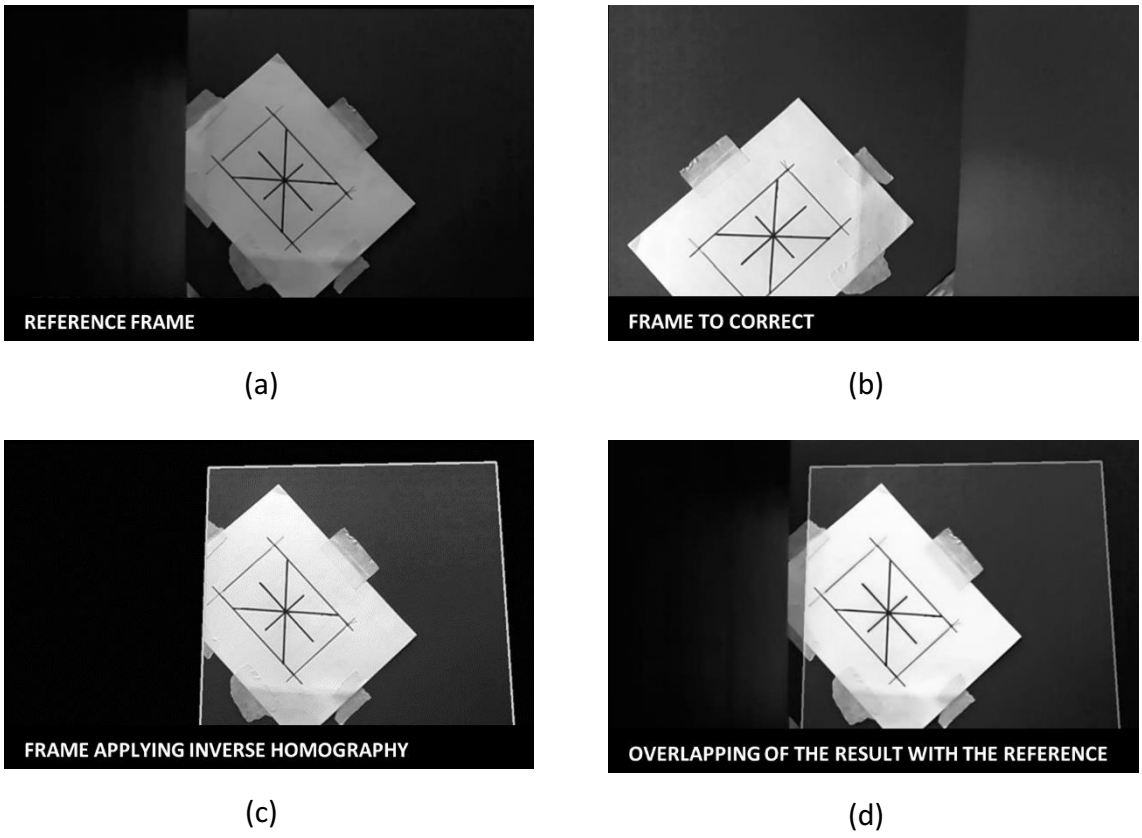


Fig. 3.7. Results of the computational stabilization process for a -90º rotation. (a) Reference image, (b) Image to be corrected, (c) Image corrected, (d) Image (c) overlapped on (a).

To quantitatively compare the results, the homographic transformation is considered to correspond to an affine transformation without scaling. Considering (3.8), $\hat{\theta} = \tan^{-1}\frac{\sin\hat{\theta}}{\cos\hat{\theta}} = \tan^{-1}\frac{h_{12}}{h_{22}}$. Under these considerations the results of Table 3.1 are obtained. It is observed that the error in estimation is less than 8% so that for minor perturbations of the vibration style can be a good video correction strategy.

Table 3.1: Summary of rotation results.

Rotated angle θ [º]	**Estimated angle $\hat{\theta}$ [º]**	**Error [º]**
-90,00	-88,95	-1,05
90,00	96,87	-6,87
-90,00	-88,78	-1,22
90,00	94,88	-4,88

3.4. Object detection

Object detection is a big challenge for vision systems. According to Youzi Xiao in [48] "*The essence of object detection is to locate and classify objects, which uses rectangular bounding boxes to locate the detected objects and classify the categories of the objects"*. This area of computer vision is closely related to classification, semantic segmentation and instance segmentation. While object detection classifies at the bounding boxes level, semantic segmentation and instance segmentation detects and classifies at the pixel-level. In addition, instance segmentation can differentiate between different objects of the same class. For example, in an image with two persons, semantic segmentation identifies them as person and instance segmentation identifies person 1 and person 2 [48]. Classical approaches in object detection invest efforts in defining what is to be detected to identify regions of interest (ROI), feature extraction and finally classification. Classical examples can be the identification of moving objects in an image sequence, some practical cases are related to security or vehicle identification. Incorporating a classifier at the output of the regions of interest allows the identification of expected objects such as different types of vehicles. Some of the techniques widely used in conventional image classification are Support Vector Machines (SVM) and Boosting.

However, challenges that are simple for a human, such as differentiating between a chair and an elephant, can be very complicated for a machine because classifiers need "good" features to differentiate them. In this context, "good" features are understood as the set of characteristics that are able to differentiate the spaces of the set to be classified. Some of the classical feature extraction strategies are the identification of corners with Harris or Shi-Tomasi algorithms, the extraction of main features such as Scale-invariant Feature Transform

(SIFT), Speeded-Up Robust Features (SURF), Features from Accelerated Segment Test (FAST), Binary Robust Independent Elementary Features (BRIEF), Oriented FAST and Rotated BRIEF (ORB) or Histograms of Oriented Gradients (HOG).

Although descriptors have been and continue to be widely used for a multitude of applications, when it comes to object detection, current trends are driven by deep learning. Advances over the last 10 years in deep learning, challenges such as the "ImageNet Large Sacale Visual Recognition Challenge" and computational capacity have catapulted image classification strategies based on convolutional neural networks (CNN). These networks are essentially composed of two blocks, the first one in charge of feature extraction and capable of learning the features that best classify a set of classes, and another classification block. Undoubtedly the main leap in object identification concerns the feature extraction phase.

Work such as that of Youzi Xiao et al. [48], classifies object detection algorithms in a first level by "handcrafted features" and "Learned features". Inside the first group are pioneering algorithms such as Viola-Jones [49] widely used in face detection systems, oriented gradient detectors (HOG) [50], dimension-based partitioning and merging clustering (DPM) [51], Oxford-MKL [52], NLPR-HOGLBP among others. Regarding the level of "Learned features", it is also generalized in the literature the classification of detectors between two-stage object detection and one-stage object detection. While the two-stage object detection group separates the object localization task from the classification task, the one-stage object detection strategies perform the identification and classification tasks in the same phase thanks to the combination of an output function that is a regressor and a classifier. One of the main advantages of two-stage detection is the accuracy of detection, however, they are very slow. In contrast, single-stage detection is very fast, but the detection accuracy is generally lower than two-stage detectors. One of the most prominent architectures for two-stage object detection are the Region-based Convolutional Neural Networks (R-CNN) [53], [54] and for one-stage object detection the You Only Look Once (YOLO) networks [55] is undoubtedly one of the most recognized. In both cases, one/two-stage detectors, the output is composed of a ROIs vector and an associated class.

The learning problem of each of these branches focuses roughly on the cost function of the optimization problem associated with learning. In review papers such as the one by Youzi Xiao et al. [48], a section of loos function for object detection in deep learning approach is included that classifies cost functions associated with classification, regression and multitasking that allow the reader to go down to the level of optimization required by supervised classification methods.

On the other hand, the growing demand for CNN-based computer vision technologies on lightweight devices such as cell phones has boosted the study of lightweight approximations

of networks from the previous groups, these groups are referred to as "Lightweight networks" [56].

Object detection with the above strategies corresponds to supervised learning methods and is limited to the identification of categories found within the training database. Identifying new objects involves a new image labeling process on a large volume of data and retraining. However, this may not be possible when the number of images available for the new class to be identified is very low with respect to other classes. Novel approaches in detection such as Few-Shot Object Detection (FSOD) [57], [58] seek to recognize new (unseen) classes of objects using few examples. According to the work of G. Huang et al. [59] "the standard approach in few-shot object detection was to pretrain a backbone for ImageNet classification, then train an object detector on top of this backbone on the base classes, and finally finetune on the novel classes". Breakthroughs in self-supervised learning have made it possible to initialize the backbone of FSOD methods from representations with pretext tasks, opening up new opportunities for study.

Throughout this section it is shown the main challenges faced by object detectors, the main evaluation metrics, and an example of object detection using cutting edge strategies embedded in UAV computer companion systems.

3.4.1. Problems of object detection

Object detection systems present a multitude of challenges in real-world scenes. Each new environment brings new challenges, but in general, detection problems can be classified into four main groups:

- **Occlusions**: Real-world scenes show situations in which the objects to be identified are not fully observed. These situations are considered occlusions. Occlusions generally result in a loss of object information that can lead to a loss of detection or false detection. The situations can be very different. Classic examples of occlusions can be shadows on objects or overlapping in the field of view of different objects. Detection of objects of the same class or not in crowded scenarios such as pedestrian detection [60] or vehicle traffic detection [61] is essential for certain applications. Classical solutions to these problems use additional object or context information such as object gray information, local feature information or object boundary information. More recent work such as Kai Chen et al. [62] combines GANs (Generative Adversial Nets) to generate images with occlusions. This new data set with occlusions is used to train a Faster R-CNN detector. This allows the detector to be more robust to occlusions.
- **Multi-scale object detection:** Detecting objects at different scales is a continuous challenge for detectors. Both a scaled-down object and a scaled-up object with respect to the detector's learning data set can be considered to lose defining information. For

this reason, multi-scale object detection presents a major challenge for object detection systems. From a Deep Learning point of view, detection approaches based on two-stage object detection have proven to be more robust in multi-scale detection than one-stage object detection systems. While conventional CNN approaches such as Faster R-CNN and YOLO use features of the last convolutional layer for the regression cost function, recent approaches [63], [64] use features of different convolutional stages. These strategies are referred to as multi-layer feature fusion and multi-layer detection.

- **Class imbalance:** A typical problem in any classification problem is bias data [65]. In object detection problems, a classification phase or stage is required. Systematic reviews such as [48] mention how one-stage detectors are vulnerable to data bias, being more robust systems that propose candidate regions such as two-stage object detectors. From the point of view of object detection, works such as those of TY. Lin et al. [66] have been proposed, which act on the cross-entropy function using weights, making it easier for the network to pay more attention to the smaller classes.
- **Redundant bounding boxes:** The redundancy of the bounding-boxes means to deduplicate the repeated boxes on the same object and to select only the most accurate bounding-boxes. The deduplication operation of the bounding-boxes can be used for the intermediate process of object detection and processing of the final results. This essential element in the object detector pipeline is called Non-Maximum Suppression (NMS). To explain the classical operation of an NMS, let us consider a detector that proposes several ROIs ($\mathcal{B} = \{b_1, b_2, \dots b_i\}$) with a different score (s_i) on the different classes (c_j^i) $\mathcal{S} = \{s_1, s_2, \dots s_i\} | s_i = \{c_1^i, c_2^i, \dots, c_j^i\}$. The main idea of traditional NMS is to discard the worst scoring bounding boxes and keep the highest scoring one with a threshold value of intersection over junction (IoU) (3.10). Within Faster R-CNN architectures, NMS can adjust the number of candidate ROIs to speed up the detection process, for one-stage detectors such as YOLO and SSD use the NMS to generate the ultimate detector region. Current work such as that of C. Guo et al. [67] proposes an attention model-based NMS algorithm called Attention-based nonmaximum suppression (ANMS) that shows promising results when implemented on a Faster R-CNN architecture based on the VGG16 network and tested with 4 classical databases.
- **Detection speed:** Detection speed is a fundamental and sometimes critical indicator for many applications. In this sense, one-stage networks and especially ligthweight networks are good candidates. Especially ligthweight networks minimize computational operations by reducing their architecture, thus reducing inference time. Moreover, one-stage networks are showing amazing results in terms of performance as reflected in the work of the latest model YOLOv7 and its reduced version YOLOv7-tiny [68], where the authors claim that YOLOv7 outperforms all known detectors in speed and accuracy in the range of 5 frames per second (FPS) to 160 FPS.

3.4.2. How to evaluate object detection?

There are a multitude of proposals for the evaluation of object detection systems depending on the final goal. However, the decoupled evaluation of object detection, the classification of regions of interest, is widespread in the literature. On the other hand, in the comparison of embedded systems, metrics can be used to evaluate inference time, energy consumption or computational performance, among others.

In terms of detection, the mean hit ratio (MHR) per image and the intersection over union (IoU) can be used. The first metric compares the number of correctly identified ROIs (3.9) per image and the second, IoU evaluates the quality of the fit of the bounding boxes getting from the detection. In (3.9) the number of ROIs identified in each i-image is divided by the ground truth ROIs image. When all ROIs contained in the ground truth image (class ROI (i)) are identified, 1 is obtained, false positives are identified in case of values >1 and in case of <1, detections are lost. The results for all i-test images are summed and divided by the total number of images. Thus, the best MHR=1. IoU is defined as the ratio of the area of the overlapping region between two given ROIs to the area of their union, equation (3.10). These ROIs can be those of detection and ground truth. The coordinates considered in each case are normalized to the image size, thus ensuring that the IoU provides a bounded measure between 0 and 1 of the goodness of fit between the detection frame obtained during inference and the test.

$$MHR = \frac{\sum_{i}^{N.Images} \frac{ROIs(i)}{class\ ROI(i)}}{N.Images} \tag{3.9}$$

$$IoU = \frac{Intersection\ Area}{Union\ Area} \tag{3.10}$$

The classification performance can be evaluated by the combination using a two-dimensional space with the accuracy (3.11) and the F-Score (3.13) as shown in [65].

The first metric, accuracy, indicates an overall accuracy of the positives, however in an unbalanced sample a classifier can show a very high accuracy without detecting some classes, so the classifier's will also be unbalanced. For this reason, its typical use the harmonic mean between precision and recall (3.12) called F_β-Score or F_1. Where β is a weight indicating the importance of the precision. If precision and recall have same relevant, $\beta = 1$. The union of these two metrics create a $\mathbb{R}^2$-space of accuracy in which the ideal classifier would be at coordinates (1,1).

$$Acc = \frac{Tp + Tn}{Tp + Tn + Fp + Fn} \tag{3.11}$$

$$Precision = \frac{Tp}{Tp + Fp};\ Recall = \frac{Tp}{Tp + Fn} \tag{3.12}$$

$$F_\beta - Score = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{\beta . Precision + Recall} \tag{3.13}$$

where Tp, Tn, Fp and Fn mean the set of combinations between true-false and positive or negative cases in the confusion matrix. In addition, accuracy-F_1plane show an interesting indicator of the classifier performance. Another widely used metric is the Receiver Operating Characteristic (ROC) curve corresponding to the precision-recall curve. The closer the ROC curve is to the upper left corner, the better the performance of the detector is. The area under the ROC curve (AUC) is another indicator of classifier quality, the closer it is to 1 the better the performance of the classifier.

The evaluation of computational efficiency can be measured by the mean inference time (MIT) as shown in equation (3.14). This ratio is the result of the quotient of the sum of the inference time for detection on each of the images by the number of total images.

$$MIT = \frac{\sum_i^{Num.Images} Inference\ time(i)}{Num. Images} \tag{3.14}$$

It is important to note that the general detection systems discussed in this section are systems that are decoupled from aircraft, so the metrics presented above are also aircraft independent. This means that the evaluation of a vision detector integrated in a UAV can be the same as for any other detection application that is not integrated in a UAV.

3.4.3. Object detection example

The aim of this example is to test the performance of object detection technologies from a drone using a lightweight and low-cost companion computer from different approaches: Classical Haar Detector, Single-Soth Detection (SSD) deep detector and a semantic segmentation system. Although the problem of semantic segmentation is a different problem than object detection, its close relationship to identifying regions of interest and classifying them can be related to pixel-level detection. In addition, these systems can be used in applications that require object detection. One of the main advantages of these strategies is to be able to identify whole regions that cannot necessarily be delimited by a bounding box such as roads, the horizon, etc.

The detectors proposed for this example are pre-trained in different conditions where inference is desired. The tests proposed in this example aim to test the robustness of these systems to a change of perspective conditions.

As for the detection with the Haar-cascade algorithm, it is used to identify the face using as support our companion computer. The model used is the one that appears in the OpenCV library [69]. The system uses a trained classifier to detect faces, but within the OpenCV library there are a multitude of classifiers trained for use with the Haar detector. The format of these classifiers is an XML file.

Haar algorithm has two phases, identifying regions of interest and classifying. As for classification, a classifier can be used, and supervised learning of a particular training set is used. A difficulty of the classifer is to localize exactly where in an image an object resides. To localize the ROI, the algorithm uses a brute force solution named sliding window. Haar features are extracted on the image and the classification is performed in different phases that increase its computational complexity. If any of the classifiers rejects the window, the process does not continue, and the region is rejected. This allows to reduce the number of sliding windows and to speed up the detection process. Detailed information can be found in the classic paper by Paul Viola and Michaels Jones “Rapid Object Detection Using a Boosted Cascade of Simple Features” [70].

SSD has two components: a backbone model and SSD head. Backbone model usually is a pre-trained image classification network as a feature extractor. The SSD network selected is MobilNet (developped by Google) because it is optimized for having a small model size and faster inference time. This is ideal to run on mobile phones and resource-constrained devices like a Raspberry Pi. For this architecture, the backbone results in a 300x300x3 feature maps for an input image. The input layer defines the dimension of the image to inference, in this case (300x300 pixels resolution). This architecture can be placed in the one-stage detection and Lightweight networks group of detectors. The architecture is downloaded from the repository [71] and is trained with 20 classes from the COCO-2017 dataset [72]. The categories for which it is trained are: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, TV monitor.

The semantic segmentation architecture used for this example is ENet [73]. The reasons are the ENet is about the authors said in their paper “the ENet is up to 18×faster, requires 75×less FLOPs, has 79×less parameters, and provides similar or better accuracy to existing models like the SegNet [74]”.

The architecture is trained with the Cytiscape database [75] with a set of 20 classes categorized into unlabelled, road, sidewalk, building, wall, fence, pole, traffic light, traffic sign, vegetation, terrain, sky, person, rider, car, truck, bus, train, motorcycle, and bicycle.

Generally, supervised learning detection models are trained on external high-powered equipment and then the architectures trained are exported on the companion computers to

be used in the inference phase. Some devices such as the Neural Compute Stick 2 (NCS2) [76] are intended to improve computational performance in inference processes, so they can be used in devices such as the raspberry pi to improve the performance of applications. NCS integrates an Intel Movidius Myriad X vision processing unit (VPU) [77] and a set of tools with OpenVINO API support [78]. A vision processing unit is a class of microprocessor; it is a specific type of AI accelerator, designed to accelerate machine vision tasks. This software development kit enables rapid prototyping, validation, and deployment of deep neural networks.

In the example detailed below, a Raspberry Pi 2 Model B (RPI), a Pi camera and an NCS2 have been used as a companion computer. The camera resolution is set to 640x480 with a framerate of 32 fps. The inference of the different architectures requires an adjustment in size of the images to the dimensions of the input layer of the networks and the total inference time on each image changes. The pickup system is installed on the underside of a DJI family of exactocopters.

Fig. 3.8 shows the device located under the vehicle. This system allows the field of view to be changed at different inclinations. Specifically for the tests, an inclination above the horizon of 0.10, 45 and 90 degrees is used, the latter being a zenithal image.

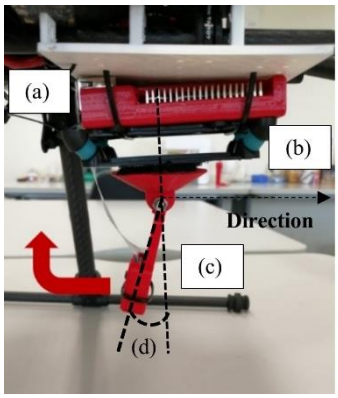


Fig. 3.8. Computer vision devise. a) Raspberry Pi computer companion, b) Vibration Stabilizer, c) Pi Camera, d) Tilt angle.

Fig. 3.9 shows examples of applying the SSD and ENet semantic segmentation detectors described above to different flight conditions. The first case, Fig. 3.9 (a-b) refers to the use of images in an urban environment from a standard perspective of a person or a vehicle. The SSD is able to identify a person and a car. The processing speed directly with the accompanying computer is 0.5 FPS, however, by adding the NCS-2 the speed increases to 6 FPS. For the ENet case it is able to identify the person, vehicle, sidewalk and other feature classes from the Cityscape dataset. The inference time is high even with NCS-2, being 0.7 FPS.

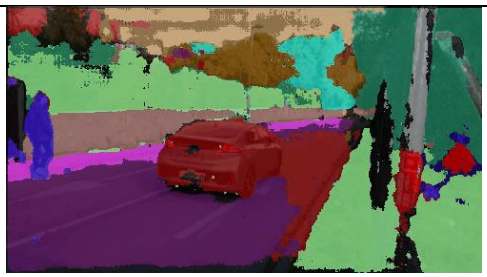
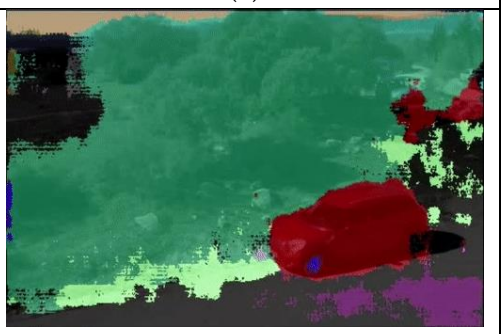
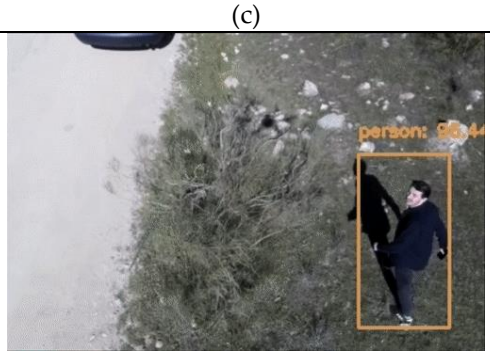
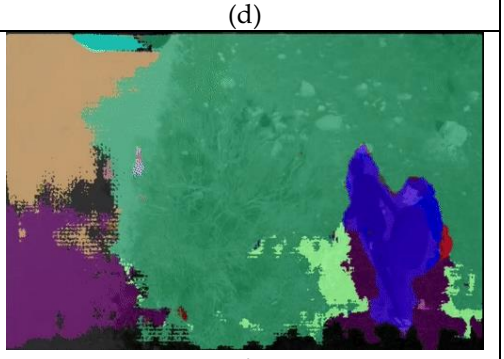
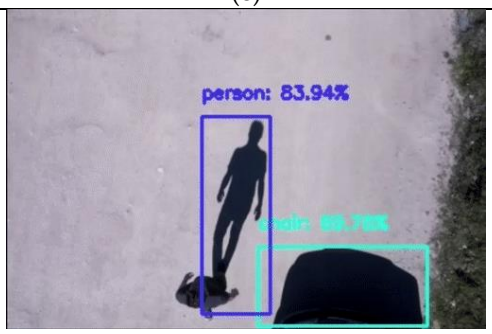
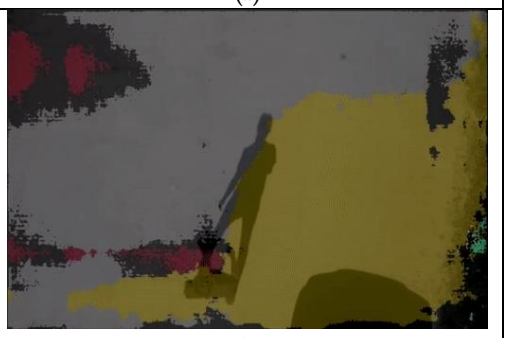
Inclination [º]	SSD	Semantic Segmentation
0	 (a)	 (b)
10	 (c)	 (d)
45	 (e)	 (f)
90	 (g)	 (h)

Fig. 3.9. Example of detection-segmentation by different vision angles. a-b) Detection and semantic segmentation in urban environment; c-d) Detection and semantic segmentation in a forest environment with a 10º camera tilt from a drone at lower altitude. e-f) Forest environment with a 45º camera tilt from a drone at lower altitude. g-h) Forest environment with a 90º camera tilt (zenithal) from a drone at lower altitude.

The second case Fig. 3.9 (c-d) shows a low altitude flight in a rural environment with urban elements (vehicles). SSD is able to identify a vehicle in the foreground, but not vehicles in the background or with shadow occlusions. In the case of ENet for these conditions the system correctly identifies the region of the main vehicle and a small region of those in the background. It also identifies the forest area behind the vehicle but has the problem of inference time.

In the third case, with a low altitude flight and a camera inclination of 45º. Both the SSD and ENet systems identify the person in the scene but include in the identification the shadow produced by the person. In addition, the semantic segmentation identifies the forest area over which the person is walking.

Finally, in the last of the tests, a zenithal plane is used to capture images with a low altitude flight in the same environment as the previous two cases. On this occasion SSD does not correctly identify the person and presents a false positive on the shadow that it produces on the ground, in the case of Fig. 3.9 (h) the system becomes unstable and does not correctly identify the classes.

Table 3.2 shows a summary of the information taken from the tests described above. It can be seen how the inference time of an SSD is much longer than that of a semantic segmentation architecture. Although the times are slow compared to real-time video (>24FPS), one must consider the low-cost devices that have been used.

Table 3.2. Deep Learning Object Detection Test Summary.

	SSD	**Semantic segmentation**
Time process (RPI & NCS2)	6 FPS	1.5 FPS
Model architecture	MobilNet	ENet
Accuracy	70%	59,5%
Several classes	Yes	Yes
Bounding box	Yes	Pixel-wise
Compatible RPi	Yes	Yes but not adviced

The tests show evidence of lack of invariance with respect to the point of view of the captured images, so that for an embedded system it will be necessary to apply retraining strategies, or few-shot detection to improve the accuracy of detection from embedded systems. In addition, for real-time applications with accompanying computers, it will be necessary to study more powerful embedded systems to lower the inference time on each frame of the captured images.

3.5. Visual object tracking

Visual object tracking (VOT) can be considered a subfield in computer vision given the many opportunities and challenges it presents itself. Some of these challenges are occlusion,

background clutter, illumination changes, scale variation, low resolution, fast motion, out of view, motion blur, deformation, in and out planer rotation [79].

VOT is the process of identifying a region of interest in a sequence and consists of four consecutive elements, including target initialization, appearance modeling, motion prediction, and target positioning. It is important to describe each of the phases so as not to confuse VOT with detection. Target initialization is the process of initially annotating the position of the object, or region of interest, using representations such as object bounding boxes, ellipse, centroid, object skeleton, object outline, or object silhouette. Appearance modeling consists of identifying visual features of the object for better representation of a region of interest and effectively building mathematical models to detect objects using learning techniques. In motion prediction, the target location or ROI is estimated in subsequent frames. The estimation approximates the position of the ROI in the next frame, in positioning this information is used together with that of the visual model to fix the most accurate location of the target (ROI) using search algorithms such as the voracious algorithm used in works such as K. Shafique and M. Shah [80].

In VOT two scenario levels can be considered, Single Object Tracking (SOT) and Multiple Object Tracking (MOT). In SOT the goal is to track a single target among a set of objects over a sequence of frames. MOT aims at tracking multiple targets along a given sequence of frames. In addition, annual challenges such as the Visual Object Tracking Challenge [81] group VOT tracker challenges into four different groups [82], [83]: Short-term tracker (ST_0), Short-term tracker with conservative updating (ST_1), Pseudo long-term trackers (LT_0) and Re-detecting long-term tracker (LT_1). The first two, ST_0 and ST_1, the target position is reported at each frame. Although ST0 and ST1 do not implement target redetection and are very sensitive to occlusions, ST1 increases robustness by selectively updating the visual model based on a tracking confidence estimation mechanism. In the cases of LT0 and LT1, the target position is not reported in frames when the target is not visible. The difference between LT0 and LT1 lies in the fact that in LT0 The tracker does not implement explicit target re-detection but uses an internal mechanism to identify and report tracking failure. On the other hand, the LT1 trackers detects tracking failure and implements explicit target re-detection.

Although MOT, also known as Multi Target Tracking (MTT), systems are outside the scope of this section (more information in [84], [85]), it is important to note that they present the additional problem to SOT systems in that they require a phase of associating ROIs between frames [86]. Among the most popular tools that implement VOT algorithms are OpenCV [86] (chosen for this section), DeepSORT [87], [88], Object Tracking MATLAB [89] or MDNet [90], [91].

This section focuses on SOT systems implemented in the OpenCV open-source software tool. To conclude the section, an example of object tracking system is presented from a drone based on a visual object tracking system.

3.5.1. Visual object tracking: classical approach

Regarding object tracking, the research of A. Brdjanin, et al. [92] or that of NS Raghava and his colleagues K. Gupta, I. Kedia and A. Goyal [93] provide great information of single object tracking algorithms as well as a benchmark for each of them from the data obtained when used in different scenarios like illumination or scale variation, occlusions, deformation, etc. The trackers present in their work are Boosting, Multiple Instance Learning (MIL), MedianFlow, Minimum Output Sum of Squared Error (MOSSE), Kernelized Correlation Filter (KCF), Generic Object Tracking Using Regression Networs (GOTURN) and Channel and Spatial Reliability Tracker (CSRT). Finally, the metrics that were used are the average success rate and computational speed in frames per second (FPS).

Works like that of Jin-Hyeok Park et al. [94] show how an object tracking algorithm can be used with AirSim simulator. In this case, an advanced Deep Reinforcement Learning-Based DQN Agent Algorithm was used. In a similar way, the work of E. Bondi et al. [95] depicts a very specific use of AirSim simulations with object detection and tracking, using it for detection of poachers in a custom-built African savanna environment.

Some of the most popular object tracking algorithms in mentioned literature, implemented in OpenCV tracking API [96] are the following:

- **Boosting** [97]: A more than a decade old algorithm. Based on an online version of AdaBoost, which is the algorithm inside HAAR cascade face detector. Given that the objective has already been detected in the first frame, the algorithm assumes that that is the objective of the tracking and everything else is background. Then, the classifier runs over every pixel close to those of the previous location and a score is stored. The updated location will be that where the score is maximum. When more and more frames have been checked, the classifier is updated.
- **MIL (Multiple Instance Learning**) [98]. The underlying idea for this tracker is somewhat similar to that of the Boosting tracker. The most important difference is that it divides the original selection and its surroundings into multiple smaller ones. These sections are identified and separated into positives and negatives “bags” for its later use. The whole positive bag does not equal all positive examples, but at least one of them will be. Giving a high chance that in the bag there is at least one image with the target nicely centred.
- **KCF (Kernelized Correlation Filters)** [99]. Building upon the ideas of the previous two, the KCF tracker makes use of the fact that om the positives bag of the MIL tracker, overlapping regions exist which leads to some mathematical properties that can be used

for making the tracker more accurate and also faster. Basically, the tracker uses filtration over a frame to find the target.

- **CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability (DCF-CSR))**[100]**.** The filter works by using a correlation filter trained with HoG (Histograms of Gradients.) and Colour Names. With the initial frame input, specific weights are added in each frame by the filter, therefore, training it for its repetition in the following frames. The spatial reliability map adjusts the filter support to the selected region on the frame.
- **MedianFlow** [101]**:** This tracker uses multiple frames (Both forward and backwards in time.) to evaluate where the object will be in the next frame of analysis. By minimizing this "ForwardBackward" error, tracking failures can be detected and reliable trajectories selected. This set of characteristics allows it to work well in video sequences with predictable trajectories of the target and no occlusion of it.
- **TLD (Tracking, Learning, Detection)** [102]. Designed with long term tracking in mind, the three pillars of the functioning of this tracker are three. Short term tracking, learning and detection. The tracking component is based on the MedianFlow algorithm with some modifications to improve failure detection. Learning component is based on P-N learning and detection component uses a scanning-window grid.
- **MOSSE (Minimum Output Sum of Squared Error)** [103]**.** This tracker works by using an adaptive correlation filter in the tracking of the object. The complete video footage is pre-processed, and stable correlation filters are produced when initialized using a single frame. As the video progresses the filter is updated on the pre-processed domain.
- **GOTURN (Generic Object Tracking Using Regression Networs)** [104]**.** This deep learning algorithm learns a generic relationship between the motion and the appearance of the object using a regression-based approach to object tracking. Essentially, they do a straightforward regression to locate target objects with a single pass feed through the network. The network takes two inputs: a search region from the current frame and a target from the previous frame. The network then compares these images to find the target object in the current image.

3.6. References

[1] M. Roy, "The role of internet of things (IoT) and big data as a road map for smart management systems: Case studies across industries," *ASEE Annu. Conf. Expo. Conf. Proc.*, vol. 2018-June, 2018, doi: 10.18260/1-2--31123.

[2] H. Chao, Y. Gu, and M. Napolitano, "A survey of optical flow techniques for robotics navigation applications," *J. Intell. Robot. Syst. Theory Appl.*, vol. 73, no. 1–4, pp. 361–372, 2014, doi: 10.1007/s10846-013-9923-6.

[3] G. Conte and P. Doherty, "An integrated UAV navigation system based on aerial image matching," *IEEE Aerosp. Conf. Proc.*, 2008, doi: 10.1109/AERO.2008.4526556.

[4] J. R. G. Braga, H. F. C. Velho, G. Conte, P. Doherty, and É. H. Shiguemori, "An image matching system for autonomous UAV navigation based on neural network," *2016 14th Int. Conf. Control. Autom. Robot.*

Vision, ICARCV 2016, vol. 2016, no. November, pp. 13–15, 2017, doi: 10.1109/ICARCV.2016.7838775.

[5] A. Macario Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, "A Comprehensive Survey of Visual SLAM Algorithms," *Robotics*, vol. 11, no. 1, 2022, doi: 10.3390/robotics11010024.

[6] S. Krul, C. Pantos, M. Frangulea, and J. Valente, "Visual slam for indoor livestock and farming using a small drone with a monocular camera: A feasibility study," *Drones*, vol. 5, no. 2, 2021, doi: 10.3390/drones5020041.

[7] D. Zou, P. Tan, and W. Yu, "Collaborative visual SLAM for multiple agents:A brief survey," *Virtual Real. Intell. Hardw.*, vol. 1, no. 5, pp. 461–482, 2019, doi: 10.1016/j.vrih.2019.09.002.

[8] S. Saleh, S. Manoharan, and W. Hardt, "Real-time 3D Perception of Scene with Monocular Camera," *Embed. Selforganising Syst.*, vol. 7, no. 2, pp. 4–7, 2020, doi: 10.14464/ess.v7i2.436.

[9] T. van Dijk, "Self--Supervised Learning for Visual Obstacle Avoidance," 2020, [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3Abf982743-f043-49c1-a502-12f3a91b739e

[10] A. Al-Kaff, D. Martín, F. García, A. de la Escalera, and J. María Armingol, "Survey of computer vision algorithms and applications for unmanned aerial vehicles," *Expert Syst. Appl.*, vol. 92, pp. 447–463, 2018, doi: 10.1016/j.eswa.2017.09.033.

[11] J. P. Llerena, J. García, and J. M. Molina, "Error Reduction in Vision-Based Multirotor Landing System," *Sensors 2022, Vol. 22, Page 3625*, vol. 22, p. 3625, 2022.

[12] K. C. Liao, H. Y. Wu, and H. T. Wen, "Using Drones for Thermal Imaging Photography and Building 3D Images to Analyze the Defects of Solar Modules," *Inventions*, vol. 7, no. 3, 2022, doi: 10.3390/inventions7030067.

[13] M. A. Hassan *et al.*, "A rapid monitoring of NDVI across the wheat growth cycle for grain yield prediction using a multi-spectral UAV platform," *Plant Sci.*, vol. 282, no. October 2017, pp. 95–103, 2019, doi: 10.1016/j.plantsci.2018.10.022.

[14] "Image APIs - AirSim." https://microsoft.github.io/AirSim/image_apis/#available-imagetype-values (accessed Dec. 09, 2022).

[15] R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision, Second Edition," 2000, Accessed: Oct. 28, 2022. [Online]. Available: www.cambridge.org/9780521540513

[16] S. Eivazi Adli, M. Shoaran, and S. M. Sayyed Noorani, "GSPnP: simple and geometric solution for PnP problem," *Vis. Comput.*, vol. 36, no. 8, pp. 1549–1557, Aug. 2020, doi: 10.1007/S00371-019-01747-X/FIGURES/21.

[17] A. Datta, J. S. Kim, and T. Kanade, "Accurate camera calibration using iterative refinement of control points," *2009 IEEE 12th Int. Conf. Comput. Vis. Work. ICCV Work. 2009*, pp. 1201–1208, 2009, doi: 10.1109/ICCVW.2009.5457474.

[18] OpenCv, "Home - OpenCV." https://opencv.org/ (accessed Jul. 21, 2021).

[19] T. Collins and A. Bartoli, "Infinitesimal plane-based pose estimation," *Int. J. Comput. Vis.*, vol. 109, no. 3, pp. 252–286, Jul. 2014, doi: 10.1007/S11263-014-0725-5/FIGURES/14.

[20] J. Dong and H. Liu, "Video Stabilization for Strict Real-Time Applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 4, pp. 716–724, Apr. 2017, doi: 10.1109/TCSVT.2016.2589860.

[21] A. Yousaf, K. Khurshid, M. J. Khan, and M. S. Hanif, "Real time video stabilization methods in IR domain

for UAVs - A review," *2017 5th Int. Conf. Aerosp. Sci. Eng. ICASE 2017*, pp. 1–9, Jun. 2018, doi: 10.1109/ICASE.2017.8374287.

[22] M. Ahmed, "Digital video stabilization-review with a perspective of real time implemention," *Int. Conf. Recent Innov. Signal Process. Embed. Syst. RISE 2017*, vol. 2018-January, pp. 296–303, Jun. 2018, doi: 10.1109/RISE.2017.8378170.

[23] J. Dong, Y. Xia, Q. Yu, A. Su, and W. Hou, "Instantaneous video stabilization for unmanned aerial vehicles," *https://doi.org/10.1117/1.JEI.23.1.013002*, vol. 23, no. 1, p. 013002, Jan. 2014, doi: 10.1117/1.JEI.23.1.013002.

[24] S. Ertürk, "Image sequence stabilisation: Motion vector integration (MVI) versus frame position smoothing (FPS)," *Int. Symp. Image Signal Process. Anal. ISPA*, vol. 2001-January, pp. 266–271, 2001, doi: 10.1109/ISPA.2001.938639.

[25] X. Chen, C. Wang, T. Zhang, C. Hua, S. Fu, and Q. Huang, "Hybrid Image Stabilization of Robotic Bionic Eyes," *2018 IEEE Int. Conf. Robot. Biomimetics, ROBIO 2018*, pp. 808–813, Jul. 2018, doi: 10.1109/ROBIO.2018.8664900.

[26] "Stable Diffusion Online." https://stablediffusionweb.com/ (accessed May 24, 2023).

[27] W. Rone, Y. Liu, P. B.-T.-B. & biomimetics, and undefined 2018, "Maneuvering and stabilization control of a bipedal robot with a universal-spatial robotic tail," *iopscience.iop.org*, 2018, doi: 10.1088/1748-3190/aaf188.

[28] "Settings - AirSim." https://microsoft.github.io/AirSim/settings/#gimbal (accessed Nov. 12, 2022).

[29] A. Hamza, R. Hafiz, M. M. Khan, Y. Cho, and J. Cha, "Stabilization of panoramic videos from mobile multi-camera platforms," *Image Vis. Comput.*, vol. 37, pp. 20–30, May 2015, doi: 10.1016/J.IMAVIS.2015.02.002.

[30] W. C. Hu, C. H. Chen, T. Y. Chen, M. Y. Peng, and Y. J. Su, "Real-time video stabilization for fast-moving vehicle cameras," *Multimed. Tools Appl.*, vol. 77, no. 1, pp. 1237–1260, Jan. 2018, doi: 10.1007/S11042-016-4291-4/TABLES/4.

[31] C. H. Chen, T. Y. Chen, W. C. Hu, and M. Y. Peng, "Video Stabilization for Fast Moving Camera Based on Feature Point Classification," *Proc. - 2015 3rd Int. Conf. Robot. Vis. Signal Process. RVSP 2015*, pp. 10–13, Feb. 2016, doi: 10.1109/RVSP.2015.11.

[32] S. Liu, L. Yuan, P. Tan, J. S.-P. of the I. Conference, and undefined 2014, "Steadyflow: Spatially smooth optical flow for video stabilization," *cv-foundation.org*, Accessed: Nov. 12, 2022. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2014/html/Liu_SteadyFlow_Spatially_Smooth_2014_CVPR_paper.html

[33] S. Kim, S. Kang, … T. W.-I. T. on, and undefined 2013, "Feature point classification based global motion estimation for video stabilization," *ieeexplore.ieee.org*, Accessed: Nov. 12, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6490269/

[34] F. Xiao, J. E. Farrell, P. B. Catrysse, and B. Wandell, "Mobile Imaging: the big challenge of the small pixel," *Digit. Photogr. V*, vol. 7250, p. 72500K, 2009, doi: 10.1117/12.806616.

[35] Y. Furukawa, B. Curless, … S. S.-2010 I. computer, and U. 2010, "Towards internet-scale multi-view stereo." [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5539802/

[36] C. Wu, "Towards linear-time incremental structure from motion," in *Proceedings - 2013 International*

Conference on 3D Vision, 3DV 2013, 2013, pp. 127–134. doi: 10.1109/3DV.2013.25.

[37] N. Jiang, Z. Cui, and P. Tan, "A global linear method for camera pose registration," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 481–488. doi: 10.1109/ICCV.2013.66.

[38] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala, "Subspace video stabilization," *ACM Trans. Graph.*, vol. 30, no. 1, Jan. 2011, doi: 10.1145/1899404.1899408.

[39] A. Goldstein and R. Fattal, "Video stabilization using epipolar geometry," *ACM Trans. Graph.*, vol. 31, no. 5, Aug. 2012, doi: 10.1145/2231816.2231824.

[40] M. Irani and P. Anandan, "About direct methods," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2000, vol. 1883, pp. 267–277. doi: 10.1007/3-540-44480-7_18.

[41] P. H. S. Torr and A. Zisserman, "Feature based methods for structure and motion estimation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2000, vol. 1883, pp. 278–294. doi: 10.1007/3-540-44480-7_19.

[42] S. Battiato, G. Gallo, G. Puglisi, and S. Scellato, "SIFT features tracking for video stabilization," in *Proceedings - 14th International conference on Image Analysis and Processing, ICIAP 2007*, 2007, pp. 825–830. doi: 10.1109/ICIAP.2007.4362878.

[43] M. A. Fischler and R. C. Bolles, "Random sample consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, doi: 10.1145/358669.358692.

[44] B. Tordoff and D. W. Murray, "Guided sampling and consensus for motion estimation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2002, vol. 2350, pp. 82–96. doi: 10.1007/3-540-47969-4_6.

[45] K. Y. Lee, Y. Y. Chuang, B. Y. Chen, and M. Ouhyoung, "Video Stabilization using Robust Feature Trajectories," in *Proceedings of the IEEE International Conference on Computer Vision*, 2009, pp. 1397–1404. doi: 10.1109/ICCV.2009.5459297.

[46] A. Litvin, J. Konrad, and W. C. Karl, "Probabilistic video stabilization using Kalman filtering and mosaicing," in *Image and Video Communications and Processing 2003*, 2003, vol. 5022, p. 663. doi: 10.1117/12.476436.

[47] Y. Matsushita, E. Ofek, X. T.-… R. (CVPR'05), and U. 2005, "Full-frame video stabilization", Accessed: Nov. 12, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1467248/

[48] Y. Xiao *et al.*, "A review of object detection based on deep learning," *Multimed. Tools Appl.*, vol. 79, no. 33–34, pp. 23729–23791, Sep. 2020, doi: 10.1007/s11042-020-08976-6.

[49] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, vol. 1. doi: 10.1109/cvpr.2001.990517.

[50] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005, vol. I, pp. 886–893. doi: 10.1109/CVPR.2005.177.

[51] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010, doi: 10.1109/TPAMI.2009.167.

[52] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, “Multiple Kernels for object detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2009, pp. 606–613. doi: 10.1109/ICCV.2009.5459183.

[53] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. doi: 10.1109/CVPR.2014.81.

[54] A. Shrivastava, A. Gupta, and R. Girshick, “Training region-based object detectors with online hard example mining,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-Decem, pp. 761–769. doi: 10.1109/CVPR.2016.89.

[55] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-Decem, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[56] Y. Zhou, S. Chen, Y. Wang, and W. Huan, “Review of research on lightweight convolutional neural networks,” in *Proceedings of 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference, ITOEC 2020*, Jun. 2020, pp. 1713–1720. doi: 10.1109/ITOEC49072.2020.9141847.

[57] S. Antonelli *et al.*, “Few-Shot Object Detection: A Survey,” *ACM Comput. Surv.*, vol. 54, no. 11 S, pp. 1–37, Jan. 2022, doi: 10.1145/3519022.

[58] M. Köhler, M. Eisenbach, and H.-M. Gross, “Few-Shot Object Detection: A Comprehensive Survey,” Dec. 2021, Accessed: Nov. 12, 2022. [Online]. Available: http://arxiv.org/abs/2112.11699

[59] G. Huang, I. Laradji, D. Vazquez, S. Lacoste-Julien, and P. Rodriguez, “A Survey of Self-Supervised and Few-Shot Object Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2022, doi: 10.1109/TPAMI.2022.3199617.

[60] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua, “Computer vision and deep learning techniques for pedestrian detection and tracking: A survey,” *Neurocomputing*, vol. 300, pp. 17–33, Jul. 2018, doi: 10.1016/j.neucom.2018.01.092.

[61] F. Zhang, C. Li, and F. Yang, “Vehicle detection in urban traffic surveillance images based on convolutional neural networks with feature concatenation,” *Sensors (Switzerland)*, vol. 19, no. 3, p. 594, Jan. 2019, doi: 10.3390/s19030594.

[62] K. Chen *et al.*, “Hybrid task cascade for instance segmentation,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, vol. 2019-June, pp. 4969–4978. doi: 10.1109/CVPR.2019.00511.

[63] Q. Zheng and Y. Chen, “Feature pyramid of bi-directional stepped concatenation for small object detection,” *Multimed. Tools Appl.*, vol. 80, no. 13, pp. 20283–20305, May 2021, doi: 10.1007/s11042-021-10718-1.

[64] B. Singh and L. S. Davis, “An Analysis of Scale Invariance in Object Detection - SNIP,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3578–3587. doi: 10.1109/CVPR.2018.00377.

[65] J. P. Llerena, J. García, and J. M. Molina, “LSTM vs CNN in Real Ship Trajectory Classification,” in *16th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021). SOCO 2021. Advances in Intelligent Systems and Computing*, Springer, Cham, 2022, pp. 58–67. doi: 10.1007/978-3-030-87869-6_6.

[66] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” *Proc. IEEE*

Int. Conf. Comput. Vis., vol. 2017-Octob, pp. 2999–3007, 2017, doi: 10.1109/ICCV.2017.324.

[67] C. Guo, D. Zhou, M. Cai, N. Ying, H. Chen, and J. Zhang, "ANMS : attention-based non-maximum suppression," pp. 11205–11219, 2022.

[68] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," pp. 1–15, 2022, [Online]. Available: http://arxiv.org/abs/2207.02696

[69] "OpenCV: Cascade Classifier." https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (accessed Nov. 12, 2022).

[70] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," *Cvpr*, vol. 1, pp. I-511-I–518, 2001, [Online]. Available: http://ieeexplore.ieee.org/document/990517/

[71] "GitHub - djmv/MobilNet_SSD_opencv: MobilNet-SSD object detection in opencv 3.4.1." https://github.com/djmv/MobilNet_SSD_opencv (accessed Nov. 12, 2022).

[72] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 740–755, 2014, doi: 10.1007/978-3-319-10602-1_48/COVER.

[73] A. Paszke, A. Chaurasia, S. Kim, E. C. preprint arXiv, and undefined 2016, "Enet: A deep neural network architecture for real-time semantic segmentation," *arxiv.org*, Accessed: Nov. 12, 2022. [Online]. Available: https://arxiv.org/abs/1606.02147

[74] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017, doi: 10.1109/TPAMI.2016.2644615.

[75] M. Cordts *et al.*, "The cityscapes dataset for semantic urban scene understanding," *openaccess.thecvf.com*, Accessed: Nov. 12, 2022. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2016/html/Cordts_The_Cityscapes_Dataset_CVPR_2016_paper.html

[76] "Intel® Neural Compute Stick 2." https://www.intel.es/content/www/es/es/products/sku/140109/intel-neural-compute-stick-2/specifications.html (accessed Nov. 12, 2022).

[77] "Intel® Movidius™ Vision Processing Units (VPUs)." https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html (accessed Nov. 12, 2022).

[78] "Intel® Distribution of OpenVINO™ Toolkit." https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html (accessed Nov. 12, 2022).

[79] M. Fiaz, A. Mahmood, S. Javed, S. J.-A. C. Surveys, and undefined 2019, "Handcrafted and deep trackers: Recent visual object tracking approaches and trends," *dl.acm.org*, vol. 52, no. 2, p. 43, May 2019, doi: 10.1145/3309665.

[80] K. Shafique, M. S.-I. transactions on pattern analysis, and undefined 2005, "A noniterative greedy algorithm for multiframe point correspondence," *ieeexplore.ieee.org*, Accessed: Nov. 12, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1359751/

[81] "VOT Challenge." https://votchallenge.net/ (accessed Nov. 12, 2022).

[82] M. Kristan, A. Leonardis, J. Matas, M. F.-… on C. Vision, and undefined 2020, "The eighth visual object tracking VOT2020 challenge results," *Springer*, vol. 12539 LNCS, pp. 547–601, 2020, doi: 10.1007/978-3-030-68238-5_39.

[83] A. Lukeźič, L. Zajc, T. Vojíř, … J. M.-I. transactions on, and undefined 2020, "Performance evaluation methodology for long-term single-object tracking," *ieeexplore.ieee.org*, Accessed: Nov. 12, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9054960/

[84] S. Zhou, M. Ke, J. Qiu, and J. Wang, *A survey of multi-object video tracking algorithms*, vol. 842. Springer International Publishing, 2019. doi: 10.1007/978-3-319-98776-7_38.

[85] Y. Dai, Z. Hu, S. Zhang, and L. Liu, "A survey of detection-based video multi-object tracking," *Displays*, vol. 75, p. 102317, 2022, doi: 10.1016/j.displa.2022.102317.

[86] P. Emami, P. M. Pardalos, L. Elefteriadou, and S. Ranka, "Machine Learning Methods for Data Association in Multi-Object Tracking," *ACM Comput. Surv.*, vol. 53, no. 4, 2020, doi: 10.1145/3394659.

[87] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," *Proc. - Int. Conf. Image Process. ICIP*, vol. 2017-Septe, pp. 3645–3649, 2018, doi: 10.1109/ICIP.2017.8296962.

[88] "GitHub - nwojke/deep_sort: Simple Online Realtime Tracking with a Deep Association Metric." https://github.com/nwojke/deep_sort (accessed Nov. 12, 2022).

[89] "Tracking and Motion Estimation - MATLAB & Simulink - MathWorks España." https://es.mathworks.com/help/vision/tracking-and-motion-estimation.html (accessed Nov. 12, 2022).

[90] H. Nam and B. Han, "Learning Multi-domain Convolutional Neural Networks for Visual Tracking," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 4293–4302, 2016, doi: 10.1109/CVPR.2016.465.

[91] "GitHub - hyeonseobnam/MDNet: Learning Multi-Domain Convolutional Neural Networks for Visual Tracking." https://github.com/HyeonseobNam/MDNet (accessed Nov. 12, 2022).

[92] A. Brdjanin, N. Dardagan, … D. D.-… on In. in, and undefined 2020, "Single object trackers in opencv: A benchmark," *ieeexplore.ieee.org*, Accessed: Nov. 12, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9194647/

[93] N. S. Raghava, K. Gupta, I. Kedia, and A. Goyal, "An Experimental Comparison of Different Object Tracking Algorithms," *Proc. 2020 IEEE Int. Conf. Commun. Signal Process. ICCSP 2020*, pp. 726–730, Jul. 2020, doi: 10.1109/ICCSP48568.2020.9182101.

[94] J. H. Park, K. Farkhodov, S. H. Lee, and K. R. Kwon, "Deep Reinforcement Learning-Based DQN Agent Algorithm for Visual Object Tracking in a Virtual Environmental Simulation," *Appl. Sci. 2022, Vol. 12, Page 3220*, vol. 12, no. 7, p. 3220, Mar. 2022, doi: 10.3390/APP12073220.

[95] E. Bondi *et al.*, "Near Real-Time Detection of Poachers from Drones in AirSim.," *ijcai.org*, 2018, Accessed: Nov. 12, 2022. [Online]. Available: https://www.ijcai.org/Proceedings/2018/0847.pdf

[96] S. Mallick, "Object Tracking using OpenCV (C++/Python)." https://learnopencv.com/object-tracking-using-opencv-cpp-python/ (accessed Jul. 21, 2021).

[97] H. Grabner, E. Zurich, H. Bischof, and M. Grabner, "Real-Time Tracking via On-line Boosting AUTOVISTA View project Nonlinear intra-modality registration View project Real-Time Tracking via On-line Boosting," 2014, doi: 10.5244/C.20.6.

[98] B. Babenko, … M. Y.-2009 I. C. on, and undefined 2009, “Visual tracking with online multiple instance learning,” *ieeexplore.ieee.org*, Accessed: Nov. 15, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5206737/

[99] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “LNCS 7575 - Exploiting the Circulant Structure of Tracking-by-Detection with Kernels,” 2012.

[100] A. Lukežič, T. Vojíř, L. Lukaˇ, L. Zajc, J. Matas, and M. Kristan, “Discriminative Correlation Filter with Channel and Spatial Reliability”.

[101] Z. Kalal, K. Mikolajczyk, J. M.-2010 20th international, and undefined 2010, “Forward-backward error: Automatic detection of tracking failures,” *ieeexplore.ieee.org*, Accessed: Nov. 15, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5596017/

[102] Z. Kalal, K. Mikolajczyk, J. M.-I. transactions on pattern, and undefined 2011, “Tracking-learning-detection,” *ieeexplore.ieee.org*, Accessed: Nov. 15, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6104061/

[103] D. Bolme, J. Beveridge, … B. D.-2010 I. computer, and undefined 2010, “Visual object tracking using adaptive correlation filters,” *ieeexplore.ieee.org*, Accessed: Nov. 15, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5539960/

[104] D. Held, S. Thrun, and S. Savarese, “Learning to Track at 100 FPS with Deep Regression Networks”, Accessed: Nov. 15, 2022. [Online]. Available: http://davheld.github.io/GOTURN/GOTURN.html

Chapter 4: Error Reduction in Vision-Based Multirotor Landing System

Juan Pedro Llerena Caña *, Jesús García Herrero and José Manuel Molina López

Applied Artificial Intelligence Group (GIAA), University Carlos III de Madrid, 28270 Colmenarejo (Madrid), Spain; jllerena@inf.uc3m.es (J.P.LL.), jgherrer@inf.uc3m.es (J.G.) and molina@ia.uc3m.es (J.M.M.).

* Correspondence: jllerena@inf.uc3m.es

Abstract: New applications are continuously appearing with drones as protagonists, but all of them share an essential critical maneuver—landing. New application requirements have led the study of novel landing strategies, in which vision systems have played and continue to play a key role. Generally, the new applications use the control and navigation systems embedded in the aircraft. However, the internal dynamics of these systems, initially focused on other tasks such as the smoothing trajectories between different waypoints, can trigger undesired behaviors. In this paper, we propose a landing system based on monocular vision and navigation information to estimate the helipad global position. In addition, the global estimation system includes a position error correction module by cylinder space transformation and a filtering system with a sliding window. To conclude, the landing system is evaluated with three quality metrics, showing how the proposed correction system together with stationary filtering improves the raw landing system.

Keywords: UAV; autonomous landing; filtering; computer vision; helipad context; global position; navigation system; SITL

Nomenclature

Reference frames (RF)		p_i^j	Position of element i in j RF
$\{\boldsymbol{t}\}$	Helipad (target)	${}^jT_i$	Linear translation i to j RF
$\{\boldsymbol{ph}\}$	Pinhole model image plane	$[{}^jR_i \vert v_i^j]$	Rotation and translation between i and j RF
$\{\boldsymbol{c}\}$	Camera	${}^jR_i$	Rotation between i and j RF.
$\{\boldsymbol{z}\}$	Camera socket	v_i^j	Translation between i and j RF.
$\{\boldsymbol{g}\}$	Gimbal	${}^jF_i$	Nonlinear RF transformation between i and j
$\{\boldsymbol{b}\}$	Body gravity center	f_x	Focal length
$\{\boldsymbol{n}\}$	North-East-Down (NED)	f_y	
$\{\boldsymbol{e}\}$	Earth-Centerd Earth-Fixed (ECEF)	c_x	Principal point
$\{\boldsymbol{g}\}$	Global WGS84 datum	c_y	
$\boldsymbol{\theta, \phi, \psi}$	Euler angles	k_i	Distortion coefficients
$\boldsymbol{\lambda, \varphi, h}$	Longitude, Latitude, altitude	A	Intrinsic camera matrix
$\mathbb{R}$	Real number space	δ	Intrinsic camera parameters
$\boldsymbol{\mathcal{B}}$	Boolean number	Ψ	General camera model
$\boldsymbol{\mathcal{P}^2}$	Planar projective space	$\hat{E}$	Reprojection error
$\boldsymbol{x}, \hat{\boldsymbol{x}}$	State and estimated state	O	Image feature function
$\|\boldsymbol{x}\|$	Euclidian norm of x	$\mathcal{N}$	Normal distribution
$\boldsymbol{L}$	Sliding window size	μ	Mean
$\boldsymbol{\beta_i}$	Constant bias of coordinate i.	σ	Standard deviation

4.1. Introduction

The growing demand for drone applications motivates the study of the support technologies for this type of small and powerful unmanned aerial vehicle (UAV). However, all new applications share an essential and critical maneuver—landing.

Generally, work in the literature about landing maneuvers, both for fixed and rotary wing UAVs, focuses on control strategies [1–3]. All of them require access to the internal vehicle states, the actuators or specific modes of the control or navigation system.

Under the precision landing concept is included all the solutions that approach this maneuver in an autonomous or supervised way, independent of the techniques and sensors used to estimate the vehicle states such as position or orientation, as well as its corresponding velocities and accelerations. The landing maneuver can be included within the navigation system where two main groups can be distinguished, outdoor and indoor navigation. Generally, outdoor navigation is based on the Global Navigation Satellite System (GNSS), but practically all current systems use fusion techniques that allow the integration of different strategies for estimating one or more vehicles' states, which is necessary for the control and/or navigation system. Some of the most common cases in navigation systems are the use of barometers and/or sonars to improve the accuracy of the altitude provided by GNSS, or the use of small zenith cameras to determine small horizontal displacements [4]. Other specific navigation techniques such as visual odometry or visual Simultaneous Location And Mapping (SLAM) [5] are beginning to be used in indoor navigation.

Thus, other landing works focus on improving the accuracy of instrument systems such as [6–8] or even context information such as safe landing zones, as in work by Shah Alam, Md et al. shown in work [9]. Some commercial solutions focus on the use of beacons that indicate the landing region, as can be seen in the work of J. Janousek and P. Marcon [7] using a commercial infrared light beacon. This type of system includes an external controller to minimize the pixel error between the region of interest (ROI), defined by the centroid of the infrared area and the reference in the image plane, generally located at the center of the image plane. These strategies need to access internal vehicle states such as velocity or acceleration to correct the error.

In terms of context information, vision systems proved to be efficient to identify ROIs. In addition, knowing the landing context, specifically where or how the helipad is where the UAV must land, can help to improve the landing maneuver.

Developing strategies to identify and understand the context information of the aircraft allows providing the systems with greater autonomy. In the survey of autonomous landing techniques for UAVs by Alvika Gautam et al. [1], the authors describe the relationship between

sensors/navigation systems and aircraft control modules, paying particular attention to vision landing techniques, generally responsible for recognizing and estimate the helipad position.

Some civil and commercial UAVs, such as certain DJI models [10], are beginning to integrate vision-based precision landing systems. In the work of Yoakum and Carreta [11], the authors conduct a study of the precision landing system of a DJI Mavic Pro, proving the aircraft and the integrated landing system meet specific accuracy requirements to use a specific wireless charging station.

Generally, vision systems for landings focus on identifying the landing area, either by means of context information of the helipad pattern or by the terrain conditions. The work of Mittal et al. [12] is an example of the identification of landing area conditions, where terrain slope is estimated to verify the feasibility of a UAV to land in urban search and rescue.

Regarding pattern recognition landing systems, works such as [13][14], among others [15–18], focus on finding the position using known patterns by Perspective-n-Point (PnP) algorithms [19]. Patterns such as Aruco [20], charuco, or new fractal patterns such as [21] or the deep learning trend You Only Look Once (YOLO) [22] try to improve the pattern pose estimation and prove to be widespread systems in the literature. In the literature and throughout this paper, an object “pose” means a set of position and heading of a specific reference system.

On the other hand, the emergence of open-source flight controllers such as Pixhawk [23], together with specific communication protocols such as Micro Air Vehicle Link (MAVLink) [24] and multiplatform APIs such as MavSDK [25], help to develop new applications and research new landing strategies.

PX4 [26] autopilot set up as a rotary wing vehicle has a planification system that allows dynamically smoothing the trajectories between different positions [27]. Specifically, it smooths the trajectories between consecutive waypoints by rounding the turns with radii over the waypoints and increasing or decreasing the drone speed when approaching or moving away from a waypoint [28–30]. The guidance algorithm integrated in PX4 is the L_1 algorithm introduced by Park et al. [31] under the linear approach. When forcing a new target position while the vehicle is navigating between two positions, the system changes target and tends to reach the newly added location by smoothing its current trajectory, as shown in works such as Stateczny et al. [32]. This behavior, when repeated with a certain frequency to include new waypoints referring to the same position, but with a certain noise, produces a spin effect on the aircraft that we call “inter-waypoint noise spin effect”.

In this work, we propose using the aircraft guidance system without downing the controller level for a widespread implementation of the precision landing system, contributing to UAV air safety and helping the emergence of new applications.

We propose a new contribution with respect to classical geolocation landing strategies based on global positioning by decoupling the landing in two phases, first reaching the target coordinates and then activating the landing mode, descending vertically with a constant descent speed α.

Our proposed landing strategy seeks to descend quickly when the target is found and to smooth its descent as the aircraft approaches, without having to adjust the controller parameters. This idea attempts to improve the image resolution quickly to improve the position estimation. The difference with respect to other works is that simultaneously descending and adjusting the positioning relies on the variable waypoint altitude adjustment without accessing the controller, so that without changing the internal descent controller of the PX4 [26] or adding new external control laws, the strategy allows smoothing the descent when approaching the target. This strategy allows taking further steps in the final phase of the approximation, improving the final estimate, and ensuring the stability of the system provided by the manufacturer. For this, we propose a function to modulate the default behavior of the PX4 controller which seeks a stationary descent at a constant speed.

In addition, this paper models the error of a precision landing system using a monocular vision system, context information from the helipad pattern, and the internal navigation system of the PX4 flight controller.

The vision system error modeling allows a fine calibration of helipad localization. This correction, together with a stationary filtering of the estimates by sliding time window and variable adjustment of the descent height, allow to reduce the spin effect produced by the estimation error of a vision system when integrated with the $L1$ navigation algorithm, without access to the internal controller parameters or relative states of the aircraft that may require re-programming of the on-board computer.

To sum up, this paper presents two main contributions in precision landing by vision-based global position: the continuous adjustment of the approach to descent trajectory, and the improvement of the position by vision through systematic error adjustment and filtering.

The proposal strategy is evaluated after including an error model correction as well as different sliding time window filters. The work is developed on a hyper-realistic software in the loop (SITL) [33] simulation system with the PX4 flight controller and the AirSim [34] simulator.

Finally, the results of the study show the estimation error analysis and filtering of the estimates with sliding time window filters, minimizing the inter-waypoint noise spin effect generated by noisy waypoint transitions and improving three quality metrics of landing time, trajectory landing length, and landing accuracy without additional control law, enabling the use of the aircraft's guidance system as an alternative for the deployment of precision landing technology.

4.2. Problem Formulation

We consider the problem of a UAV landing on a certain landing pad using its internal autopilot waypoint guidance system and a monocular vision system with gimbal integrated in the UAV.

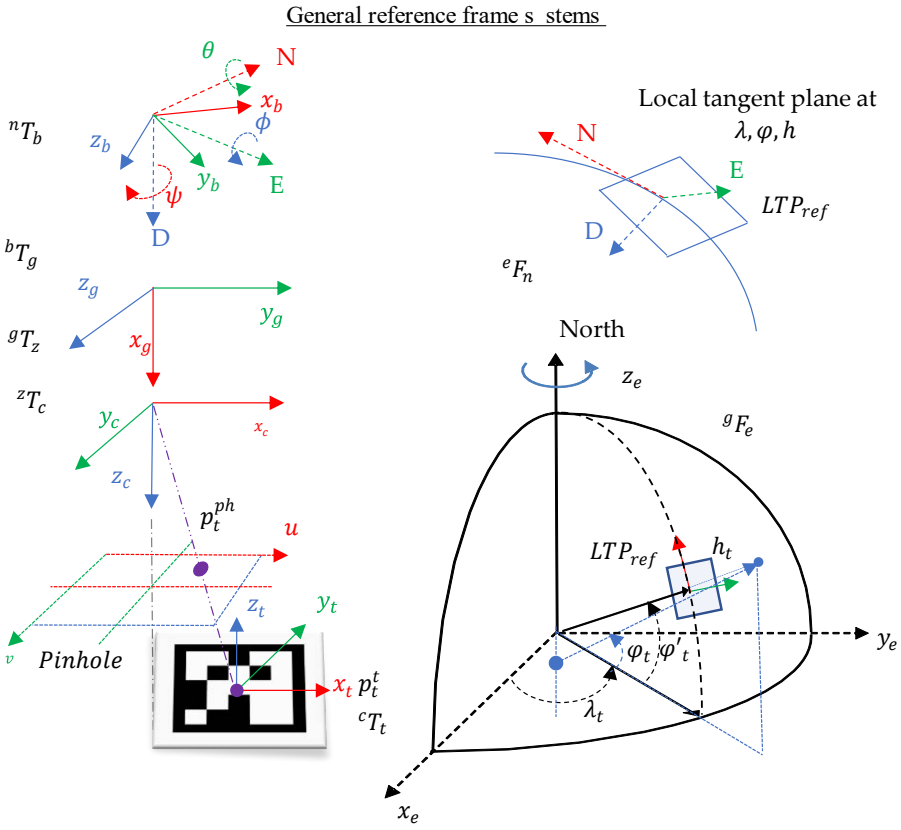


Fig. 4.1. General reference frame systems. Reference frames bottom-left to up: helipad (target), pinhole camera model (image plane), camera, gimbal socket, body, NED. Reference frames right up to down: NED, ECEF, Global.

Fig. 4.1. shows the set of reference frames, where the superscripts $\{t, ph, c, z, g, b, n, e \text{ and } g\}$ correspond to the reference frames of the helipad (target),

pinhole camera model, camera, gimbal socket for the camera, gimbal, body, North-East-Down (NED), Earth-Centered Earth-Fixed (ECEF), and global.

In this way, any given point p_t^t expressed in a flat pattern reference frame $\{t\}$ and the homogeneous transformation ${}^{n}T_t$ between the landing pad reference frame to the NED referent frame $\{n\}$ can be expressed in the global reference frame $\{g\}$ system p_t^g applying a set of transformations shown in (4.1) and (4.2).

$$p_t^n = \overbrace{{}^{n}T_b \cdot {}^{b}T_g \cdot {}^{g}T_z \cdot {}^{z}T_c \cdot {}^{c}T_t}^{{}^{n}T_t} \cdot p_t^t \tag{4.1}$$

$$p_t^g = {}^{g}F_e\left\{{}^{e}F_n[p_t^n, {}^{e}F_g(p_{Ref}^g)]\right\} \tag{4.2}$$

where superscript $\{j\}$ over point p_i^j means the reference frame system, and the subscript $i = \{t, Ref\}$ denotes the name of the point (target and reference). ${}^{j}T_i$ means the homogeneous transformation between reference frame i and j. On the other hand, ${}^{j}F_i$ refers to nonlinear transformations between reference frame i and j. p_{Ref}^g indicates the global position of the body (UAV) as a global reference point.

The set of reference frame systems involved in the transformations are shown in Fig. 4.1 and denoted as: helipad (target) $\{t\}$, pinhole camera model $\{ph\}$, camera $\{c\}$, gimbal socket for the camera $\{z\}$, gimbal $\{g\}$, body $\{b\}$, North-East-Down (NED) $\{n\}$, Earth-Centered Earth-Fixed (ECEF) $\{e\}$, and global $\{g\}$.

4.2.1. Pattern (Helipad) Detection

We consider as the helipad a reference pattern defined by an Aruco pattern [35] with a certain number of bits, as part of a library B. As shown in the paper [20], the system identifies candidate square regions as Aruco markers, then encodes these regions and compares them to the pattern dictionary as desired.

The full process can be divided into the following steps:

- **Image conversion:** Obtain an RGB image and transform it to grayscale.
- **Edge extraction**: We understand as edge an intensity change boundary, some classical algorithms are Canny [36] and Sobel [37].
- **Contour extraction**: We understand a contour as a curve of points without gaps or jumps. Therefore, the objective is to identify if the edges found represent contours. An example of simple contour extraction can be given by a binarized image of an object whose outer contour can be extracted by subtracting the original binarized-dilated image from the original binarized image. To check if closed regions appear, a

segmentation by connected components would provide us candidate regions of interest (ROI) as a result.

- **Contour filtering**: Only show rectangular regions.
- **Removing ROI perspective distortion**: For this it is necessary to find the general plane $\mathcal{P}^2$ projective transformation $h: \mathcal{P}^2 \rightarrow \mathcal{P}^2 | \ h(m) = m' = mH$, where m is a point in a plane. $H^{3\times3}$ is a non-singular matrix where m' is the linear transformation H of m. The transformation H is biunivocal and homogeneous, in other words, a point over a plane is a unique point over another plane and $kH | \ k \in \mathbb{R}$ and $k \neq 0$ is also the solution. This condition allows dividing the matrix H by the element h_{33}, decreasing the dimension of terms to identify from 9 to 8. The correspondence between points $(x_i, y_i) \leftrightarrow (x'_i, y'_i)$ can be expressed in matrix form as $b_i = A_i h$, and their relationship is expressed as (4.3) (more details in [38]). Knowing n pairs of points, the system of $2n$ equations and 8 unknowns is established as $b = Ah$, where $A = [A_1, A_2, \dots, A_n]^T$, $b = [b_1, b_2, \dots, b_n]^T$, and $h^{3\times3}$ matrix as $h_{33} = 1$. For $n = 4$, the direct solution $h = A^{-1}b$; if $n > 4$ the system is overdetermined and least squares can be applied, $h = [A^T A]^{-1} A^T b$. For cases where $h_{33} = 0$ refer to [38].

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}$$
$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{4.3}$$
$$A = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}; \ b = \begin{bmatrix} x' \\ y' \end{bmatrix}; \ h = [h_{11}, h_{12}, h_{13}, h_{21}, \dots h_{32}]^T$$

- **Pattern library matching check:** The binary code of the ROI is extracted, superimposing on the binarized and perspective-corrected image a grid of the same cell size as the searched one. Each grid cell receives a binary value according to if the corresponding color is black (zero) or white (one). The Hamming coding algorithm is applied to the extracted code to eliminate false negatives. This resulting code is compared with the selected pattern dictionary, filtering the regions identified as markers and belonging to the pattern dictionary from other regions. In addition, this step provides information about the marker id if the ROI belongs to the library.

4.2.2. Helipad Pose Estimation

For pose estimation, the Perspective-n-Point (PnP) problem [35] is formulated where the objective is to minimize the reprojection error (4.7) of 3D points in the image plane $\{ph\}$. This problem is closely linked to a calibrated system, since it requires a camera model, pinhole, and a pattern that allows to relate identified features of an image with features of the pattern.

Given a point $p_t^c \in \mathbb{R}^3$ belonging to the knowing pattern located in real-world 3D space and expressed in the camera reference frame $\{c\}$, it can be expressed in the image camera plane reference frame $\{ph\}$ as $p_t^{ph} \in \mathbb{R}^2$. The relationship between the two reference frames is provided by the pinhole camera model in (4.4).

$$sp_t^{ph} = Ap_t^c \tag{4.4}$$

where s is a scale factor and A intrinsic camera matrix [17]. The internal matrix A is composed of the focal distances (f_x, f_y) and the principal points (c_x, c_y). The pinhole model can be improved with radial, tangential, or prism distortion corrections, adding n set of k_i parameters to the model [39–41]. The set of internal parameters of the camera model can be expressed by the vector $\delta = (f_x, f_y, c_x, c_y, k_1, \dots, k_n)$.

If the point is expressed in coordinates of the pattern reference frame p_t^t, there exists an extrinsic homogenous transformation ${}^cT_t$ to relate the reference frame of the pattern to the camera reference frame (4.5) is used.

$$p_t^c = {}^cT_t p_t^t \tag{4.5}$$

where the transformation ${}^cT_t = [{}^cR_t | v_t^c]$ is a rototranslation composed of the pattern's orientation ${}^cR_t = R_x(\theta_1)R_y(\theta_2)R_z(\theta_3)$ to the camera and the pattern position vector to the camera $v_t^c \in \mathbb{R}^3$. Thus, the parameter vector to be identified to obtain the camera–pattern relationship is $\theta = (R, v) = (\theta_1, \theta_2, \dots, \theta_6) \in \mathbb{R}^6$.

Joining (4.4) and (4.5) and adding distortion models, the camera model remains as a Function (4.6) that projects points $p_t^t \in \mathbb{R}^3$ to $p_t^{ph} \in \mathbb{R}^2$ points of the camera image plane.

$$p_t^{ph} = \Psi(\delta, \theta, p_t^t) \tag{4.6}$$

Then, helipad pose estimation is the problem of minimizing the reprojection error (4.7) of the observed helipad pattern features. One of the classic features to identify by computer vision are the corners. If the pattern is known, we know a priori the 3D position of these corners in the reference frame of the pattern.

$$\hat{E} = arg\,min \sum_{p_i^t \in \mathsf{C}} \left[\Psi(\delta, \theta, p_i^t) - O(p_i^t)\right]^2 \tag{4.7}$$

where $p_i^t \in \mathsf{C}$ and C is a corner set of the pattern. $O(p_i^t) \in \mathbb{R}^2$ is the corners obtained in the camera plane by a specific computer vision algorithm such as the Harris or Susan algorithm [22,42].

Furthermore, since all points p_i^t belong to a pattern plane, the z-component of all corners in the pattern frame will always be 0. This quality allows solving (4.7) using specific methods such as the Infinitesimal Plane-Based Pose Estimation (IPPE) [43].

The estimation of internal camera parameters requires a learning phase modeled in (4.7) as an optimization problem. In addition, identifying the six parameters to define the transformation ${}^cT_t$ between the pattern calibration and the camera involves a similar process. Although both processes can be clustered as shown in (4.7), the internal camera parameters δ will be constant for a particular vision system; however, the position of the pattern may change. For this reason, it is decoupled in two phases: on the one hand, a camera parameter learning (calibration) process, using a set of images of a known pattern to estimate internal camera parameters, and, on the other hand, the estimation of the helipad position for a certain image during flight.

4.2.3. Camera-Gimbal Frame

The camera is placed in a camera-gimbal socket, so it is necessary to include this referent frame $\{z\}$. As the camera-gimbal socket axis is equivalent with the general gimbal axes, but static, the ${}^zT_c$ transformation is shown in (4.8).

$$\begin{gathered} {}^zT_c = [R_c^z | 0^{3\times1}] \equiv \begin{pmatrix} R_c^z & 0^{3\times1} \\ 0^{1\times3} & 1 \end{pmatrix}; \\ R_c^z = R\left(x, \frac{\pi}{2}\right) R\left(z, \frac{\pi}{2}\right) \end{gathered} \tag{4.8}$$

where $R(x,\theta)$ and $R(z,\psi)$ represent θ and ψ rotations about the x and z axes of the camera reference system. As the axes of the gimbal and camera-gimbal socket are equivalent, the relationship between camera-gimbal socket and gimbal corresponds to the identity matrix ${}^gT_z = I^{4\times4}$.

4.2.4. Gimbal Body Frame

The gimbal's reference frame $\{g\}$ to the UAV's body gravity center frame is defined as the composition of a roto-translation in (4.9).

$$\begin{gathered} {}^bT_g = [R_g^b | p_g^b] \\ R_g^b = R_g(\theta, \phi, \psi); p_g^b = (x_g, y_g, z_g)^T \end{gathered} \tag{4.9}$$

For this work, we consider the gimbal is static but located at the p_g^b position with $R_g(\theta,\phi,\psi)$ rotation to the body axes.

In this paper we consider ${}^bT_z = \left[R_b\left(0, -\frac{\pi}{2}, 0\right) \middle| (0,0,0.1)^T\right]$, as shown in the Test environment section.

4.2.5. Body-NED Frame

The North-East-Down (NED) frame coordinates $\{n\}$ to the UAV body gravity center $\{b\}$, ${}^{n}T_{b}$ is equivalent to the body rotation at the angle defined by the yaw angle ψ to geographic north or azimuth, pitch attitude to horizon plane ϕ, and roll angle defined to gravity direction θ. These angles refer to the attitude and heading reference system (AHRS) frame of reference that groups magnetic, angular rate, and gravity (MARG) information. Generally, these systems usually include air data to provide altitude or wind speed information.

$$
\begin{gathered}
{}^{n}T_{b} = [R_{b}^{n} | 0^{3\times 1}]; \\
R_{b}^{n} = \begin{pmatrix} \cos\theta\cos\psi & \sin\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \sin\psi\sin\theta\cos\phi + \cos\psi\sin\phi \\ \cos\theta\sin\psi & \cos\psi\cos\theta + \sin\psi\sin\theta\sin\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}
\end{gathered} \tag{4.10}
$$

4.2.6. NED-ECEF-Global

The coordinate transformation between NED to the global reference frame $\{g\}$ requires the use of the Earth-Centered Earth-Fixed (ECEF) reference system $\{e\}$, which allows us to apply the corresponding geodetic transformations to the terrestrial model and finally obtain the coordinates in global terms. In our case, we use a WGS84 (World Geodetic System 84) [44] datum.

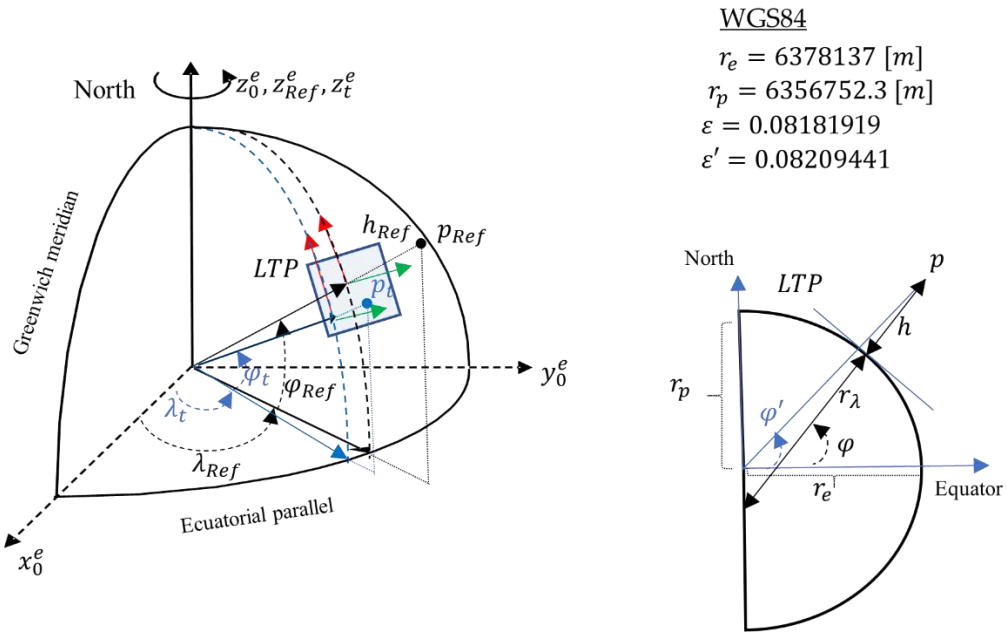


Fig. 4.2. LTP, ECEF, and WGS84 reference systems and geometric relationships.

The constant parameters of the WGS84 datum in Fig. 4.2 refer to: r_e semimajor axis (equatorial radius), r_p semiminor axis (polar axis radius), ε first eccentricity and ε' second eccentricity of the ellipsoid. It is important to differentiate the geocentric coordinates, referred to as the ECEF system, from the geodetic coordinates, referred to as the geodetic

model (WGS84). This difference is provided by the geodetic model (datum) and is represented in the diagram on the right of Fig. 4.2, where φ' refers to geocentric latitude and φ refers to geodetic latitude.

Given a point p_t^n expressed in NED reference frame $\{n\}$ of a local tangent plane (LTP) to a geodesic surface at a known point $p_{Ref}^g = (\lambda, \varphi, h)^T{}_{Ref}$, it can be expressed in ECEF coordinates $\{e\}$ applying Equation (12). This equation corresponds to a translation in ECEF reference frame. However, to obtain p_{Ref}^ecoordinates of our reference point in ECEF frame it is necessary to transform the global coordinates to ECEF applying Equation (4.14). The transformation between local coordinates and ECEF is given by the transformation (4.13). In this work, we consider $p_{Ref}^g = p_{UAV}^g$.

On the other hand, a given point p_t^e expressed in ECEF can be expressed in global coordinates p_t^g applying the transformation (4.11).

$$p_t^g = \begin{pmatrix} \lambda \\ \varphi \\ h \end{pmatrix}_t = {}^gF_e(p_t^e) = \begin{pmatrix} \tan^{-1}\left(\frac{y_t^e}{x_t^e}\right) \\ \tan^{-1}\left(\frac{z_t^e + e'^2 Z_0}{r}\right) \\ U\left(1 - \frac{r_p^2}{r_e V}\right) \end{pmatrix} \tag{4.11}$$

where $(\lambda, \varphi, h)^T$means longitude, latitude, and altitude in WGS84 datum. $(x_t^e, y_t^e, z_t^e)^T$ are the coordinates in the ECEF reference frame. The transformation (4.11) corresponds to Jijie Zhu's algorithm [45] analyzed and compared in [46].

$$p_t^e = \begin{pmatrix} x_t^e \\ y_t^e \\ z_t^e \end{pmatrix} = {}^eF_n(p_t^n, p_{Ref}^e) = R_n^e \cdot p_t^n + p_{Ref}^e \tag{4.12}$$

$$R_n^e = \begin{pmatrix} -\sin\varphi_{Ref}\cos\lambda_{Ref} & -\sin\lambda_{Ref} & -\cos\varphi_{Ref}\cos\lambda_{Ref} \\ -\sin\varphi_{Ref}\sin\lambda_{Ref} & \cos\lambda_{Ref} & -\cos\varphi_{Ref}\cos\lambda_{Ref} \\ \cos\varphi_{Ref} & 0 & -\sin\varphi_{Ref} \end{pmatrix} \tag{4.13}$$

$$p_{Ref}^e = \begin{pmatrix} x_{Ref}^e \\ y_{Ref}^e \\ z_{Ref}^e \end{pmatrix} = \begin{pmatrix} (r_\lambda + h_{Ref})\cos\varphi_{Ref}\cos\lambda_{Ref} \\ (r_\lambda + h_{Ref})\cos\varphi_{Ref}\sin\lambda_{Ref} \\ \left((1-\varepsilon^2)r_\lambda + h_{Ref}\right)\sin\varphi_{Ref} \end{pmatrix} \tag{4.14}$$

$$r_\lambda = \frac{r_e}{\sqrt{1 - \varepsilon^2 \sin^2\varphi}} \tag{4.15}$$

4.3. Proposal

In this section, first the landing strategy is described, then the method to determine the helipad's global position is detailed, and finally the error analysis of the helipad's position estimation is given.

4.3.1. Landing Strategy

The landing strategy is responsible for telling the UAV navigation system the position to which it must go and the attitude it must have to align with the target (Algorithm 4.1). The position of the target is static, but the attitude and altitude to helipad vary overtime when the UAV attempts to land.

Algorithm 4.1 Landing Strategy

$[p_t^g, (\theta, \phi, \psi)_t^g, a]$ = helipad identification
h_{UAv}, ψ_{UAV} = UAV navigation sistem
if a = True
  Buffer ← $[p, (\theta, \phi, \psi)]_t^g$
  if $frequency = 1Hz$ & $Buffer \geq 10$
    $p'^g_t, \psi'^g_t = Filter\ (Buffer)$
    Buffer reset
    Buffer (1) ← $[p',\ \psi']_t^g$
    $\psi_{set} = \psi_{UAV}^n + \psi_t^b$
    $[\lambda, \varphi]_{set} \leftarrow p'^g_t$
    $h_{set} = h_{UAV}(1 - 0.1e^{\frac{-1}{h_t^b - 0.5h_0}})$
    UAV navigation planer ←[$\lambda_{set}, \varphi_{set}, h_{set}, \psi_{set}$]
    if $h_{UAV} \leq h_0$
      PX4 landing mode
      break
    end if
  else
    $goto \rightarrow 1$
  end if
else
  $goto \rightarrow 1$
end if

4.3.1.1. Helipad Azimuth

To align the drone to the marker, it is necessary to determine the azimuth of the marker ψ_t^n. For this, we use the azimuth of the drone ψ_{UAV}^n and the orientation of the marker to the drone ψ_t^b.

Fig. 4.3 shows how the helipad azimuth can be obtained graphically by adding to the drone azimuth the orientation of the aircraft to the landing pad in (4.16). When both systems are aligned, the marker azimuth will be equal to the drone azimuth.

$$\psi_t^n = \psi_{UAV}^n + \psi_t^b \tag{4.16}$$

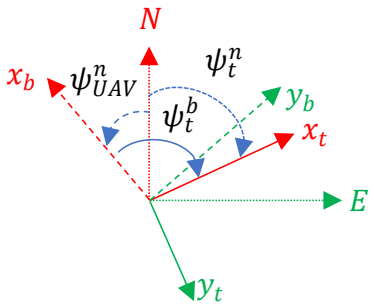


Fig. 4.3. Helipad azimuth set formulation.

4.3.1.2. Altitude Setpoint Strategy

In order to change the default controller descent behavior, it is possible to use the behavior of the system in the transient state, i.e., before reaching the maximum velocity of stationary descent. Thus, if the new desired height is reached without having to reach the maximum descent speed, the behavior will be smooth, and if the destination point is far enough away, the controller saturates and descends with maximum constant speed behavior, without exceeding the internal controller parameters.

To define step points that allow a linear descent at constant speed α in an iterative loop, the new step point will correspond to the current height minus a certain parameter α.

$$h_{set} = h_{UAV} - \alpha \tag{4.17}$$

Considering this process is iterative (discrete) with a sample time of Δt, the previous equation can be expressed as follows:

$$h_{t+1} = h_t - \alpha \Delta t \tag{4.18}$$

where t subscript means instant time. Solving the α term, it is verified that alpha corresponds to a speed term.

$$\frac{(h_{k+1} - h_k)}{\Delta t} = \alpha = \frac{\Delta h}{\Delta t} = cte. \tag{4.19}$$

In our case, the aim is to design the $h_{set}(h_{UAV})$ function such that the aircraft approaches with a smooth behavior to h_0 and at that point lands automatically with internal autopilot.

To perform this, we propose (4.20).

$$h_{set} = h_{UAV}(1 - \beta_1 e^{\frac{-1}{h_t^b - \beta_0 h_0}}) \tag{4.20}$$

where β_1 is the weight of the exponential function and $\beta_0 < 1$, which allows slightly shifting the value of h_0 and to be able to switch to automatic landing mode. The $\beta_1 = 0.1$ value is set heuristically, while $\beta_0 = 0.5$ is set to shift 50% less than the switching height h_0. The system must consider the relative flight altitude h_{UAV} and the height of the UAV relative to the landing pad h_t^b.

Fig. 4.4 shows an approximate representation of the altitude-set function behavior (4.20) vs. constant decreasing (4.17).

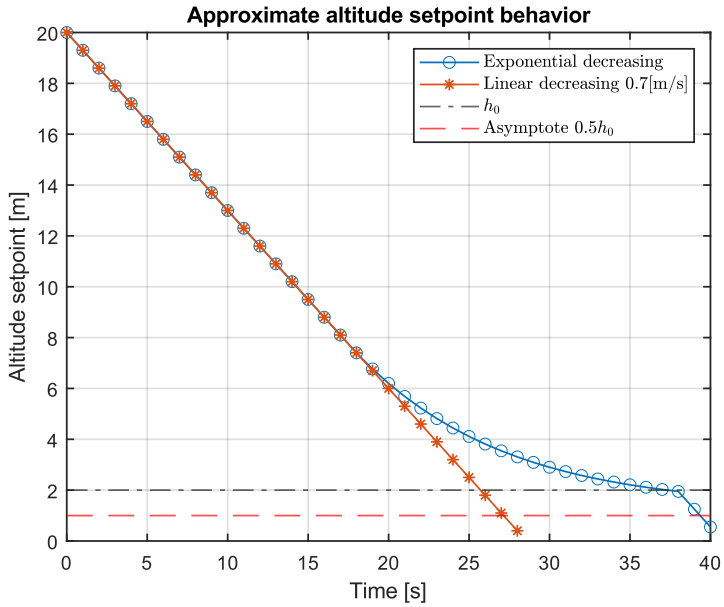


Fig. 4.4. Approximate altitude setpoint evolution.

Fig. 4.4 shows the approximate descent behavior of our proposal versus a constant speed descent. In the final phase of the approximation, the descent becomes smoother than in the linear behavior. The Fig. 4.4 behavior should be taken as an illustrative example of the desired behavior, not as a realistic simulation. The final behavior can be seen in experimentation.

4.3.1.3. Filter

The states to be filtered are the global position of the helipad $p_t^g = (\lambda_t, \varphi_t)$ and its orientation to north or azimuth ψ_t^n. All these variables are static, since the landing pad is static; therefore, the filter model does not need to provide information for each new measurement, rather we need to know their stationary statistical values. For this we propose to generate a data buffer with memory. The size of the buffer defines the size of the filtering window L. The initialization saves L new measurements and then finds the mean or median of the buffered data. Finally, the buffer is reset.

To propagate the information over time, the sliding window does not overlap with previous values, but the value filtered at the previous instant is included as the first measurement in the clean buffer.

As for the filter memory, if the new values change substantially it will vanish in the long term, since the weight given to the past values is $\frac{1}{L}$ versus $\frac{L-1}{L}$ for each new data, so the window size can be critical for cases where the target is moving. Finally, the size of the buffer/window L is linked to time thanks to the 1 Hz system sampling time to provide new measurements.

4.3.2. Helipad Global Position Estimation

The helipad global position estimation system is responsible for integrating the vision system, the heliport context information, the gimbal, and the UAV navigation states, to provide the landing strategy with the helipad global position. In addition, this includes a spatial correction system for the NED frame, which is the objective of study of this work.

Figure 5 shows the diagram of the estimation system which is formulated in (2). The system works as follows:

- Aircraft global position p_{UAV}^{g} and attitude $(\theta, \phi, \psi)_{UAV}^{b}$ is requested by the PX4 flight controller via MAVLink protocol [24] supported by the MAVSDK API [25].
- The gimbal position $p_{g}^{b}(x, y, z)$ and attitude $(\theta, \phi, \psi)_{g}^{b}$ is requested by the AirSim simulation environment via UDP protocol described in the "4.4.1. Test Environment" section. This information composes the ${}^{b}T_{z}$ transform.
- The vision system receives I image of $W \times H$ size and 3 RGB channels. The image is received via UDP protocol from the simulation system. In addition, the vision system has as input the context information from the helipad, the library (Lib) of the marker, the marker's identification number ($Id \in Lib$), and the real marker's size (MS) in meters. The library is characterized by the number of horizontal and vertical bits (squares) that form the geometry of the marker and the number of elements that make up the library. The vision system output provides a Boolean variable $a \in \mathcal{B}$, that indicates if the landing pad has been detected or not. In addition, it provides the position of the landing pad to the camera p_{t}^{c} and the attitude $(\theta, \phi, \psi)_{t}^{c}$.
- The camera pose estimation (p_{c}^{t} and ${}^{t}_{.}T_{c}$) is gated by the PnP method integrated in the OpenCV Aruco library [47] from a previously pre-calibrated camera (4.4.1. Test Environment).
- The aircraft, gimbal, and landing pad position are combined in the set of ${}^{n}T_{b}$, ${}^{b}T_{z}$, and ${}^{z}T_{c}$ transformations to obtain the positioning p_{t}^{n} and attitude $(\theta, \phi, \psi)_{t}^{n}$ of the landing pad in NED frame.

- The correction module provides the ${p'}_t^n$ positioning and attitude ${(\theta, \phi, \psi)'}_t^n$, tuned in NED coordinates.
- Finally, the target position in NED frame ${p'}_t^n$, together with the drone global position p_{UAV}^g and ellipsoid WGS84 approximation, are used to obtain the helipad global position in (11).

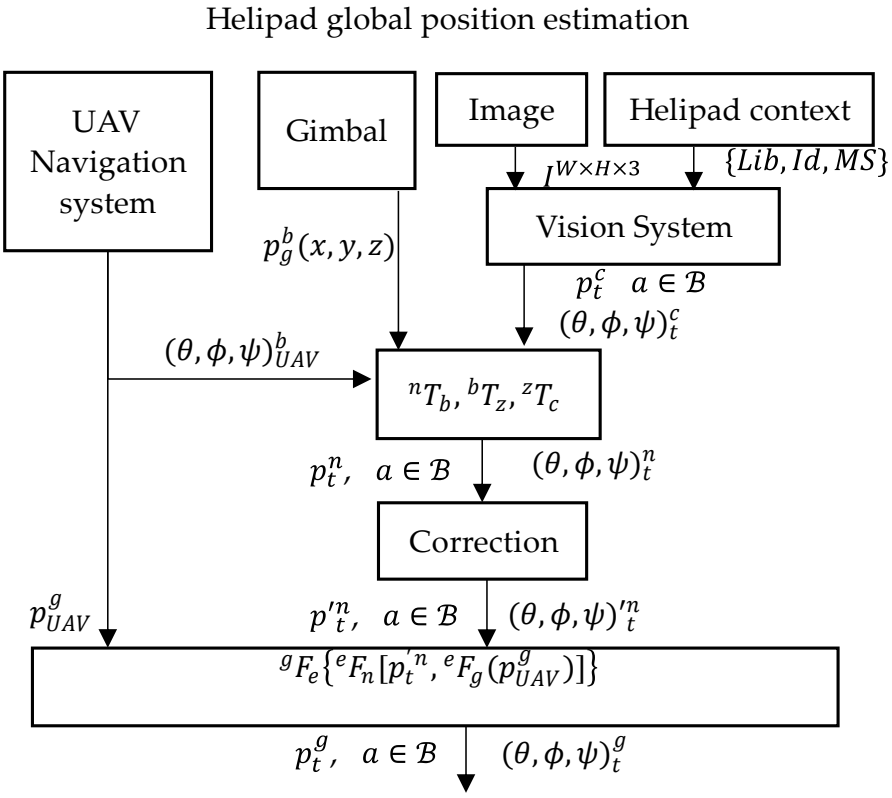


Fig. 4.5. Helipad global position estimation system.

4.4. Landing System Analysis

The aim of this section is to evaluate the proposed estimation system and to identify the necessary corrections to be incorporated in the "correction" module of Figure 4.5. To achieve this, first the test environment and the necessary parameters are detailed in the subsection Test environment. Next, the system estimation error is modeled to provide the landing system a correction module. The quality of the correction is evaluated using the root mean square error (RMSE) together with the variation in the data distribution in terms of data distribution structure, mean, and standard deviation.

Finally, a full landing system and classical linear decreasing descent are compared and evaluated with four quality metrics, which quantify the trajectory length, the time to land, and the accuracy of landing on the helipad.

4.4.1. Test Environment

In this work, we use a hyper-realistic test environment based on Software in The Loop (SITL). SITL systems are simulation architectures where virtual world environments interact to simulate object, vehicle, and sensor together with external systems such as a flight controller or ground station, among others. These environments are powerful testing tools for earlier phases of system integration, as they allow realistic results to be obtained without potentially dangerous and expensive risks.

In our case, AirSim [48] is used as world environment and PX4 flight controller configured as a quadcopter. The simulated physical model corresponds to the Iris quadcopter and the set of sensors, and their specifications are detailed in Table 4.1. The models of the simulated sensors can be found detailed in [34].

Table 4.1. Sensor parameters simulated in AirSim.

Sensor	Parameters
Barometer IMU GPS Magnetometer Distance	Default AirSim settings [49]
Gimbal-Camera	Resolution $W \times H$: 640 × 480 Field of view (FOV): 95 Depth of field focal distance: 100 Depth of field focal region: 100 Depth of field F-Stops: 2.8 Target gamma: 1.5 $p_c^b \equiv p_g^b = [0,0,0.1]$ $(\theta, \phi, \psi)_g^b = (0, -\frac{\pi}{2}, 0)$

Fig. 4.6 shows an SITL communication diagram between the main system modules in SITL. The GCS module refers to the ground control station, in our case QGround control [50]. GCS is used to help to download the .log files generated in the test missions.

The vision-based estimation system requires knowledge of the internal camera parameters $\{A, k_i\}$. These parameters are obtained by standard calibration [39] using a chess pattern with nine rows, six columns and 20 cm sides of the squares. This pattern is integrated into the AirSim environment as a texture over a rectangular prism with $1.8 \times 1.2 \times 1.8\ [m]$ sides.

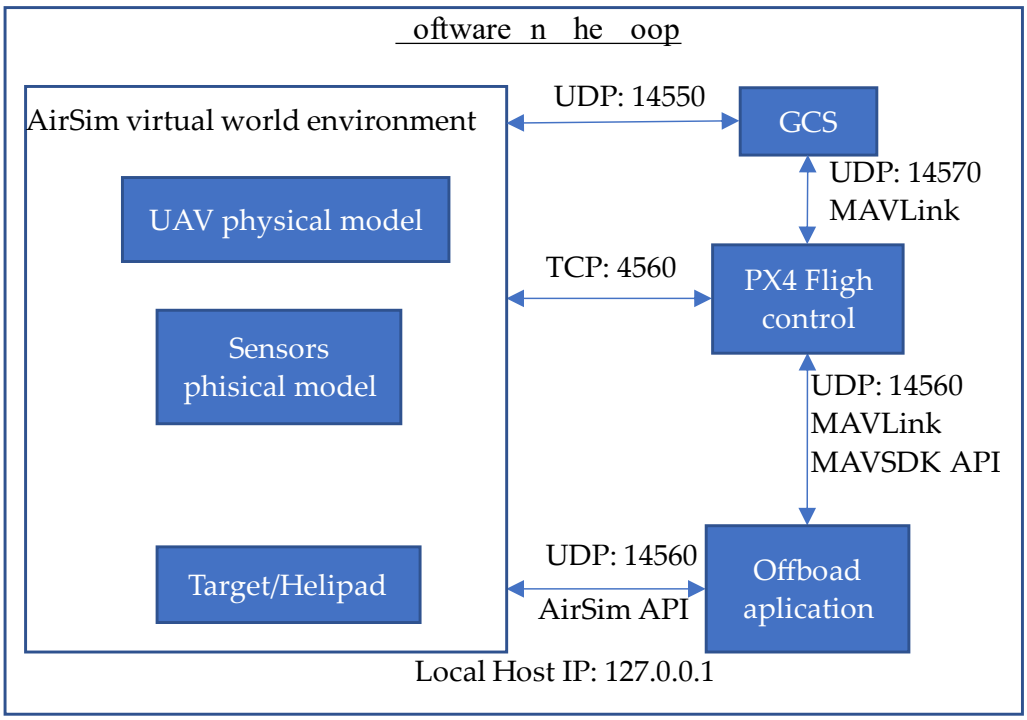
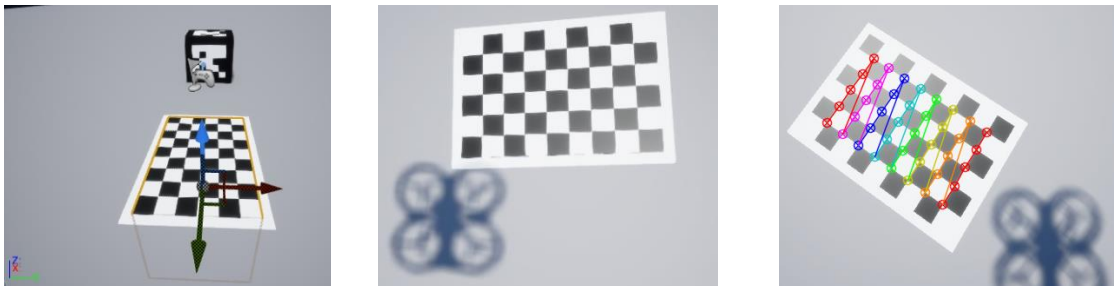


Fig. 4.6. SITL Communication and protocol diagram.

To capture images, we implemented a system that automatically captures images while performing a spiral upward flight over the reference pattern. This allows obtaining a large set of images with the pattern from different positions.



(a) (b) (c)

Fig. 4.7. Simulation environment in the calibration process: (**a**) Image of the calibration pattern in the AirSim reference frame; (**b**) Random image of the image registration process; (**c**) Example of reprojection error.

Fig. 4.7shows the image of the calibration pattern in the AirSim reference frame, random image of the image registration process used for calibration, and a sample of the reprojection error. Finally, the internal camera parameters are shown in Table 4.2. The context information used for the experimentation is: $Lib = 5 \times 5 \times 1000, Id = 68$, and $MS = 1\ [m]$.

The landing system was developed in Python 3.6 with the AirSim [34] and MAVSDK [25] APIs. The experiments and the SITL environment were developed on a Windows Server 2019,

64 bits, hosted in AMD Ryzen 9 3900X 12-Core Processor CPU, 3.79 GHz with 64 GB RAM and 2x1TB SSD + 2xHDD 1.5TB of internal memory, graphic card Nvidia GeForce RTX 2060.

Table 4.2. Internal camera parameters.

Parameter	Value
f_x	293.35 [mm]
f_y	293.31 [mm]
c_x	319.64 [px]
c_y	239.64 [px]
Distortion coefficients k_i	$\{16.44, 35.89, 6.35, -6.35, 100.7\} \times 10^{-4}$

4.4.2. NED Error Modeling

To evaluate the estimation error, we propose to analyze the estimation data provided by the vision system over twenty static flights located at seventeen different positions at the same relative altitude above the ground, 10 meters.

The selected positions correspond to five different headings centered, 45° {northeast, southeast, southwest, and northwest} and four different distances $\{2,3,4,5\}\sqrt{2}$ to the takeoff origin where the helipad is located. In each position is recorded a total of 1000 p_{UAV}^{t} samples.

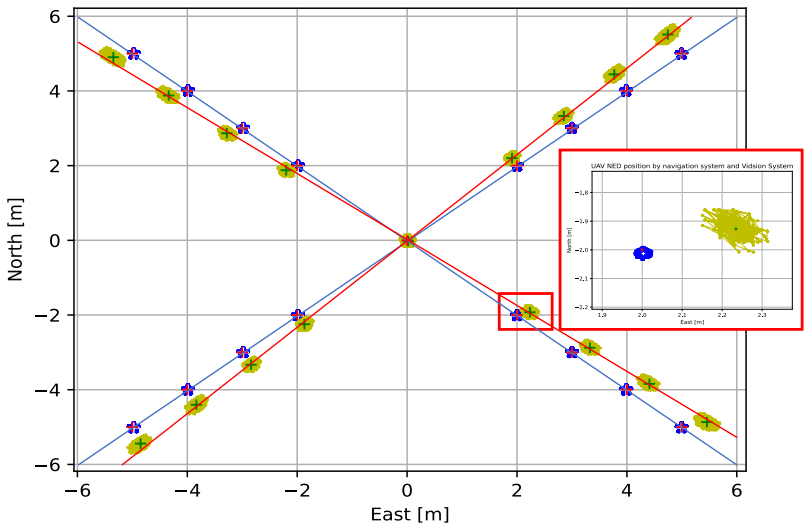


Fig. 4.8. Data registration in twenty different flights. Yellow, UAV positions with vision system. Blue, UAV positions with navigation system. Blue and red line, linear approaches between navigation and vision system data centers, respectively. Red box, zoom in [−2,2,10] NED position.

We consider the aircraft control system is asymptotically stable so that in steady state its position converges to the reference one. Thus, we consider as ground truth the reference positions for the steady flight.

The error position for each of the components is given by (4.21).

$$e(x)_i^f = x_i^f - x_{GT_i}^f \tag{4.21}$$

where $e(x)_i^f$ means the position error of the component x of the system i in f referent frame. GT subscript means the ground truth in f reference frame.

Looking at Fig. 4.8, while the blue line maintains the desired directions of 45° (NE, SE, SO, NO), the centers of the positions recorded by the vision system, the red line, are decoupled, showing a constant angular deviation of the positions. Looking at the errors (Fig. 4.9), the error distribution increases with increasing distance from the north-east plane origin (0,0). This means the error position depends on the position in the NE plane.

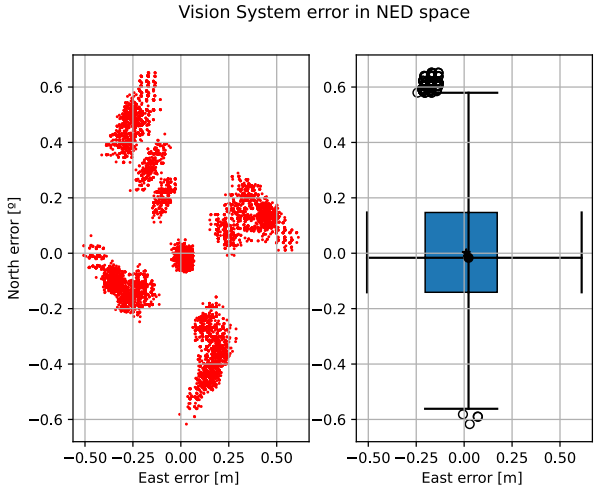
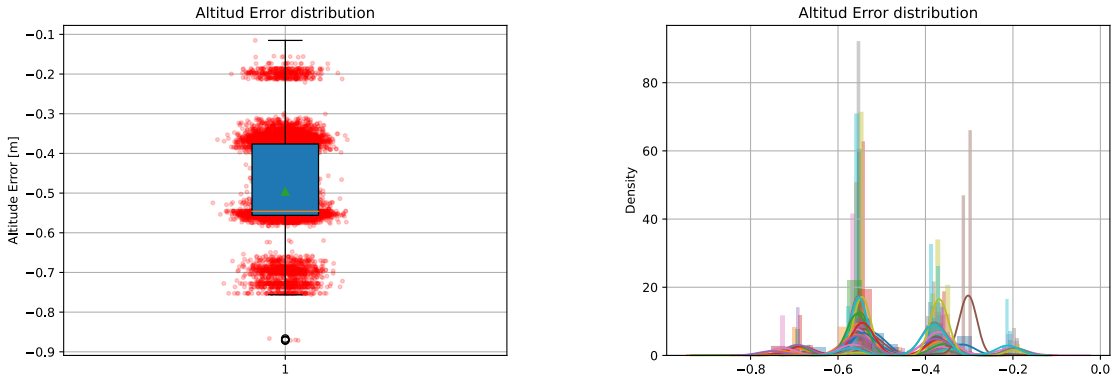


Fig. 4.9. Vision system error in north-east plane coordinate. Left, scatter distribution; Right, north-east boxplot with 1.5 whiskers.

Regarding altitude error, Fig. 4.10 (a) and (b) show for each twenty register positions a distribution with four “modes”. In Fig. 4.10 (a) these modes show as four scatter clusters and in Fig. 4.10 (b) as four peaks in each twenty distributions. In addition, the mean and median of the total error distribution, Fig. 4.10, are displaced from the origin, showing a bias in altitude.

The different colors in Fig. 4.10 (b) show each of the twenty records, all of them showing four modes and centered on the same error terms. In this work, we focus on bias correction of mean and median; however, modeling the error in altitude is outside the scope of this paper.



(a) (b)

Fig. 4.10. Altitude error: (a) Scatter and boxplot. Green triangle: mean, orange line: median; (b) Altitude error distribution for twenty different register positions.

The altitude error distribution may be a consequence of the internal discretizing of the simulator in the image render, so that the vision system, when segmenting the ROI of the helipad, extracts its contour with a size variation. This would be explainable according to the pinhole model of (4.4) and PnP Formulation (4.7), since its scale factor is constant, but the size of the ROI and corner positions would change.

4.4.2.1. Polar Space Error Analysis

The visual results in Fig. 4.8 and Fig. 4.9 show an apparent angular and radial bias of the helipad global position estimation system. We change the cartesian space to the cylindrical space defined by (4.22), where the terms E, N, D are the coordinates in the NED referent frame.

$$\begin{aligned} r_0 &= \sqrt{E^2 + N^2} \\ \theta_0 &= atan\left(\frac{N}{E}\right) \\ D_0 &= D \end{aligned} \tag{4.22}$$

When plotting data in the new space in Fig. 4.11, it can be seen how the data set is apparently clustered around a constant bias in the angle and radial error.

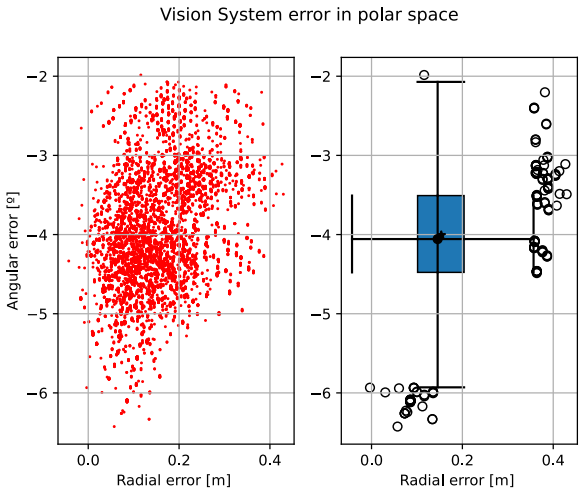


Fig. 4.11. Vision system error in polar space. Left, scatter distribution; Right, 2D boxplot with 1.5 whis.

However, when showing the distance error (radial) behavior versus distance, Fig. 4.12 shows a high linear correlation between distance and radial error. The angular error is also tested for linear dependence on distance, but the correlation does not exceed 35% of variance score r^2 in (4.23), so it has been discarded.

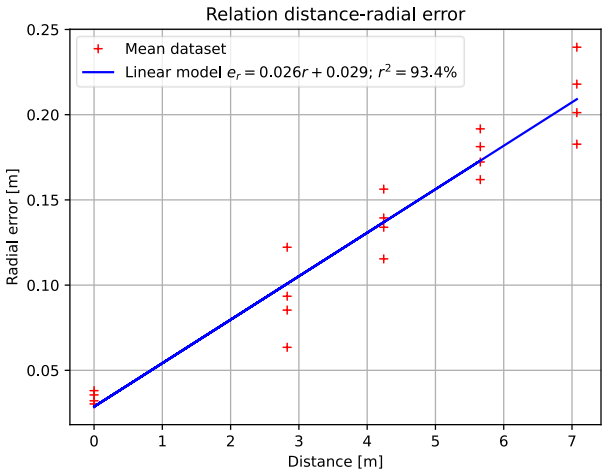


Fig. 4.12. Relationship between distance and radial error. Red + symbol, centroid of each position. Blue line, linear model fitted by least squares.

$$r^2 = 1 - \frac{Var(x - \hat{x})}{Var(x)} \tag{4.23}$$

Where $Var(x - \hat{x})$ indicates the variance of the error between $\hat{x}$ model estimation and x data. Fig. 4.12 shows with "+" the radial error centers of each of the measurement positions and the linear model fitted (blue line) by least squares to these points. This model has a slope $\alpha_r = 2.4923 * 10^{-2}$ and independent term $\beta_r = 2.778497 \cdot 10^{-2}\ [m]$. The linear model obtains around 94% of the variance score in (4.23).

Given the results in Table 4.3 and Fig. 4.12, the error bias in polar space can be tuned by the model of (4.27), where the apostrophe over coordinates means corrected coordinate, subscript zero start value, and β_i the bias of coordinate i.

$$\begin{aligned} r' &= r_0(1-\alpha_r) - \beta_r \\ \theta' &= \theta_0 - \beta_\theta \\ D' &= D_0 - \beta_D \end{aligned} \tag{4.24}$$

Table 4.3. Stationary error.

	$\boldsymbol{\beta_\theta}$ [°]	Distances [m]	Altitude $\boldsymbol{\beta_D}$ [m]
Mean	−4.007078	0.153636	0.488412
Median	−4.056872	0.145540	0.545400
Var	0.512071	0.005182	0.014129

4.4.2.2. Error Correction in NED Space

Given N, E, and D coordinates of a point p_i^n in the NED frame and knowing the correction in a cylindrical space in (4.27), the objective is to return to the NED space. For this purpose, we apply the transformation (4.28).

$$\begin{aligned} \widehat{N} &= \hat{r}.cos(\theta) \\ \widehat{E} &= \hat{r}.sin(\theta) \\ \widehat{D} &= D_0 - \beta_D \end{aligned} \tag{4.25}$$

where its terms $\hat{r}, \theta$ are taken as shown in (4.26):

$$\begin{aligned} \hat{r} &= \left|v_{\beta_\theta}\right| - r' \\ r' &= \alpha_r r_0 + \beta_r \\ \theta &= atan\left(\frac{v_N}{v_E}\right) \\ v(\beta_\theta) &= \begin{cases} v_E = E\cdot\cos(\beta_\theta) - N\cdot\sin(\beta_\theta) \\ v_N = E\cdot\sin(\beta_\theta) + N\cdot\cos(\beta_\theta) \end{cases} \end{aligned} \tag{4.26}$$

where the hat over N, E, and D in (4.29) means the NED coordinate with cylindrical corrections. $\beta_{\{\theta,r,D\}}$ are the biases of the radial, angular, and altitude terms, respectively. v_{β_θ} components and θ mean new position and new angular position after β_θ rotation correction.

Fig. 4.13 shows the error distributions of the raw position estimation and error distribution after applying the correction (4.26) with the parameters of Table 4.1.

For each drawing in Fig. 4.13, the distributions of the real data are shown in blue and with a blue line their error distribution function [51]. The orange line shows a Gaussian distribution

equivalent to the real data in (4.27). For the NED coordinate, three different figures are represented: raw data Fig. 4.13 (a–c), data after cylindrical correction using the mean of the raw data Fig. 4.13 (d–f), and data after cylindrical correction according to the median of the raw data Fig. 4.13 (g–i). In addition, the mean value and the standard deviation of each case represented by μ and σ, respectively, are indicated on each graph in their legend.

$$\mathcal{N}(\mu, \sigma, x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.27}$$

Fig. 4.13. shows how correction (4.25) modifies the structure of the error distribution for the cases of N and E coordinates, when comparing the blue with the orange lines.

The effect on the Gaussian approximations in mean and standard deviation represented in Fig. 4.13 is quantified in Table 4.4.

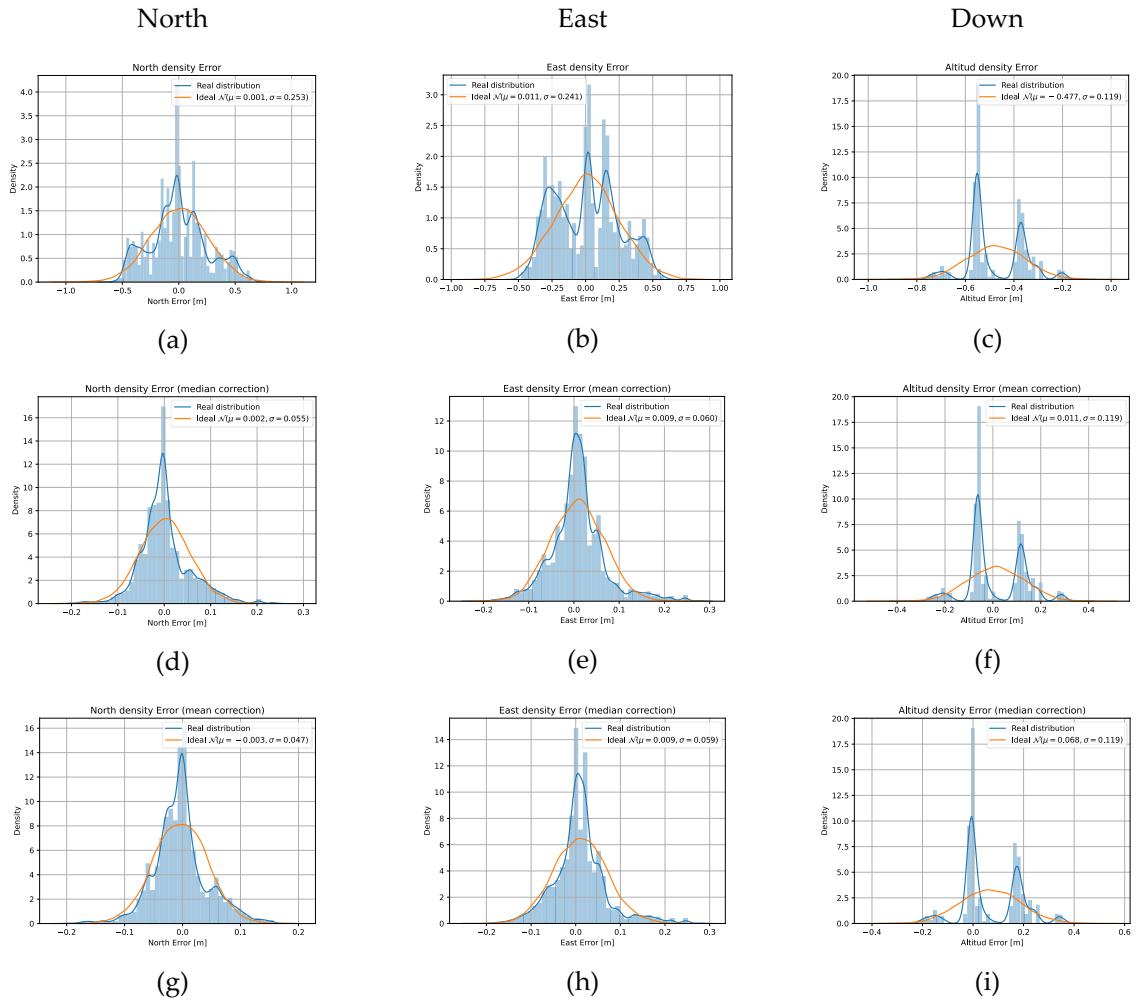


Fig. 4.13. NED coordinates density error distribution. North-East-Down data without correction (a, b, and g). Data with mean cylindrical correction (c, d, and h). Data with median cylindrical correction (e, f, and i).

Table 4.4 shows the effect of mean and standard deviation on the data when applying the correction (4.25) using the mean and median bias value indicated in Table 4.3.

Table 4.4. Gaussian density distribution approximation.

	$\mathcal{N}(\boldsymbol{\mu_i}, \boldsymbol{\sigma_i})$	Raw	Mean	Median
North	μ_N [$\times 10^{-4}$]	7.930	20.200	17.120
	σ_N [$\times 10^{-2}$]	25.334	5.293	5.454
East	μ_E [$\times 10^{-2}$]	1.087	0.851	0.882
	σ_E [$\times 10^{-1}$]	2.407	0.595	0.594
Altitude	μ_D [$\times 10^{-1}$]	−4.774	0.110	0.680
	σ_D [$\times 10^{-1}$]	1.191	0.119	0.119

The standard deviation rows of Table 4.4 decrease one order of magnitude in all coordinates when applying the correction (4.25). The same effect can be seen in Table 4.5 when the RMSE (4.28) of each NED coordinate is calculated. This metric can be considered as an indicator of accuracy.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{x}_i^n - x_{GT_i}^n\right)^2} \tag{4.28}$$

Table 4.5. RMSE value.

Coor./RMSE	Raw	Mean	Median
North [$\times \mathbf{10^{-2}}$]	6.418	0.280	0.298
East [$\times \mathbf{10^{-2}}$]	5.804	0.361	0.361
Altitude [$\times \mathbf{10^{-2}}$]	2.421	0.143	0.188

4.4.3. Landing Evaluation

To evaluate the landing system, we propose to test twenty landing missions from the same position at (10, 10, −20) NED meters to the helipad. The twenty flights are divided into four groups corresponding to using the raw landing system without correction ($without$) (4.2), applying the bias correction (4.25)($Bias$), with bias correction and a mean filter (Section 4.3.1.3) with a sliding time window ($Mean\&Bias$) and a median filter together with the bias correction ($Median\&Bias$). For each mission, we use as quality metrics the landing trajectory distance (4.29), time to land (4.30), and landing accuracy. In addition, the results are compared with classical linear descent setpoints with $\alpha = 0.7\ [m/s]$.

$$Dist = \sum_{k=1}^{K} \|p_{k+1}^n - p_k^n\| \tag{4.29}$$

$$Time = t_{start} - t_{land} \tag{4.30}$$

where the k term means temporal step starting in t_{start} instant and ending in t_{land} moment. p_k^n represents the global position at k instant in the NED reference frame.

For the flights' analysis, we use the information obtained from the logs recorded by the flight controller in each flight and unloaded with the ground control station (GCS). In particular, we focus on the global positioning of the UAV. This positioning is given by the EKF2 fusion system [52] integrated in PX4 and with the specific sensor parameters indicated in the SITL [33] section "Test environment". To ensure that we evaluate exclusively the landing phase of the UAV, we study the trajectories from the instant t_{start} where the height gradient is detected to be negative, and the altitude is less than 99.8% of the altitude desired. The final instant t_{land} is obtained when the helipad is reached by the same method as t_{start}.

Finally, third quality metric, landing accuracy, is obtained with the RMSE of the last ten position samples of the NED coordinates (without altitude). In this way, we ensure that we are on the ground with the same value plus a precision error. For this, we take as ground truth the control position of the marker.

Fig. 4.14 shows the behavior of the aircraft when activating the landing system with the four modes corresponding to the landing system without correction Fig. 4.14 (a), landing with bias correction Fig. 4.14 (b), landing with bias correction and mean filter Fig. 4.14 (c,d), and landing with bias correction and median filter. The five trajectories in each of the figures show a different flight, using the corresponding landing mode in each case. The total number of flights is five for each landing mode, i.e., twenty flights.

Fig. 4.15 illustrates the temporal behavior of each analysis group, showing the three global position components: latitude, longitude, and relative altitude. In addition, our proposal is comparing with linear descent, paying special attention to altitude evolution Fig. 4.15 (e and f). In this case, estimated altitude and setpoints are shown for the exponential proposal and linear descending.

It can be seen in all cases in Fig. 4.15 how the position of the landing pad, defined by a blue dashed horizontal line, is obtained in the stationary state. The effect of the error in precision is shown with an oscillating behavior (blue dotted line). However, when bias (4.26) and filtering (Section 4.3.1.3) corrections are applied, the effect is damped.

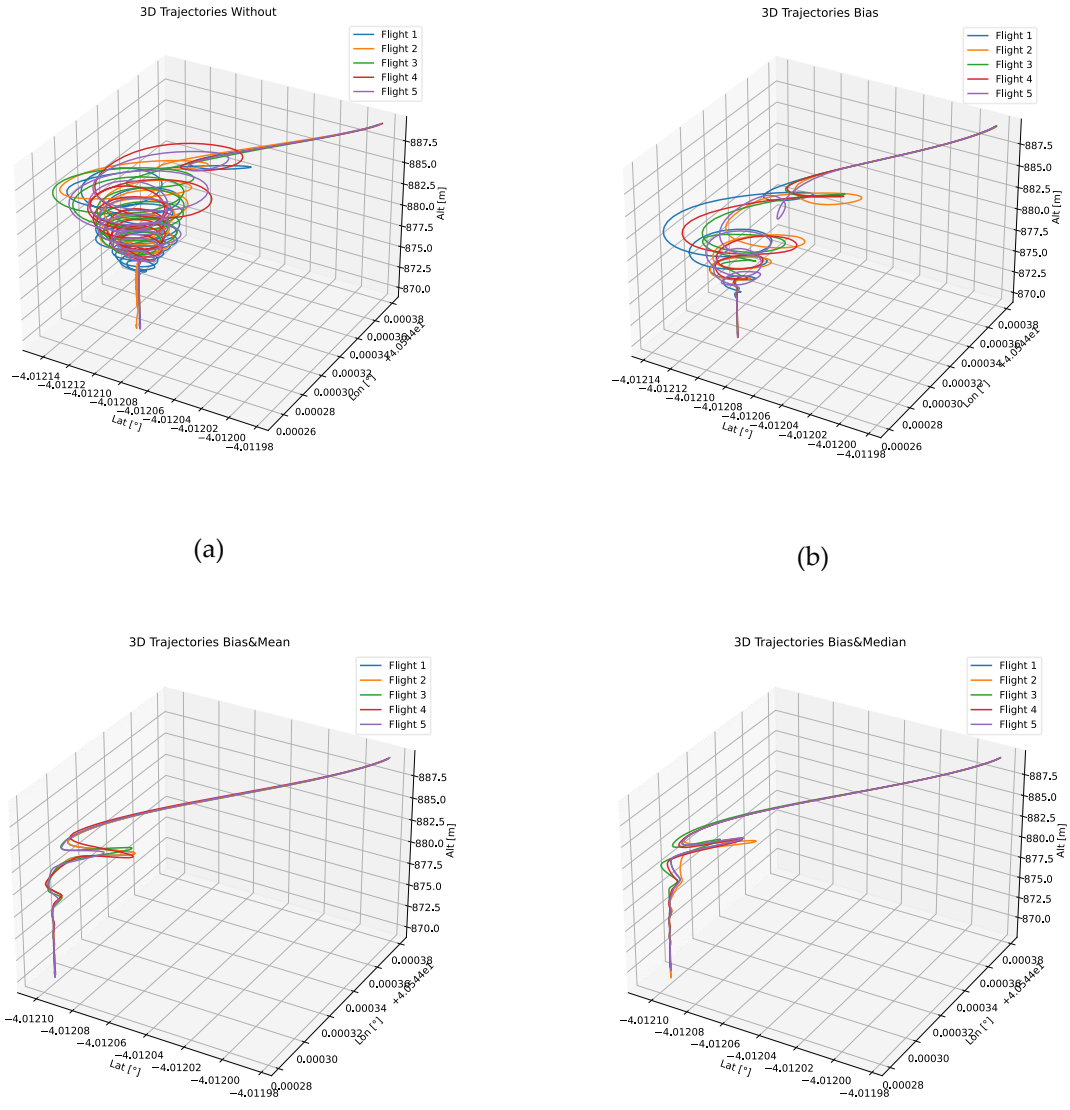


Fig. 4.14. Five-flight 3D graphics for each of the four groups: (a) without correction; (b) bias correction; (c) bias and mean filter; (d) bias correction and median filter.

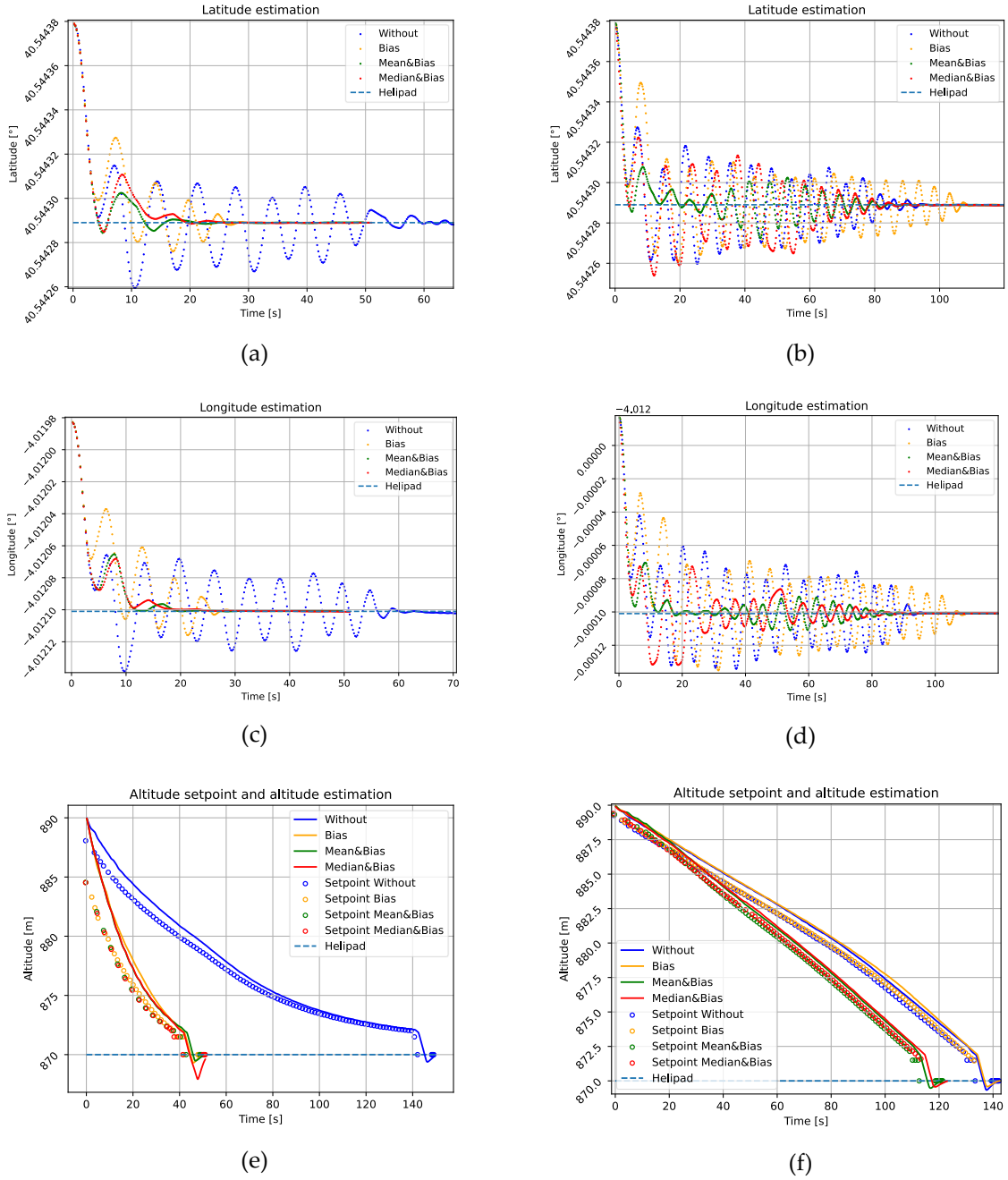


Fig. 4.15. Time evolution of the latitude, longitude, and altitude of four flights with different correction modes in the landing phase: (a, b) latitude, (c, d) longitude, and (e, f) altitude. First column (a, c, e) exponential decrease, second column (b, d, f) linear decrease.

The linear decrease approach converges in latitude and longitude Fig. 4.15 (b,d), but the system finds it difficult to dampen the spin effect. In the case of an exponential decrease, the inter-waypoint spin effect is considerably less than linear Fig. 4.15 (a–d). In both cases, the linear and exponential altitude decreasing approaches (Fig. 4.15) show when error correction is applied, and filtering estimation inter-waypoint noise spin effect is smoothed.

Fig. 4.15 (e) and (f) show a small break at the end of the trajectory that exceeds the reference and picks it up again. To identify the instant to obtain the landing surface, we keep the first term that satisfies the gradient and proximity conditions explained above.

This effect may be a consequence of a decoupling of the fusion system in the estimation of the height, for example for giving more weight to the estimation than to the measurements in the EKF2 filter. Therefore, when identifying the instant of reaching the pad, we are left with the first term that meets the conditions of gradient and proximity explained above.

The total of twenty flights with exponential decreasing approach and the other twenty flights with linear decreasing approaches are summarized in Table 4.6 and Table 4.7. These tables are built with the mean and median values of all flights, so the quality metrics means the mean and median values of all flights.

Table 4.6. Mean value of quality metrics.

Exp.|Linear	**Distance [m]**	**Time [s]**	**RMSE landing** $\times \mathbf{10^{-7}}$
Without	131.25|205.88	143.77|136.37	1.650|1.74
Bias	57.07|192.15	44.30|131.62	1.011|2.38
Mean&Bias	37.44|54.73	44.64|116.09	0.875|1.17
Median&Bias	37.91|67.25	43.62|131.07	1.119|2.68

Table 4.7. Median value of quality metrics.

Exp.|Linear	**Distance [m]**	**Time [s]**	**RMSE landing** $\times \mathbf{10^{-7}}$
Without	130.14|205.95	144.93|136.37	2.885|2.12
Bias	57.25|195.46	44.54|131.81	1.193|2.43
Mean&Bias	34.89|60.68	44.95|116.09	1.037|1.18
Median&Bias	36.21|85.58	44.34|132.73	1.042|2.70

Table 4.6 and Table 4.7 show the results of the exponential and linear decreasing approaches grouped for easy comparison. The previous tables show for all cases that the exponential descent proposal improves the results of the linear descent, emphasizing that in the best-case scenario of the linear approach ($Mena\&Bias$), the trajectory distance is reduced by 32%, the time to land by 61%, and the RMSE of the precision landing by 12%.

In both tables, the mode that provides the minimum mean landing trajectory distance is the bias correction together with the median filter. The minimum mean time to land is provided by the bias correction together with the median filter and the minimum mean RMSE is again provided by the bias correction together with the mean filter. Finally, the similarity between the mean and median values in Table 4.6 and Table 4.7 are a good indicator of normal distribution.

4.5. Conclusions

Through the study of the global position estimation system error of Section 4.4, an angular and radial bias was identified. In addition, it was shown how the error distribution increases its degree of dispersion with the distance to the origin. This position error was initially modeled in a cylindrical space in (4.24) and transferred to the NED reference space under the transformation (4.25) and (4.26). The corrections over cylindrical space produced a structural transformation of the position error distribution, approximating its distributions to the Gaussian error.

On the other hand, it was verified how using a system aimed at smoothing trajectories between waypoints can produce a spin effect if the new waypoints are updated with a frequency such that the UAV cannot obtain the previous target and these new waypoints correspond to the same position, but with high uncertainty. Therefore, the path planning system with path smoothing between waypoints can work as an error amplifier performing a circular trajectory.

Finally, we conclude that the combination of an exponential altitude decrease, together with the correction of systematic estimation error and a sliding time window filtering, improves all three-quality metrics proposed and reduces the effect of the inter-waypoint noise spin effect. These results facilitate the development of new applications that require a lightweight but robust precision landing strategy.

Author Contributions: Conceptualization, J.P.L.C., J.G.H. and J.M.M.L.; Formal Analysis, J.P.L.C., J.G.H. and J.M.M.L.; Funding Acquisition, J.G.H. and J.M.M.L.; Investigation, J.P.L.C.; Methodology, J.P.L.C.; Project Administration, J.G.H. and J.M.M.L.; Resources, J.P.L.C., J.G.H. and J.M.M.L.; Software, J.P.L.C.; Supervision, J.G.H. and J.M.M.L.; Validation, J.P.L.C., J.G.H. and J.M.M.L.; Visualization, J.P.L.C.; Writing—Original Draft Preparation, J.P.L.C.; Writing—Review and Editing, J.P.L.C., J.G.H. and J.M.M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by public research projects of Spanish Ministry of Science and Innovation, references PID2020-118249RB-C22 and PDC2021-121567-C22 - AEI/10.13039/501100011033, and by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors, reference EPUC3M17.

Acknowledgments: Thanks to Cristina Muntañola for her support to convert Figures 1 and 2 to .svg image format.

Conflicts of Interest: The authors declare no conflict of interest.

4.6. References

[1] Gautam, A.; Sujit, P.; Saripalli, S. A survey of autonomous landing techniques for UAVs. In Proceedings of the 2014 International Conference on Unmanned Aircraft Systems (ICUAS), Orlando, FL, USA, 27–30 May 2014; pp. 1210–1218.

[2] Olivares-Mendez, M.A.; Mondragon, I.; Campoy, P.; Martinez, C. Fuzzy controller for UAV-landing task using 3D-position visual estimation. In Proceedings of the 2010 IEEE World International Conference on Fuzzy Systems, WCCI 2010, Barcelona, Spain, 18–23 July, 2010.

[3] Shi, G.; Shi, X.; O'Connell, M.; Yu, R.; Azizzadenesheli, K.; Anandkumar, A.; Yue, Y.; Chung, S.-J. Neural lander: Stable drone landing control using learned dynamics. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 9784–9790.

[4] Jeong, H.-J.; Choi, J.D.; Ha, Y.-G. Vision Based Displacement Detection for Stabilized UAV Control on Cloud Server. *Mob. Inf. Syst.* 2016, *2016*, 1–11.

[5] Chen, Y.; Zhou, Y.; Lv, Q.; Deveerasetty, K.K. A review of V-SLAM. In Proceedings of the 2018 IEEE International Conference on Information and Automation, Wuyi Mountain, China, 11–13 August 2018; pp. 603–608.

[6] Kang, Y.; Park, B.-J.; Cho, A.; Yoo, C.-S.; Kim, Y.; Choi, S.; Koo, S.-O.; Oh, S. A Precision Landing Test on Motion Platform and Shipboard of a Tilt-Rotor UAV Based on RTK-GNSS. *Int. J. Aeronaut. Space Sci.* 2018, *19*, 994–1005.

[7] Janousek, J.; Marcon, P. Precision landing options in unmanned aerial vehicles. In Proceedings of the *2*018 International Interdisciplinary PhD Workshop (IIPhDW 2018), Swinoujscie, Poland, 9–12 May 2018; pp. 58–60.

[8] Aishwarya, C. The Instrument Landing System (ILS)—A Review. *Int. J. Progress. Res. Sci. Eng.* 2022, *3*, 1–6.

[9] Alam, S.; Oluoch, J. A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (UAVs). *Expert Syst. Appl.* 2021, *179*, 115091.

[10] DJI. Página Oficial. Available online: https://www.dji.com/es (accessed on 19 April 2022).

[11] Yoakum, C.; Cerreta, J. A Review of DJI's Mavic Pro Precision Landing Accuracy. *Int. J. Aviat. Aeronaut. Aerosp.* 2020, *7*, 1–19.

[12] Mittal, M.; Mohan, R.; Burgard, W.; Valada, A. Vision-Based Autonomous UAV Navigation and Landing for Urban Search and Rescue. *arXiv* 2022, arXiv:1906.01304.

[13] Wubben, J.; Fabra, F.; Calafate, C.T.; Krzeszowski, T.; Marquez-Barja, J.M.; Cano, J.-C.; Manzoni, P. Accurate Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition. *Electronics* 2019, *8*, 1532.

[14] Zheng, Y.; Xie, H. Review on Neural Network Identification for Maneuvering UAVs. In Proceedings of the International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC 2018), Xi'an, China, 15–17 August 2018; pp. 339–346.

[15] Mebarki, R.; Lippiello, V.; Siciliano, B. Autonomous landing of rotary-wing aerial vehicles by image-based visual servoing in GPS-denied environments. In Proceedings of the 2015 IEEE International Symposium on Safety, Security, and Rescue Robotics, West Lafayette, IN, USA, 18–20 October 2015.

[16] Abujoub, S.; McPhee, J.; Westin, C.; Irani, R.A. Unmanned Aerial Vehicle Landing on Maritime Vessels using Signal Prediction of the Ship Motion. In Proceedings of the OCEANS 2018 MTS/IEEE Charleston, CA, USA, 22–25 October 2018.

[17] Mondragón, I.F.; Campoy, P.; Martínez, C.; Olivares-Méndez, M.A. 3D pose estimation based on planar object tracking for UAVs control. In Proceedings of the 2021 IEEE International Conference on Robotics, Automation and Artificial Intelligence, Hong Kong, China, 21–23 April 2021; pp. 35–41.

[18] Lebedev, I.; Erashov, A.; Shabanova, A. Accurate Autonomous UAV Landing Using Vision-Based Detection of ArUco-Marker. In *International Conference on Interactive Collaborative Robotics*; Springer: Cham, Switzerland; Volume 12336, pp. 179–188.

[19] Li, S.; Xu, C.; Xie, M. A Robust O(n) Solution to the Perspective-n-Point Problem. *IEEE Trans. Pattern Anal. Mach. Intell.* 2012, *34*, 1444–1450.

[20] Garrido-Jurado, S.; Muñoz-Salinas, R.; Madrid-Cuevas, F.J.; Marín-Jiménez, M.J. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognit.* 2014, *47*, 2280–2292.

[21] Romero-Ramirez, F.J.; Munoz-Salinas, R.; Medina-Carnicer, R. Fractal Markers: A New Approach for Long-Range Marker Pose Estimation Under Occlusion. *IEEE Access* 2019, *7*, 169908–169919.

[22] Li, B.; Wang, B.; Tan, X.; Wu, J.; Wei, L. Corner location and recognition of single ArUco marker under occlusion based on YOLO algorithm. *J. Electron. Imaging* 2021, *30*, 033012. https://doi.org/10.1117/1.JEI.30.3.033012.

[23] Pixhawk. The Hardware Standard for Open-Source Autopilots. Available online: https://pixhawk.org/ (accessed one 13 March 2022).

[24] Koubaa, A.; Allouch, A.; Alajlan, M.; Javed, Y.; Belghith, A.; Khalgui, M. Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey. *IEEE Access* 2019, *7*, 87658–87680.

[25] Introduction MAVSDK Guide. Available online: https://mavsdk.mavlink.io/main/en/index.html (accessed on 13 March 2022).

[26] Open-Source Autopilot for Drones—PX4 Autopilot. Available online: https://px4.io/ (accessed on 22 February 2022).

[27] Lizarraga, M.; Curry, R.; Elkaim, G.H. Flight test results for an improved line of sight guidance law for UAVs. In Proceedings of the American Control Conference, Washington, DC, USA, 17–19 June 2013; pp. 818–823.

[28] Anderson, E.P.; Beard, R.W.; McLain, T.W. Real-time dynamic trajectory smoothing for unmanned air vehicles. *IEEE Trans. Control Syst. Technol.* 2005, *13*, 471–477.

[29] Kikutis, R.; Stankūnas, J.; Rudinskas, D.; Masiulionis, T. Adaptation of Dubins Paths for UAV Ground Obstacle Avoidance When Using a Low Cost On-Board GNSS Sensor. *Sensors* 2017, *17*, 2223.

[30] Iii, D.W.S.; Sanfelice, R.G. Autonomous Waypoint Transitioning and Loitering for Unmanned Aerial Vehicles via Hybrid Control. In Proceedings of the AIAA Guidance, Navigation, and Control Conference, San Diego, CA, USA, 4–8 January 2016.

[31] Park, S.; Deyst, J.; How, J. A New Nonlinear Guidance Logic for Trajectory Tracking. In Proceedings of the AIAA Guidance, Navigation, and Control Conference an Exhibit, Providence, RI, USA, 16–19 August 2004.

[32] Stateczny, A.; Burdziakowski, P.; Najdecka, K.; Domagalska-Stateczna, B. Accuracy of trajectory tracking based on nonlinear guidance logic for hydrographic unmanned surface vessels. *Sensors* 2020, *20*, 832.

[33] Ma, C.; Zhou, Y.; Li, Z. A New Simulation Environment Based on Airsim, ROS, and PX4 for Quadcopter Aircrafts. In Proceedings of the 6th International Conference on Control, Automation and Robotics (ICCAR 2020), Singapore, 20–23 April 2020; pp. 486–490.

[34] Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. *Springer Proc. Adv. Robot.* 2018, *5*, 621–635.

[35] Adli, S.E.; Shoaran, M.; Noorani, S.M.S. GSPnP: Simple and geometric solution for PnP problem. *Vis. Comput.* 2019, *36*, 1549–1557.

[36] PLi, P. Quantum implementation of the classical Canny edge detector. *Multimed. Tools Appl.* 2022, *81*, 11665–11694.

[37] Luo, S.; Hou, J.; Zheng, B.; Zhong, X.; Liu, P. Research on edge detection algorithm of work piece defect in machine vision detection system. In Proceedings of the 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC 2022), Chongqing China, 4–6 March 2022; pp. 1231–1235.

[38] Hartley, R.; Zisserman, A. *Multiple View Geometry in Computer Vision*, 2nd ed.; Cambridge University Press: Cambridge, UK, 2003; pp. 262–278.

[39] Datta, A.; Kim, J.S.; Kanade, T. Accurate camera calibration using iterative refinement of control points. In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision (ICCV 2009), Kyoto, Japan, 29 September–2 October *2009;* pp. 1201–1208.

[40] Home—OpenCV. Available online: https://opencv.org/ (accessed on 21 July 2021).

[41] Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library—Adrian Kaehler, Gary Bradski—Google Libros. Available online: https://books.google.es/books?hl=es&lr=&id=LPm3DQAAQBAJ&oi=fnd&pg=PP1&dq=G.+Bradski+and+A.+Kaehler,+Learning+OpenCV3:+ComputerVision+in+C%2B%2B+With+the+OpenCV+Library,+2nd+ed.+Newton,+MA,+USA:+O'Reilly+Media,+2013&ots=2wLqQga9C7&sig=nzLIWPd4uyeVkH93pJkiN7b3hbA&redir_esc=y#v=onepage&q&f=false (accessed on 22 March 2022).

[42] Patel, T.P.; Panchal, S.R.; Student, P.G. Corner Detection Techniques: An Introductory Survey. *IJEDR* 2014, *2*, 2321–9939.

[43] Collins, T.; Bartoli, A. Infinitesimal Plane-Based Pose Estimation. *Int. J. Comput. Vis.* 2014, *109*, 252–286.

[44] Valavanis, K.P.; Vachtsevanos, G.J. *Handbook of Unmanned Aerial Vehicles*; Springer: Dordrecht, Netherlands, 2015.

[45] Zhu, J. Conversion of Earth-centered Earth-fixed coordinates to geodetic coordinates. *IEEE Trans. Aerosp. Electron. Syst.* 1994, *30*, 957–961.

[46] Osen, K. Accurate Conversion of Earth-Fixed Earth-Centered Coordinates to Geodetic Coordinates. Ph.D. Thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2017.

[47] OpenCV: Detection of ArUco Markers. Available online: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (accessed on 14 March 2022).

[48] Home—AirSim. Available online: https://microsoft.github.io/AirSim/ (accessed on 22 February 2022).

[49] Settings—AirSim. Available online: https://microsoft.github.io/AirSim/settings/ (accessed on 24 March 2022).

[50] QGC—QGroundControl—Drone Control. Available online: http://qgroundcontrol.com/ (accessed on 29 March 2022).

[51] Chen, Y.-C. A tutorial on kernel density estimation and recent advances. *Biostat. Epidemiol.* 2017, *1*, 161–187. https://doi.org/10.1080/24709360.2017.1396742.

[52] García, J.; Molina, J.M.; Trincado, J. Real evaluation for designing sensor fusion in UAV platforms. *Inf. Fusion* 2020, *63*, 136–152.

Part II: Deep Learning, forecasting, filtering, and classification

Chapter 5: Artificial Neural Networks

5.1. Introduction

According to the Cambridge dictionary [1] Artificial Intelligence (AI) is defined as "*the study of how to produce machines that have some of the qualities that the human mind has, such as the ability to understand language, recognize pictures, solve problems, and learn*". It is a general concept that addresses different fields of study. Approaches that aim to replicate biological behaviors have been very successful in the last decades, some of these success stories are bio-inspired algorithms or artificial neural networks. Today the concept of artificial neural networks (ANN) is commonly used by the digitized society. However, the first proposals date back to the 1950s. At the end of the 1950s, the scientist Frank Rosenblatt, inspired by the previous work of Warren MCCulloch and Walter Pitts [2], developed the perceptron model [3]. These first advances on artificial neurons were inspired by the discovery of Camillo Golgi and Santiago Ramón y Cajal, on the biological neuron and its functioning. These discoveries were recognized with the Nobel Prize in Medicine in 1906.

The basic unit of an ANN is the neuron, when several neurons are connected, they form a network, and when the connections between different neurons are modified, the behavior of the network changes. The process of tuning the connections between neurons in a network to a final objective is known as learning.

During the second half of the 20th century, research on the fundamentals of neural networks, learning mechanisms, and applications progressed steadily. Although their potential remained latent for decades, the end of the 20th century and especially the beginning of the new century brought remarkable technological advances in computing and an explosion of data from the Internet. The information age had arrived, and artificial intelligence (AI), especially ANNs, was of great interest to society. The cases of success are innumerable, covering problems in any type of sector such as economics, engineering, robotics, medicine, ecology, etc.

In general, the machine learning problems in which networks have shown outstanding performance are classification and regression. In the first case, classification, one of the main engines of research has been the search for solutions for computer vision systems. In this field, researchers for decades have devoted their efforts to finding the main features of images on which to apply traditional classification strategies. Convolutional Neural Networks (CNN) burst

with force in the field of vision by the hand of classification systems such as AlexNet [4]. The main advantage is that they are able to learn the main features from the training set. This allows to find the hyperspace of variables that maximizes the separation of classes.

As with regression problems, natural language processing has been and continues to be an important area of research for neural networks. This is due to the complexity of modeling long-term temporal properties that are difficult or impossible to approximate by Markovian models such as state space models. In this area, Recurrent Neural Networks (RNN) with models such as Long-Short Term Memory (LSTM) [5] or Gated Recurrent Units (GRU) [6] have proven to be able to extract nonlinear and Markovian trends.

Although papers such as the Prof. O.I. Abiodun et al. [7] discuss the hegemony of CNNs and the RNN, as of the date of this manuscript, it is strange to find papers on AI that do not mention "Transformers". According to Stanford University researchers in [8], transformers "*represent a paradigm shift for AI*". This type of network has been published by A. Vaswani et al. [9] as a proposal for the problem sequence 2 text translation. It is characterized by learning from context without recurrent and convolutions. This is another level of intelligence if we compare it to humans, because from a human point of view, understanding, interpreting, or reasoning the context is another level of abstraction.

5.2. The basic unit of ANN

The artificial neuron tries to reproduce the essence of biological neural systems and emulate their behavior. The first advances date back to the 1950s and 1960s, when the scientist Frank Rosenblatt developed the perceptron [3]. These early advances in artificial neurons can be described as follows:

- Each neuron gets a series of "inputs" (either original data or "outputs" from other neurons). Each input arrives through a connection that has a certain "strength", or "weight", which is equivalent to the synaptic efficiency of a biological neuron. Each neuron also has a certain threshold value. The weighted sum of the inputs minus the threshold value makes up the "activation" of the neuron (also known as the Post-Synaptic Potential (PSP) of the neuron).
- The activation signal flows through an activation function, or transfer function, to produce the "output" of the neuron. This function limits the range of values that the neuron's output variable can take.

Consider that there is a signal x_j at the input of j-synapse of neuron k, when crossing through the j-th synaptic connection, the variable x_j is multiplied by the “weight” w_{kj} and a

threshold b_k ("Bias") is added to the result. This result z, becomes mapped by an activation function φ. In the literature it is usual to consider b_k as an extra weight w_{k0}.

$$a_k = \varphi(z_k) | \; z_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{5.1}$$

5.2.1. Activation Neurons

The first functional artificial neuron, called perceptron, was composed of a step activation function φ, which can only take binary values. This function is also called the McCulloch-Pitts neuron and is defined by (5.1) Fig. 5.1 (a). It is commonly used in classification problems because of its binary output.

$$\varphi(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{5.2}$$

The limitation in the input and output values of the perceptron motivated the study of other kind of activation functions. The first substantial change occurred when using the sigmoid neuron (5.3). This neuron can take input values between 0 and 1 as can be show in Fig. 5.1 (b).

The sign of a sigmoid function means whether it opens to the left or to the right. It is well suited to represent concepts such as "very large" or "very negative". The importance of this function is that its derivative is always positive and close to zero for big values. This feature is essential when applying the learning rules. If the slope is very high, the sigmoid will approximate the step function.

$$\varphi(a, z, c) = \frac{1}{1 + e^{-a(z-c)}} \tag{5.3}$$

In addition, if it is desired to translate the interval from [0 1] to [-1 1] the sigmoid function can be defined as the hyperbolic tangent (5.4), Fig. 5.1 (c).

$$\varphi(z) = \tanh z = 2sigm(2z) \; - 1 \tag{5.4}$$

Another derivable function used in the literature is the Gaussian function (5.5), Fig. 5.1 (d). In this case, the width and amplitude can be adjusted using the A and B parameters.

$$\varphi(z) = \mathrm{A} e^{-Bz} \tag{5.5}$$

The search for solutions to nonlinear problems has led to the generation of models with great depth, in other words, a large number of neurons (nodes) between the input and output of the system. This implies a high computational cost for systems with derivable but complex activation functions, such as sigmoidal or Gaussian. One of the proposals to preserve the

degree of saturation at the extremes and the derivability is the piecewise defined function (5.6), Fig. 5.1 (e).

$$\varphi(z) = \begin{cases} 1, & z \leq -1/2 \\ z + 0.5, & 1/2 > z > -1/2 \\ 0, & z \geq 1/2 \end{cases} \tag{5.6}$$

The slope in the linear region is assumed to be unity but can easily be approximated by the step function when the slope is infinite. In all other cases it is the combination of three simple functions.

However, the most popular function at present is the Rectified Linear Unit (ReLU) function (5.7), Fig. 5.1 (f). These functions let all positive values pass without changing them, while setting all negative values to zero. It could be understood as a piecewise definite function with slope one and zero ordinate at the origin.

$$\varphi(z) = \max(0, z) \tag{5.7}$$

ReLU has two main advantages:

- Non-saturated gradient, by the fact that x > 0; thus, the problem of gradient dispersion in back propagation process is alleviated and the parameters in the first layer of the neural network can be updated quickly.
- Low computational complexity, given its own definition.

However, it has the disadvantage that the ReLU neuron may die when it receives a high negative gradient during backpropagation. This can be avoided by carefully initializing the weights or using Leaky ReLU (5.8), Fig. 5.1 (f), which is similar to ReLU, but its output is linear multiplied by a small value (about 0.001) when the input is negative. This reduces the possibility of node death by avoiding returning zero for values below zero.

$$\varphi(z) = \max(0.01\mathrm{z}, \mathrm{z}) \tag{5.8}$$

Relevant reviews, such as that of Professor Tomasz Szandała [10], approach their study from the formulation of learning problems and their challenges. In addition, it describes the behavior of other neurons such as Swish, Softplus or Maxout which may be alternatives to ReLU.

Fig. 5.1 shows the function of each of the activation functions described above.

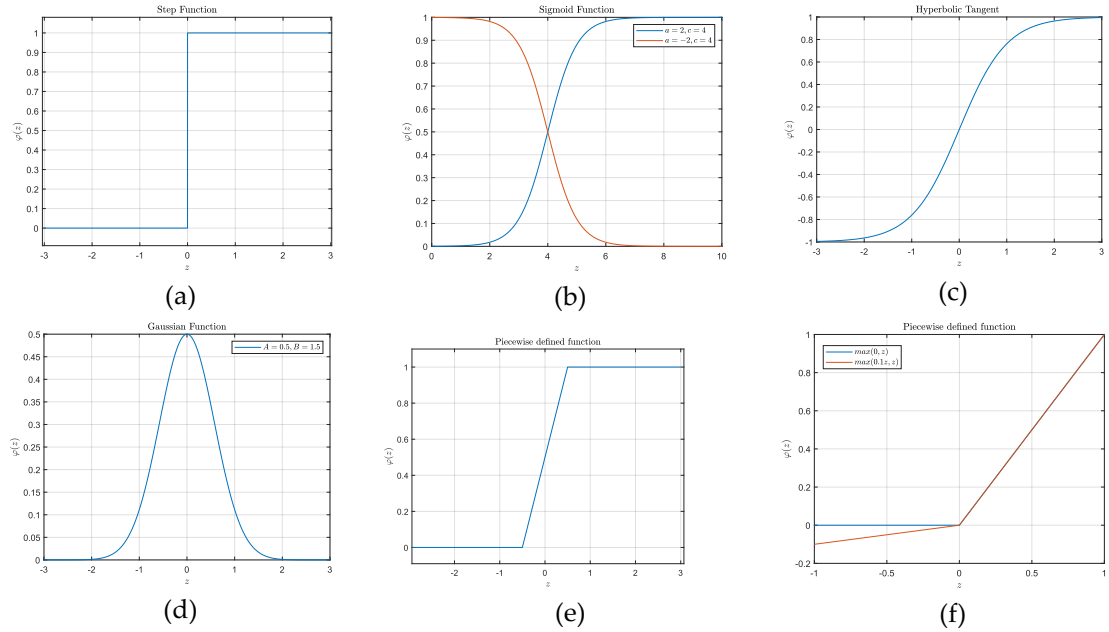


Fig. 5.1: Activation functions. a) Step function. b) Sigmoid function with [a, b] = [2,4] and [a, b] = [-2, 4] (red). c) Hyperbolic tangent function, d) Gaussian function with A=0.5 B=1.5. e) Piecewise function. f) ReLU and Leaky ReLU (red).

5.3. Artificial Neural Network

In the previous section it was shown how the input of a neuron is the linear combination of its different inputs. the union of different neurons forms a network. One of the most generalized ways to connect neurons is a layered structure. This divides the network into three main parts, an input layer, an output layer, and one or more layers in between, called hidden layers. This is the topology of a feedforward network [11]. In it, all the nodes of one layer are connected to each node of the next layer, thus feeding the successive layers to each other. Each node is a neuron. As in the basic neuron unit, biases can be added in all layers (input, output and hidden).

Each l-layer of a network with n-layers is defined by L_n, so the first layer is L_1 and the output L_n. The dimensionality of a layer is defined by the number of neurons that compose it (s_l).

In this type of architecture, all neurons are connected, with each connection having its own weight. For an input, the neurons would start a cascade of operations, receiving inputs and passing the outputs to the neurons of the next layer. This can be seen very graphically with the visualization tool provided by TensorFlow [12]. The network parameters consist of $W_{ij}^{(l)}$ and $b_i^{(l)}$. The former corresponds to the weight between the j^{th} neuron of layer L_l and the i^{th} neuron of layer L_{l+1}. For example, $W_{12}^{(3)}$ means the connection coefficient between

the second neuron of the third layer and the first neuron of the fourth layer. The other parameter, $b_i^{(l)}$, denotes the bias associated with the i^{th} neuron of layer L_{l+1}.

To reduce computational cost, weight parameters are stored in matrices and vectors in a process called "vectorization". By organizing the parameters into matrices and applying tensor operations, it is possible to take advantage of fast linear algebra routines to perform the neural network computation efficiently. Finally, as shown in (eq.), the weight coefficients $W_{ij}^{(l)}$ and $b_i^{(l)}$ can be vectorized.

$$W^{(l)} = \begin{bmatrix} W_{11}^{(l)} & \cdots & W_{1s_l}^{(l)} \\ \vdots & \ddots & \vdots \\ W_{s_{l+1}1}^{(l)} & \cdots & W_{s_{l+1}s_l}^{(l)} \end{bmatrix} \; ; \; b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \vdots \\ b_{s_{l+1}}^{(l)} \end{bmatrix} \tag{5.9}$$

The dimension of the matrix $W^{(l)}$ will be $s_{l+1} \times s_l$, or zero in the case of no connection, and the length of the vector $b^{(l)}$ will be s_{l+1}. Moreover, these values are the parameters that define the connections between the layers L_l and L_{l+1}. Therefore, in a neural network with n_l layers, it will be composed of n_{l-1} matrices/vectors for each parameter.

Taking $a_i^{(l)}$ as the output value or activation of the i^{th} neuron in the L_l layer. For the input layer $l = 1$, the activation will be the i^{th} input signal, then $a_i^{(1)} = x_i$. For the output layer, $l = n_l$, the activation will be the output signal, defined as $h_{W,b}(x)$. This is the hypothesis that will define a neural network given fixed parameters W and b.

To generalize the activation function of a feedback neural network, the definition of (5.1) can be used. First, we want to identify the activation function of each layer. To do this, recall that the output of the i-neuron in layer L_l, called $a_i^{(l)}$, is the output of the activation function φ, which takes as input the sum of the activations of all the neurons connected in layer L_{l-1} to the i-neuron in layer L_l, multiplied by the coefficients $W_{ij}^{(l-1)}$ of each connection, and adds a bias $b_i^{(l-1)}$. Thus, the i-neuron of layer l is defined by (5.10).

$$a_i^{(l)} = \varphi\left(\sum_{j=1}^{s_{l-1}} W_{ij}^{(l-1)} \cdot a_j^{(l-1)} + b_i^{(l-1)}\right) \tag{5.10}$$

For l-layer, L_l, the matrix form is given by (5.11).

$$a^{(l)} = \varphi\left(\begin{bmatrix} W_{11}^{(l-1)} & W_{12}^{(l-1)} & \cdots & W_{1s_{l-1}}^{(l-1)} \\ W_{21}^{(l-1)} & W_{22}^{(l-1)} & & W_{2s_{l-1}}^{(l-1)} \\ \vdots & & \ddots & \vdots \\ W_{s_l1}^{(l-1)} & W_{s_l2}^{(l-1)} & \cdots & W_{s_ls_{l-1}}^{(l-1)} \end{bmatrix} \cdot \begin{bmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ \vdots \\ a_{s_{l-1}}^{(l-1)} \end{bmatrix} + \begin{bmatrix} b_1^{(l-1)} \\ b_2^{(l-1)} \\ \vdots \\ b_{s_{l-1}}^{(l-1)} \end{bmatrix}\right) \tag{5.11}$$

Typically, the weighted sum of all inputs of the i-neuron in layer L_l, $z_i^{(l)}$, is used to compact the notation (5.12).

$$a_i^{(l)} = \varphi\left(\sum_{j=1}^{s_{l-1}} W_{ij}^{(l-1)} \cdot a_j^{(l-1)} + b_i^{(l-1)}\right) = \varphi\left(z_i^{(l)}\right) \tag{5.12}$$

What has been shown is a basic concept of a feedforward neural network. These networks have no feedbacks or loops between neurons or layers. Classification systems have as many neurons as classes at the output of the network, and recurrent networks feedback their layers and neurons. This increases the complexity of the architectures and consequently their computational cost.

5.3.1. The space power of CNNs

To understand the power of CNNs, it is important to put them in the context of their previous appearance in the field of computer vision and neural networks. A low-resolution image, say 128x128 [px], in three-channel Red, Green, Blue (RGB) format contains a total of 128x128x3=49,152 data. Approaching this problem from the back-propagation neural network approach requires an input layer with almost 5000 neurons. The computational cost was excessive for the time, so dimensionality reduction strategies were applied, where the search for main features in the images was widely accepted. However, these features were based on concepts such as corners, edges, etc. With these features, the dimension of the space is reduced, but in a proper way, some of the information contained in the image may be wasted. Faced with the dimensionality problem posed by the challenges of pattern recognition, K. Fukushima [13] came up in 1980 with the "Neocognitron", the forerunner of what we know as Convolutional Neural Networks (CNN). At the end of the 1980s, work such as that of LeCun et al. [14] appeared, which refined and applied the concepts of convolutional kernels and successfully trained the architectures using the "back-propagation" algorithm [15].

These mathematical mechanisms together with activation layers allow CNNs not only to stabilize the input dimensionality, but also to learn hidden features in the image data.

Although they are widely known for their efficiency in problems related to images and classification, there are many cases in which they have been successfully applied in vessel trajectory classification [16], audio classification, temporal sequences or regression, among others [17].

Classical convolutional blocks such as those of the relevant AlexNet architecture [4] are composed of three layers, convolution, activation, and pooling [17]. This configuration allows

concatenating convolutional blocks, making easier the dimensionality structure between inputs and outputs.

For a classification architecture with CNN, this set of blocks provides the feature extraction. In addition, the activation layers of the convolutional blocks are the focus of attention mechanisms in image detection systems such as networks mentioned in Part I, Chapter 3 Machine vision systems, section 4 of object detection.

5.3.2. The sequential domain of the RNNs

Temporal dependencies in data, such as time series, can be modeled by Markov chains [18]. Markov chains model the transitions between states of an observed sequence. Hidden Markov Models (HMM) appear 50 years later, in the 1950s, and probabilistically model an observed sequence depending on a sequence of unobserved states [19]. But these models have limitations. First, they are limited because their states must be extracted from a reduced state space. Programming algorithms for inference with these model's scale with time. Transitions that capture the probability of moving between adjacent states become infeasible with an HMM when the set of possible hidden states is large. Furthermore, each hidden state can only depend on the immediately preceding state, as is the case with dynamical systems in state space models.

While it is true that the temporal window can be expanded by creating new state spaces, the state space grows exponentially with the size of the window, making them computationally impractical for modeling long-term dependencies.

Given these limitations, recurrent networks attempt to address the limitations of HMMs, especially long-term temporal dependencies.

According to Z.C, Lipton and J. Berkowitz [19], "*Recurrent neural networks are feedforward neural networks augmented by the inclusion of edges that span adjacent time steps, introducing a notion of time to the model*."

This assumes that the connection between edges connecting adjacent steps produces cycles. The size of the cycle denotes the degree of recurrence, where one cycle means self-connections of a neuron or node to itself. It is assumed that at t-time step, the nodes receive the newly data $x(t)$, but also information from the hidden states at previous times $h(t-1)$. Thus, the output $\hat{y}(t)$ is computed with the hidden nodes $h(t)$. This allows a past input $x(t-1)$ to influence the current output $\hat{y}(t)$, (5.13).

$$h(t) = \varphi(W_{hx}\, x(t) + W_{hh} x(t-1) + b_h) \tag{5.13}$$

Where W_{hx} are the weights between the inputs x and the hidden layer h. On the other hand, W_{hh} is the matrix of recurrent weights between itself and adjacent time steps. The vector b_h is a bias vector of the hidden layers.

Papers published by Jordan [20] in 1987 and by Elman [21] in the early 1990s introduced connections between outputs and new neurons connected to hidden layers called "context units". The goal of these networks is to learn the context of the input data. These works were essential for the development of the popular LSTMs [5] almost ten years later and the GRUs networks [6] in 2014.

5.3.3. Transformers: Understanding the context

Recently, AI fields related to natural language processing (NLP) and computer vision have been catapulted in performance and efficiency by ANN models such as GPT2/3/4 [22], SAM [23] or XMem [24], among others.

Transformers base their potential on attention mechanisms. Attention mechanisms allow networks to focus learning efforts on understanding context. One of their advantages is the reduction of computational cost, which allows to process a larger amount of data. The vanilla Transformer, unlike RNNs, does not have recurrences, but instead uses positional coding to model sequences [25]. The central block of the vanilla Transformer architecture is the self-attention module. It is assimilated to a fully connected layer in which weights are dynamically generated based on the input patterns. The result of this block makes it possible to maintain the maximum length of a fully connected layer, but with far fewer parameters. This feature makes it suitable for modeling long-term time properties. The review paper by M. Guo et al. [26] describes attention mechanisms depending on the data domain. The authors divide attention methods into six categories, which are further divided into two groups: basic mechanisms and hybrid mechanisms. The basic mechanisms include:

- Chanel attention: aims to identify "*what to pay attention to"*.
- Spatial attention: aims to identify "*where to pay attention"*.
- Temporal attention: seeks to identify "*when to pay attention"*.
- Branch attention: aims to identify "*which to pay attention to"*.

On the other hand, the hybrid mechanisms are Chanel & spatial attention and Spatial & temporal attention.

Attention mechanisms allow parallelizing the internal processes reducing the computational cost of training with respect to RNN and CNN [27]. This allows training with huge databases as in the case of the SAM [23] (Segment Anything Model), which was trained with more than 11 thousand images and 1 billion masks. This does not mean that transformers improve the

performance of RNNs or CNNs in all cases, but they do improve the computational cost in training and inferences.

Attention models have limitations, including that they can be difficult to train, requiring complex optimization methods and/or new strategies. These difficulties, together with their complexity, can be negative for some applications. In addition, the parallelization of attention mechanisms and input tokens limits the exploitation of the full sequential potential of networks [28]. However, all these limitations open the possibility of exciting challenges for researchers and IA.

5.4. References

[1] C. Dictionary, "Cambridge Dictionary | English Dictionary, Translations & Thesaurus." https://dictionary.cambridge.org/ (accessed Oct. 28, 2022).

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[3] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958.

[4] B. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2012.

[5] S. Hochreiter, "Long Short-Term Memory," vol. 1780, pp. 1735–1780, 1997.

[6] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *EMNLP 2014 - 2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, pp. 1724–1734, 2014, doi: 10.3115/v1/d14-1179.

[7] O. I. Abiodun *et al.*, "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition," *IEEE Access*, vol. 7, pp. 158820–158846, 2019, doi: 10.1109/ACCESS.2019.2945545.

[8] R. Bommasani *et al.*, "On the Opportunities and Risks of Foundation Models," pp. 1–214, 2021.

[9] A. Vaswani *et al.*, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 5999–6009, 2017.

[10] T. Szandała, "Review and comparison of commonly used activation functions for deep neural networks," *Stud. Comput. Intell.*, vol. 903, pp. 203–224, 2021.

[11] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.

[12] TensorFlow, "A Neural Network Playground." https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.79563&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=fal (accessed May 18, 2023).

[13] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern

recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, 1980.

[14] Y. LeCun, B. Boser, J. Denker, … D. H.-N., and U. 1989, "Backpropagation applied to handwritten zip code recognition," *ieeexplore.ieee.org*, Accessed: May 18, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6795724/

[15] D. E. Ruineihart, G. E. Hint, and R. J. Williams, "Learning internal representations by error propagation," 1985, Accessed: May 18, 2023. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA164453

[16] J. P. Llerena, J. García, and J. M. Molina, "LSTM vs CNN in Real Ship Trajectory Classification," in *16th International Conference on Soft Computing Models in Industrial and Environmental Applications, SOCO 2021, Advances in Intelligent Systems and Computing*, 2021, vol. 1268 AISC, no. SOCO, pp. 58–67.

[17] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *J. Big Data 2021 81*, vol. 8, no. 1, pp. 1–74, Mar. 2021, doi: 10.1186/S40537-021-00444-8.

[18] W.-K. K. Ching Ximin Huang Michael Ng Tak-Kuen Siu Models, "International Series in Operations Research & Management Science Markov Chains", Accessed: May 24, 2023. [Online]. Available: http://www.springer.com/series/6161

[19] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning," pp. 1–38, 2015, [Online]. Available: http://arxiv.org/abs/1506.00019

[20] M. I. Jordan, "Serial order: A parallel distributed processing approach," *Advances in Psychology*, vol. 121, no. C. 1986.

[21] J. L. Elman, "Finding Structure in Time," *Cogn. Sci.*, vol. 14, no. 2, pp. 179–211, Mar. 1990, doi: 10.1207/S15516709COG1402_1.

[22] C. Zhang *et al.*, "A Complete Survey on Generative AI (AIGC): Is ChatGPT from GPT-4 to GPT-5 All You Need?," vol. 1, no. 1, pp. 1–56, 2023, [Online]. Available: http://arxiv.org/abs/2303.11717

[23] A. Kirillov *et al.*, "Segment Anything," 2023, [Online]. Available: http://arxiv.org/abs/2304.02643

[24] H. K. Cheng and A. G. Schwing, "XMem: Long-Term Video Object Segmentation with an Atkinson-Shiffrin Memory Model," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2022, vol. 13688 LNCS, pp. 640–658. doi: 10.1007/978-3-031-19815-1_37.

[25] Q. Wen *et al.*, "Transformers in Time Series: A Survey," 2023, [Online]. Available: http://arxiv.org/abs/2202.07125

[26] M. H. Guo *et al.*, "Attention mechanisms in computer vision: A survey," *Comput. Vis. Media*, vol. 8, no. 3, pp. 331–368, 2022, doi: 10.1007/s41095-022-0271-y.

[27] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, vol. 3, no. September, pp. 111–132, 2022, doi: 10.1016/j.aiopen.2022.10.001.

[28] F. Becattini and T. Uricchio, "Memory Networks," pp. 7380–7382, 2022, doi: 10.1145/3503161.3546972.

Chapter 6: An approach to forecasting and filtering noise in dynamic systems using LSTM architectures

Juan Pedro Llerena Caña [1]*, Jesús García Herrero [1] and José Manuel Molina López [1]

[1] Carlos III University of Madrid (Madrid-Spain), Applied Artificial Intelligence Group (GIAA); jllerena@inf.uc3m.es (J.P.LL.), jgherrer@inf.uc3m.es (J.G.) and molina@ia.uc3m.es (J.M.M.).

* Correspondence: jllerena@inf.uc3m.es

Abstract. Some of the limitations of state-space models are given by the difficulty of modeling certain systems, the filters convergence time, or the impossibility of modeling dependencies in the long term. Having agile and alternative methodologies that allow the modeling of complex problems but still provide solutions to the classic challenges of estimation or filtering, such as the position estimation of a mobile with noisy measurements and unknown motion models, are of high interest. In this work, we address the problem of position estimation of 1-D dynamic systems from a deep learning paradigm, using Long-Short Term Memory (LSTM) architectures designed to solve problems with long term temporal dependencies, in combination with other recurrent networks. A deep neuronal architecture inspired by the Encoder-Decoder language systems is implemented, remarking its limits and finding a solution capable of making predictions of high accuracy with models learnt from training data of a moving object. We use a panel data model for training and validation. In the experimentation, we use sliding overlapping time windows in a recursive and standardized way to avoid the saturation problem of the networks in increasing trend estimates. The results are finally compared with the optimal values from the Kalman filter, obtaining comparable results in error terms. These results show the proposed system has great potential for target tracking.

Keywords: Deep Learning, Filtering, Forecasting, LSTM, Encoder-Decoder, Attention.

6.1. Introduction

A wide variety of physical and scientific problems are based on the estimation of the state variables of a system that evolves with time, using for these purpose sensors that provide measurements with a certain level of uncertainty, so-called noisy observations.

To a large extent, these problems are formulated with state-space approximations. These approaches model the system behavior through a mathematical approximation mainly centered on a state vector, which is intended to contain all relevant and necessary information to describe it and make predictions. The sensors provide measurement or observation vectors that are related to the state vector of the analyzed system.

To analyze and infer a dynamic system, it is mainly required a model that describes the evolution of the states with time, and a second one that relates the observations with the states. These two large groups can be denominated from the state-space formulation as equations for state dynamics, and equations for observations (or likelihood), respectively.

In this context, many problems are tackled from the probabilistic formulation of the state space with Bayesian approximations, which provide a general solution for dynamic states estimation problems. Knowing the governing equations for dynamic systems allows forecasting, estimations, or control studies by structural stability analysis and bifurcations. However, when systems are very complex and/or when measurements are corrupted by not modeled errors [1], many complications may appear. In H. H. Afshari et al. [2] work can be found a summary of different state estimation techniques from classical and Bayesian perspectives.

It has been addressed that State Space Models (SSM), such as Hidden Markov Models (HMM) and Linear Dynamic Systems (LDS), have been and continue to be powerful tools for series modeling, estimation, and filtering. However, these approaches are based on linear, Gaussian, or Markov assumptions, while in real systems it is difficult for them to be linear or Gaussian and can have long-term dependencies that cannot be captured by these techniques, so their use is restricted.

Distinguishing the aforementioned cases by their probabilistic inference model, using artificial intelligence (AI) paradigms we can add intelligent inference methods.

If we group under AI data-driven estimators, there are recent novel works such as Tianchen Li and Honqui Fan [3]. In this work, the authors address the problem of real-time detection and tracking of non-cooperative targets [4] in the challenging scenario of not having a priori target information such as target dynamics, birth, death, or probability of detection. For the estimation of the movement, authors uses trajectory functions on time (T-FoT) [5]. This system fits a polynomial-time function to the received data. Least squares and Mahalanobis distance are used to adjust the polynomial-time parameters. This estimator is adjusted iteratively by applying online training with sliding-time windows, thus allowing it to adapt to hostile situations such as maneuver changes. In addition, the authors propose an initialization system based on clustering and a hypothesis test to identify if two measurements belong to the same object. This binary test is based on the Mahalanobis distance and the neighborhood radius. The proposal is evaluated in two cases of non-cooperative objectives. On the one hand a linear system and on the other hand a non-linear system, using the optimal subpattern assignment error (OSPA) [6] as a metric. In addition, it is compared with an ideally modeled Bernoulli filter [7, 8]. The proposal shows comparable results with the ideally modeled Bernoulli filter and

even outperforms it under certain conditions, demonstrating that the proposal provides a promise alternative to state space models.

In [9] software sensors are treated as an alternative way to obtain estimators by means of classical methods. These AI-based estimators are computational algorithms designed to predict unmeasured parameters that are relevant for developing control laws or other applications.

LSTM neural architectures are not new [10], having been used in many applications that were related with natural language processing [11] or attention [12] problems. Additionally, LSTM has shown good results in other scenarios, such as classification systems [7, 8], signal filtering after measurement [15], time estimations (e.g., oil production estimation [16]), traffic forecasting [17], stock index prediction [18] and system modeling [19], among others.

Orimoloye's and working team [18] compare deep learning (DL) architectures with soft architectures such as a support vector machine (SVM) to predict stock indexes with different sample times (hours, minutes, seconds) observing that predictive accuracy reaches a maximum point regardless of the size of the training set. It is also verified how the rectified linear unit (ReLU) activation function presents better results than the tanh (hyperbolic tangent).

Rassi et al. [20], model highly non-linear systems that are restricted in the state space or centered around equilibrium points with ideal synthetic data. Rudy et al. work [1], models highly non-linear systems with noisy measurement information. The aim of this work focuses on the modeling of highly nonlinear systems using neural networks combined with numerical methods around equilibrium points or attractors that prevent saturation effects in the neural models. Section 4 of the paper [1] expresses certain limitations that mainly reflect the problems of estimation very close to the attractors and even with changing initial conditions. The neural architectures used are composed of dense layers in a four-step Runge-Kutta scheme without a study of the associated forgetting rate, so that in non-Markovian models they may present different results since they are not designed to maintain long-term trends. Both papers [1] and [20] show the behavior of a single trajectory under different noise levels but no general statistical analysis of the performance of the models is performed. In the framework of target tracking challenges, it is interesting to study the behavior of the systems in the absence of measurements, however, in both papers, the authors focus especially on the modeling problem of the systems. These limits knowing the potential of this work in the face of the interesting challenge of monitoring objectives in the absence of measures.

In Zheng et al. work [21] present a new combined algorithm between LSTM and Monte Carlo for tracking, testing a continuous increasing function with noise (line) but bounded to a specific time sequence.

In this paper we tackle the estimation/filtering problem with the position in a 1D moving object with an RNN inspired by the language encoder-decoder systems among others, comparing with the optimal solution of the Kalman filter (KF). This work brings to the neuro-estimators area, a new neural architecture, and we obtained comparable results in terms of error to Kalman and opening new alternatives to problems not approachable by classical systems. Our work, in contrast to the majority of studies or problems in the literature such as [1, 14, 15] delocalizes the problem from a specific or bounded estimation region and generalizes it, transforming in it into a recursive standardization-inference-unstandardization problem. For this purpose, the system is trained in a wide range of initial conditions.

6.2. Problem formulation

We suppose an unknown dynamic system $f(x(t))$ not necessarily linear, Gaussian, or Markovian. From the system, we know $z(t)$ measurements of some of its states x in time t. These measurements are related to the states of the system f by $h(x(t), v(t))$ function. Generally, h can be considered nonlinear and dependent on a stochastic parameter with noise $v(t)$.

$$\frac{dx(t)}{dt} = f(x(t)) \tag{6.1}$$

$$z(t) = \mathrm{h}(x(t), v(t)) \tag{6.2}$$

Where $x(t) \in \mathbb{R}^n$ is a state vector f and is a state vector field.

Not knowing if the stochastic system can present temporary dependencies in the long term means that the estimation of states of a system cannot be done only with immediate contiguous states in time. In this way discretization of a continuous system can be done as follows:

$$x_{k+1} = F(x_k) = \int_{t_l}^{t_{k+1}} f(x(\tau))\mathrm{d}\tau \tag{6.3}$$

Where t_l is a temporal instant less than k and generally unknown in non-Markovian systems, where the approach for the previous discretization can no longer be used. In this way, a classical dynamic system can be considered as a particular case of a non-Markovian system.

According to this notation, forecasting state problems is formulated in relation to previous states (6.3), that is to say, forecasting consists of identifying states in future times (x_{k+1}). On the other hand, a filtering problem consists to identify certain x_k states at the same moment in which z_k noise measurements are received (6.4).

$$x_k = h^{-1}(z_k, v_k) \tag{6.4}$$

However, in real problems, the value of v_k is not usually known, neither if the function h is invertible and in case of being able to demonstrate its local or global existence by the inverse function theorem, this will only tell us if it exists, but not what it is or how to calculate it. But it is important to note that the function h is most likely not invertible in real cases.

This paper proposes the estimation of state variables of an a priori unknown noisy dynamic system, which may not depend only on a previous state but may present long- term temporal dependencies.

Thus, the state vector must be estimated from the observations. If we call $\hat{F}^+$ and $\hat{F}$ filtering and prediction estimators respectively, the problem is how to determine the estimators from the data.

$$\hat{x}_k = \hat{\mathrm{F}}^+(z_0, \dots, \mathrm{z_{k-1}}, \mathrm{z_k}) \tag{6.5}$$

$$\hat{x}_{k+1} = \hat{\mathrm{F}}(z_0, \dots, z_{k-1}, z_k) \tag{6.6}$$

The ∧ term over F function, mentions the estimator term that we have inherited from the classical stochastic observers notation.

For this purpose, we simulate the behavior of an ideal one-dimensional uniform rectilinear motion (URM), $W_k = [0, 0]^T$ in which all parameters are controlled and distorted under constant Gaussian noise V_k, simulating measurements z_k of the position state variable $H = [1\ 0]$:

$$\underbrace{\begin{bmatrix} p \\ v \end{bmatrix}_k}_{x_k} = \underbrace{\begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}}_{A} \overbrace{\begin{bmatrix} p \\ v \end{bmatrix}_{k-1}}^{x_{k-1}} + W_k \tag{6.7}$$

$$z_k = Hx_k + V_k \tag{6.8}$$

For this simple, short-term model, when all parameters are known, classical SSM estimation techniques can be used. However, in the general case, LSTM architectures could generate a better prediction of series using long-term dependencies without assuming linear, Gaussian, or Markovian systems.

Thus, the use of LSTM architectures has great potential in object tracking problems with complex dynamics that cannot be handled by classical systems.

In this line, we propose to approach the problem from the deep learning (DL) perspective as a "sequence-to-sequence" learning problem, widely used in natural language processing problems. We propose to adapt our neural network (NN) architecture Fig. 6.8 to a supervised

database $\boldsymbol{\Phi}$ composed by N number of Φ_i data packages. Each data package Φ_i is composed by the association of a series of measurements with noise Z_i of time length $\tilde{k}$ and n characteristics with a target series X_i of time length $\tilde{k}'$ and n' characteristics. We set $\tilde{k} = \tilde{k}'$ and $n = n'$. The temporal relationship between Z and X is that the target series X contains the filtered values of measured series Z and one temporal unit ahead, both with the same number of samples.

Given $j = \{p, v\}$ as the dimensions of state vector of the system f (6.7), $\tilde{n} = \sum(j)$ is the size $\tilde{n}$ of the full state vector x. It is important to note that X_i is a sequence and does not have to contain the complete state vector of the system (x). So, X_i is formed by the components of state vector in the i-sequence, which can be complete or not ($n \leq \tilde{n}$), directly linking to z-state measurements by Z_i sequence.

$$X_i = \begin{Bmatrix} x_{i,1,1}, \dots, x_{i,1,\tilde{k}'+1} \\ \cdots \\ x_{i,n',1}, \dots, x_{i,n',\tilde{k}'+1} \end{Bmatrix}; Z_i = \begin{Bmatrix} z_{i,1,1}, \dots, z_{i,1,\tilde{k}+1} \\ \cdots \\ z_{i,n,1}, \dots, z_{i,n,\tilde{k}+1} \end{Bmatrix} \tag{6.9}$$

In our case X_i is not complete and only includes the ideal state corresponding to positions, corresponding to the observed measurements Z_i ($\tilde{n} > n = n' = 1 | \, j = 1$).

Therefore, each data package is defined as $\Phi_i = \boldsymbol{Z_i} \cup \boldsymbol{X_i} | \; Z_i = \{\mathrm{z_{i,1}}, \mathrm{z_{i,2}}, \dots \; \mathrm{z_{i,k}}\}$ and $X_i = \{x_{i,2}, x_{i,3} \dots \; \mathrm{x_{i,k+1}}\}$. In learning terms, it can be said that the system learns to forecast a value one time step beyond the observations.

Most dynamic systems are not restricted in their state space domain, while neuronal architectures are bounded systems defined by the functions that constitute each layer. These layers are composed by the functions that define each of their units and, in greater depth, the activation functions of each of the artificial neurons. In this way, the regression problems will be bounded to the training space unless a generalization is proposed to cover all the domain.

The most common artificial neural network (ANN) activation functions are bounded, so the composition of all the neuron layers provides a system composed of bounded functions, which means that the output function of the network will be bounded. That is, the network function provides a subset of solutions restricted to the training space, which implies that estimates outside this range require special treatment.

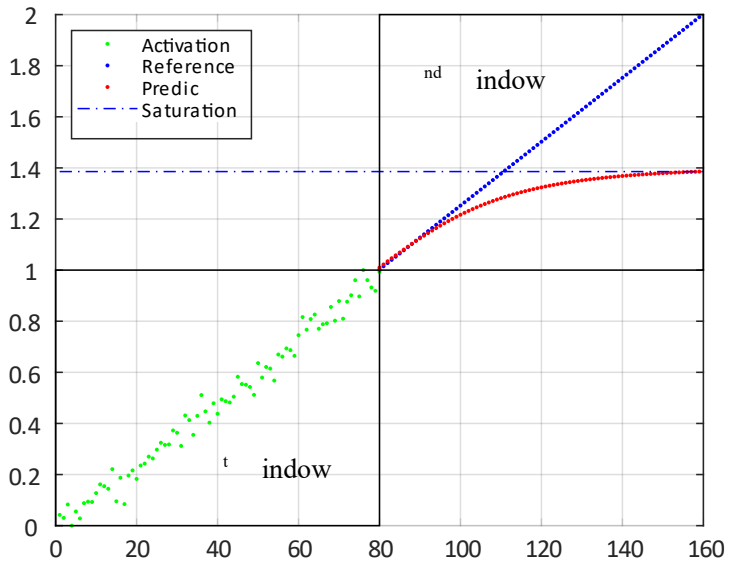


Fig. 6.1. Network saturation effect in forecasting process when we move away from the training space (watched in URM).

In Fig. 6.1 we can see in red-point forecasting positions outside the training space. A noisy data set with a full-size training window is shown in green. The blue dotted line above the second time window shows the ideal trend values. Finally, the blue horizontal line with dots and lines shows the asymptote to which the forecasts tend. In the first forecasts, the values can be adjusted to the blue target, but during few predictions, we check how the system tends asymptotically to a maximum value (Saturation). This example shows the saturation network effect when we try to forecast with a growing trend outside the training space.

This effect can be seen also in terms of internal network activation level. Fig. 6.2 shows the first 5 hidden units of the lstm-1 Table 6.3 architecture in prediction for two sequences inside the normalized space (set 1 and set 2) and the last one forecasting outside the normalized space. We can see how the first two heat maps are similar while the third one does not evolve with time and is different from the previous two and it stays with the same all the time. This means that there are no internal updates, and the LSTM cells discard the new inputs through the input gate. As there are no changes in the network while the input changes, we can say the network is saturated.

To avoid this problem, some authors work around stable equilibrium points of systems that guarantee the restriction of space along with online training. In other cases, the systems do not have growing trends, only oscillation. In this work, to address this issue, we propose using a recursive method of standardization based on the sliding time window through the data, maintaining a small overlap region with the previous window for network activation at each window shift. This overlap area hopes to retain the long-term dependencies Fig. 6.4.

The activation process consists to introducing a small section of measured sequence into the network, so that the internal network architecture can adjust its internal weights to link them to the training data. These corrections are made by transitions, and the transitions happen when measurements are inserted.

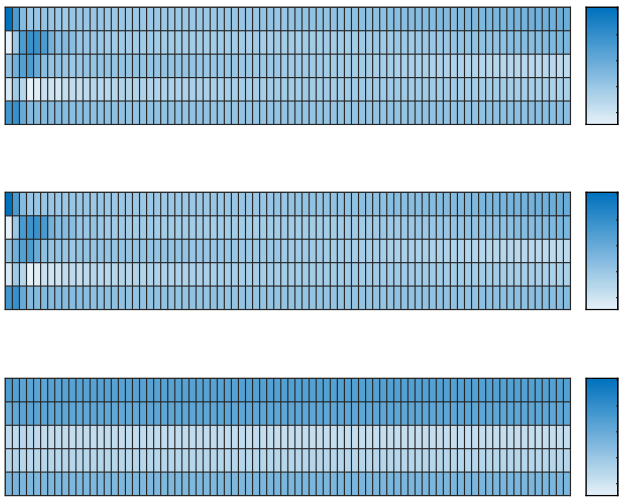


Fig. 6.2. Internal activation heatmaps. The first and second heatmaps belong to two predicted series in standard training space. Third heatmap predicted outside standard training space.

6.3. Database

We generate a synthetic database to simulate the measurement of objects positions with URM. These trajectories are linear, and they are generated with the model (6.7) to obtain ideal values. To simulate the sensor behavior, Gaussian noise is added to the ideal values (6.8).

We consider positive and negative positions and speed as initial conditions. With previous descriptions, our synthetic database $\mathbf{\Phi}$ (6.10), it's composed by N=1000 measured paths Z_i and their corresponding ideal values X_i, according to parameters in Table 6.1.

$$\mathbf{\Phi} = \bigcup_{i=1}^{N} Z_i \cup X_i \tag{6.10}$$

*w*here (6.10) means the $\boldsymbol{\Phi}$ database is the union of N data packets $\Phi_i = Z_i \cup X_i$. These ϕ_i data packets are composed of the measured data Z_i and their corresponding ideal states X_i.

The simulated values were generated by adding Gaussian noise to the ideal trajectory using a random number generator as in Kay chapter 5-9 [22] with the initial conditions in the ranges of Table *6.1*.

Table 6.1. Synthetic data generation parameters.

	Data generation range	
Parameter	Minimum	Maximum
Initial position [m] ǀ Speed [m/s]	-25 ǀ -55	25 ǀ 55
Simulation end times [s] ǀ Sampling time [s]	8 ǀ 0.05	
Number of window data ǀ Overlap [Nº data]	80 ǀ 15	
Gaussian noise measurement $\boldsymbol{V_k} \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, \boldsymbol{\sigma_Z})$	0.9	

The speed range was decided considering that the maximum speed of a vehicle for this type of problem is 198 km/h. The rest of the values have been considered in a heuristic way.

6.3.1 Database division

In supervised learning processes, the databases must be split up for training and validation processes. In this splitting process, it is common to assign a percentage of data for training and the remainder for validation. If we represent the most used general structures of the database, we can divide our data in two principal ways as shown in Fig. 6.3.

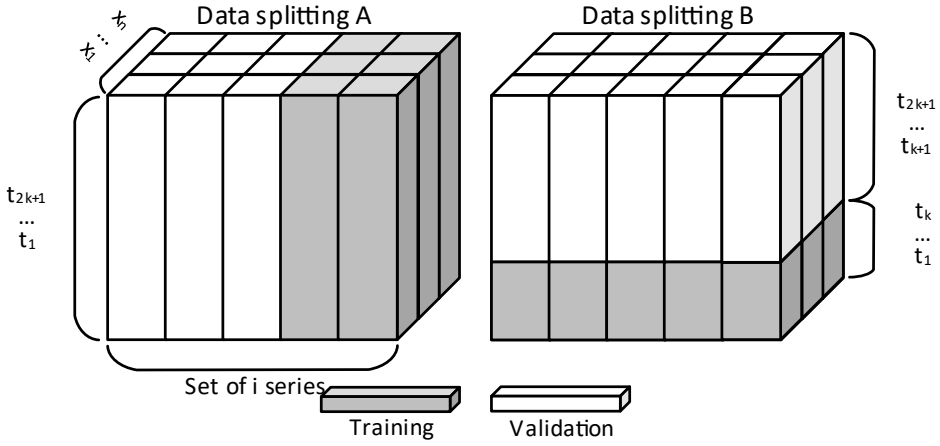


Fig. 6.3. Principal training and validation data splitting method.

With the aim of extracting the maximum information from the set of trajectories, that we assume start from different initial conditions, in this work we choose to select time windows of each of the i series option B in Fig. 6.3. This data selection can be considered as a panel data structure [23]. To control the training and validation process, we selected two sets of data from the database corresponding to two time-symmetric and contiguous time windows without overlapping with each other.

The training and validation subsets are obtained through two consecutive time windows of 81 samples for each path. The first-time window is associated to the training set and the second to the validation set, obtaining two subsets with the same number of data.

6.3.2. Data standardization

Considering the networks' sensitivity to data scaling, data standardization is performed as in X. Song [16] but under a geometrical interpretation of them. The behavior of a URM, in general, shows an increasing tendency in absolute value, so this interpretation is essential for training and model inference.

For this interpretation, we use a URM 1D trajectory given its conceptual simplicity in the visualization growing trend. This growth is a key point to control and avoid the saturation problem in the forecasting process whit ANN.

So, the activation process can have certain previous information, a small region of overlap is used between the adjacent windows, defined by a set of data from the previous window that is used for the activation of the network at each window movement (overlap).

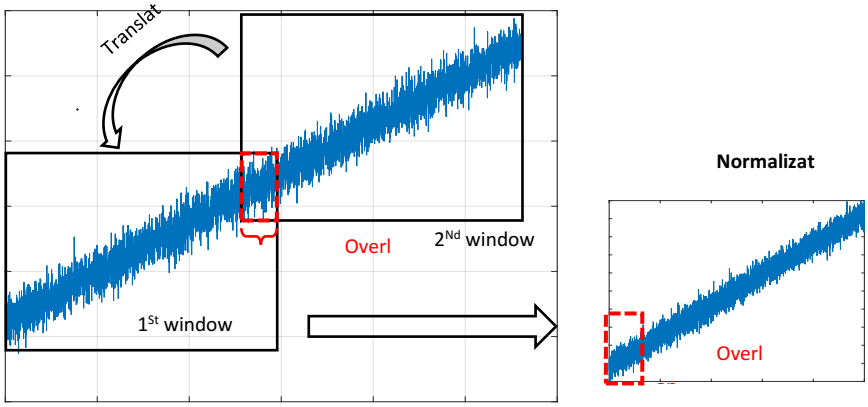


Fig. 6.4. Graphical data standardization process.

A translation is performed to transform for the second (and successive) time window into the first, by subtracting the minimum (m) value of the data series to all its measurements. Then, knowing the maximum (M) value of the window and the minimum, the normalization is done by dividing the set of data from which the minimum value has been subtracted by the amplitude of the data series in the window, which we can be obtained as the difference between the maximum value minus the minimum, this normalization represents a scaling in geometric terms. This whole process is shown in Table 6.2, procedure 1.

Finally, if we want to go back to the initial space we need to "undo" the previous rules so we can apply a unstandardization process to transform the data in the opposite order to the previous one Table 6.2. To do this we need to know the maximum (M) and minimum (m) values of the standardized data.

Table 6.2. Standardization/ unstandardization algorithm.

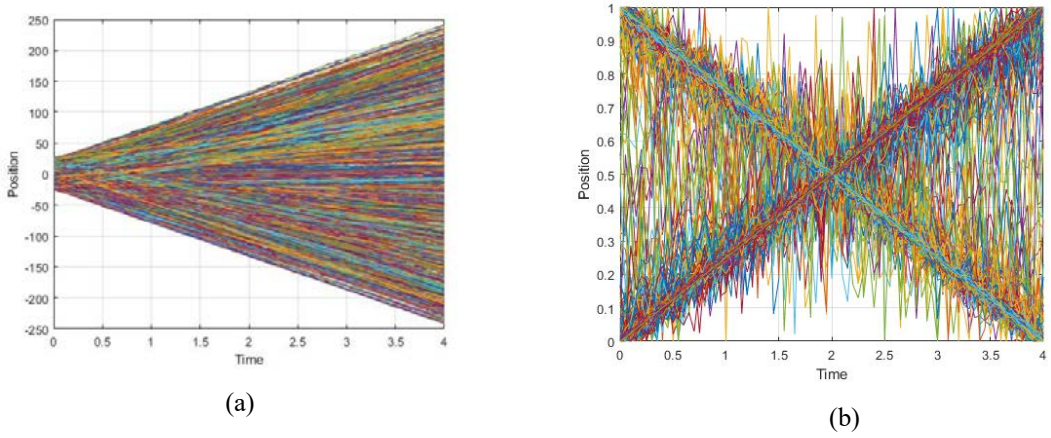
Procedure 1 STANDARDIZATION ($\boldsymbol{X}^*$)

$[m, M] = [\min(X^*), \max(X^*)]$
if (m − M) = 0
  if $M = 0$
    $X^{*\prime} = X^*$
  else
    $X^{*\prime} = (X^* - m)/M$
  end if
else
  $X^{*\prime} = (X^* - m)/(M - m)$
end if
end procedure 1

Procedure 2 UNSTANDARDIZATION ($\boldsymbol{X}^{*\prime}, \boldsymbol{m}, \boldsymbol{M}$)

if (m − M) = 0
  if $M = 0$
    $X^* = X^{*\prime}$
  else
    $X^* = X^{*\prime}M + m$
  end if
else
  $X^* = X^{*\prime}(M - m) + m$
end if
end procedure 2

The m and M parameters required for unstandardization are essential for a good fitting between the results of the standardized space and the real space with which to obtain comparative metrics, so they will be specified for each one of the experimental sections. Applying the transformation to a subset of data belonging to the database **Φ**, we get a representation like the following:



(a) (b)

Fig. 6.5. Standardization/ unstandardization dataset. (a) Raw database image, (b) Standardization database image.

In Fig. 6.5 (a) we can see different URM paths in real world position starting as indicated in Table *6.1* initial conditions range. Fig. 6.5 (b) shows URM paths under standardization procedure 1 shown in Table 6.2. Over each image, the ideal X_i paths and simulated noise measurements Z_i are shown.

In Fig. 6.5 (b) it's interesting to note how all the slopes of X_i overlap in two different classes associated with positive and negative speeds. On the other hand, it is shown how the noise associated with steep slopes is reduced with the transformation. However, low slopes increase the noise level after standardization but bounded. This differentiation and bounded data are good signs to use ANN.

In this way the standardization system allows to transform sections of trajectories from not bounded systems to a bounded space.

6.3.3. *Setting up data for training*

From a supervised learning point of view, we can define the learning objective or "target" as the ideal trajectories X_i corresponding to the measured trajectory Z_i. The input data Z_i and network target X_i have the same time lengths, and they are standardized and truncated as follows Fig. 6.6.

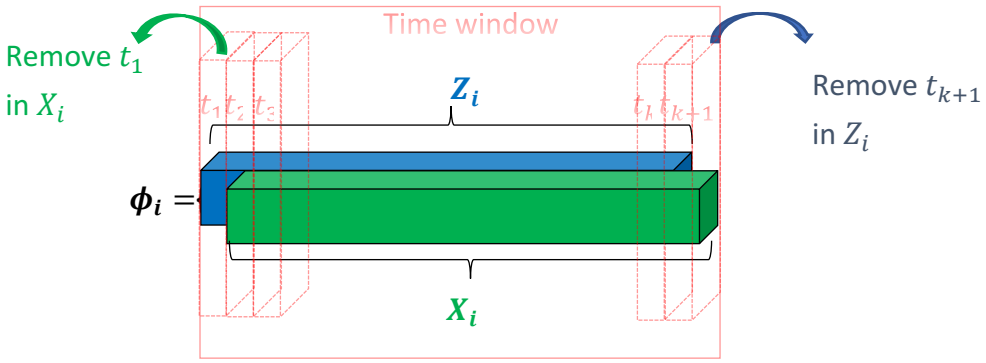


Fig. 6.6. Visual data packages structure.

From each data packet ϕ_i we take a time window size S with $k+1$ values. The time size of the Z_i measurement series and the X_i target series has the same time size S as a time unit displaced from each other. The last value of each Z_i measurement series is removed and the first value of X_i too.

In this way, data are structured for a sequence-to-sequence architecture of the same input-output dimension but shifted one unit of time, allowing the long-term estimation of how long ("window size") $X = [\overbrace{x_2, \dots, x_{k+1}}^{S}]$ target from measured values $Z = [\underbrace{z_1, \dots, z_k}_{S}]$ under a certain Gaussian noise. This process is similar to the validation paths.

6.4. LSTM neuro position estimator

In this section, we describe the general process of our proposal, the ANN architecture employed, and the training parameters used.

The general process shown in Fig. 6.7 describes at a high level the inference process with our system. First, a standardized data set is taken to activate the network (yellow area in each sliding window). Then, we check if we have new z_k measurements. If we have new z_k measurements, these are standardized in order to introduce them into the network and

predict the filtered state $\hat{x}_{k+1}$. This prediction is unstandardized to bring it back to real space. In the case without new measurement, our system introduces in the network the state forecast in the previous temporal step. The maximum number of predictions is bounded by the size of the selected time window, in our case 80 measurements (4 seconds). To remember what happened in the previous time window, the windows are overlapped with an overlapping region that is used for network activation/initialization.

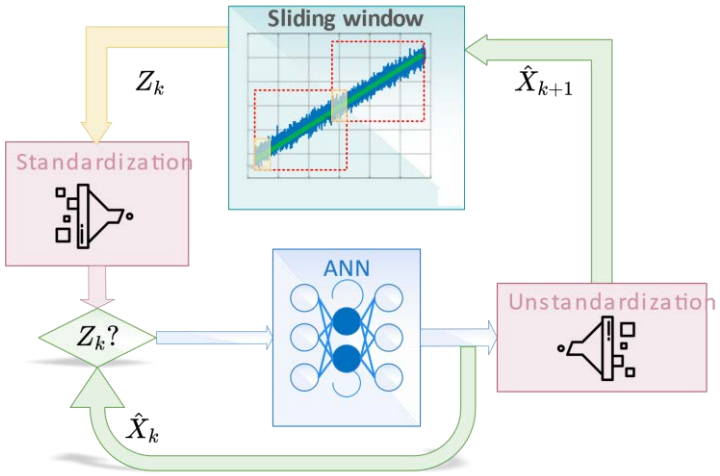


Fig. 6.7. General inferences process.

Based on the good results in linear regression models with multilayer models of perception [24] and under the stability studies in recurrent neural networks of [25], the deepest part of the network is composed of a fully connected layer with ReLU activation functions and a dropout layer of 20%. According to reminding long term tendencies and under the good results in estimation problems with LSTM architectures like [18, 19] among others, and around encoder-decoder architectures concept for non Markovian models like [11, 14, 20] together with the good results in filtering problems [15] and for the system identification with noise [1], [20] the final architecture is composed by 8 set layers Fig. 6.8.

In Fig. 6.8 the network structure presents two main high-density blocks, the encoder, and the decoder. For the encoder, the density is defined by the size of the input layer (80) and the encoder depth (400). For decoder density came from the fully connected interconnection layer (16) and from the decoder depth (200). In Table 6.3 we detail in a summarized and structured way the information of the proposed architecture.

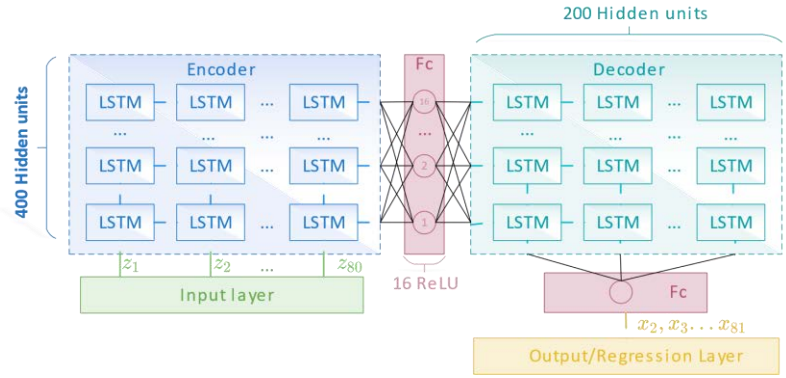


Fig. 6.8. General neural network architecture.

We can see that the input and output layers are formed by time sequences. At this point, we can ask ourselves how can the system work in real time with only one measured z_k without having a complete sequence? This happens thanks to the internal LSTM states that can interpret the input measurements and remember from learning the previous and later measurement relations. This case is considered as a partial sequence and can be predicted step by step saving and updating the internal states of the network. If we have the complete sequence, the network reconstructs the data and predicts a filtered step, while receiving step-by-step measurements predicts step-by-step. In the same way as Kalman, by having a larger number of previous measurements, the system is better adjusted to the target. On the other hand, to initialize the system we need at least one measurement. To better adjust the states of the network we take a set of measurements that we call the activation area. This activation area can be considered as a transition to minimize the error in the filtered forecast.

Table 6.3. Listing of neural network layer: S=80 is the number of samples per input trajectory

Nr	Name and type	Activation/ prop.	Learnable	States
1	Sequence Input: 1x80	1	-	-
2	lstm_1: LSTM Hidden units: 400	State activation function: tanh Gate activation function: sigm	Input Weights: 1600x1 Recurrent Weights: 1600x400 Bias:1600x1	Hidden States: 400x1 CellState:400x1
3	fc_1: Fully connected	16	Weights: 16x400 Bias:16x1	-
4	relu_1: ReLU	16	-	-
5	Do: Dropout 20%	16	-	-
6	lstm_2: LSTM Hidden units: 200	State activation function: tanh Gate activation function: sigm	Input Weights: 800x16 RecurrentWeights:800x200 Bias:800x1	Hidden States: 400x1 CellState:400x1
7	fc_2: Fully connected	1	Weights: 1x200 Bias: 1x1	-
8	Regression output	Loos function: HMSE	-	-

For training our model proposal, we used Adam optimizer for the excellent result shown in multi-layer recurrent network training [28]. We train with 20 batches during 80 epochs starting from an initial learning rate of 0.005 and with a drop of the learning factor of 0.5 after the first 8 epochs. The training updates the individual weights using the Adam algorithm but with an $\mathcal{L}_2$ adjustment of the target function under the regularization factor of 10^{-4} with the intention of reducing over/under fitting in the training process. Loss training function:

$$\mathcal{L}_{HMSE}(\boldsymbol{\theta}) = \frac{1}{2S}\sum_{i=1}^{S}\sum_{j=1}^{R}\left(X_{ij} - \hat{F}_{\boldsymbol{\theta}}(Z_{ij})\right)^2 \tag{6.11}$$

Where $\hat{F}_{\theta}$ means a network function parameterized by the internal $\boldsymbol{\theta}$ terms. These internal parameters are the weights and biases of each internal neuron. The term ^ means "estimated", which is inherited from the classical notation from stochastic observers. S is the sequence length and R is the number of sequences parameters. In our case S=80 and R=1. Our loss function is not normalized by R.

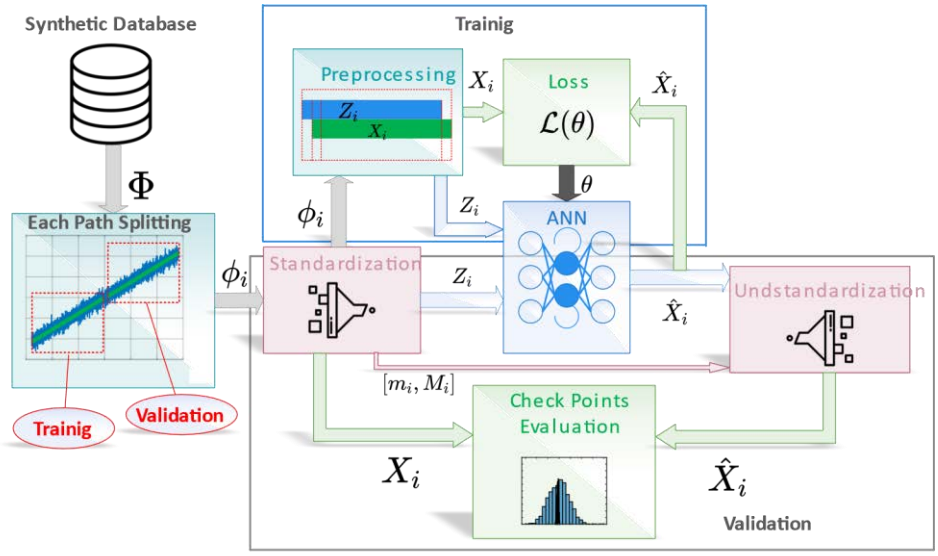


Fig. 6.9. Training and validation process.

Finally, in Fig. 6.9 we represent the training process as well as the validation process. It is important to mention that the ϕ_i data packages for the training and validation process are different and correspond to consecutive non-overlapping time windows. The upper central part of the figure shows the training process while the lower part refers to the validation process. Although the two processes are shown on the same figure, they are carried out in different and totally decoupled phases. In the experimental section, we describe in detail the evaluation process.

6.5. Experiments

The following section presents 3 different experiments. First, the LSTM model presented in chapter 0 is validated with the data set presented in chapter 0 and compare with KF, using a visual comparison over 2 histograms Fig. 6.11 of the estimation error and the metric of equation (6.12). The second experiment simulates the estimated filtering behavior as it receives new measurements. Finally, the third experiment simulates the estimation behavior with loss measurements.

In the first experiment, we use the root means square error (RMSE) (6.12) metric over two checkpoints (CP) {1,2}, the last filtered value from the validation set (CP=1), Fig. 6.11 (a), and the first one after the overlap window (CP=2), Fig. 6.11 (b), always in real space.

$$\mathrm{RMSE}_1 = \sqrt{\frac{1}{\mathrm{N}}\sum_{i=1}^{N}\left(X_{i,k,j} - \hat{X}\left(Z_{i,k,j}\right)\right)^2}\Bigg|_{\mathrm{k=CP\{1,2\}}} \tag{6.12}$$

Where j is the system state to be estimated (position j=1) and $\hat{X}$ is the estimator used (Kalman or LSTM). The k term is the trajectory time step and refers to the CP to be checked. To validate experiments 0 and 0 we use the RMSE-2 (6.13) after the overlapping area O over new simulated trajectory with new x_0 initial conditions. The size of the time window is S and $j = 1$ is the state to be study, position.

$$\mathrm{RMSE}_2 = \sqrt{\frac{1}{S}\sum_{\mathrm{k=O+\,1}}^{S}\left(X_{k,j} - \hat{X}\left(Z_{k,j}\right)\right)^2} \tag{6.13}$$

Each experiment is compared with the output of a KF. The KF is used as a reference system to compare the proposal. In order to use the KF the measurements are simulated with Gaussian noise. This KF assumes as models a zero process noise $W_k = [0,0]^T$ for system prediction, and position measurement with Gaussian noise $\mathcal{N}(0,\sigma_Z)$ (6.8) with parameters indicated in 0. The system model corresponds to equation (6.7). KF is initialized after two consecutive measurements to determine the unmeasured state (speed) as $v_2 = (p_2 - p_1)/\Delta T$ and the covariance matrix starts like this: $P_2 = \sigma_Z \begin{pmatrix} 1 & 100 \\ 100 & 2 \end{pmatrix}$.

6.5.1. LSTM validation.

To validate our model, we use a time window in the same way as the training process but using the validation set, as shown in Fig. 6.10**.**

After applying the estimation methods (Kalman and LSTM) on each of the validation paths, two control points are used on each path of the validation set. The first is located after the activation region of each i validation series. The second CP is located on the last filtered measurement. This CP is justified based on the worst and best estimate expected from Kalman

in the optimal estimation for a linear system with continuous feed measurement (no losses) and gaussian noise. This is done after the overlap/activation region required for internal network activation. We understand activation as a transitory state of the network required to adjust its internal states.

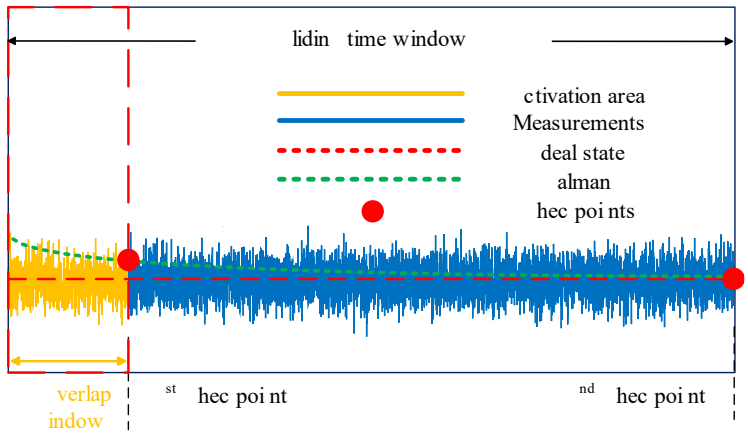


Fig. 6.10. Validation Checkpoint in sliding time window.

Fig. 6.10 shows graphically where the checkpoints are located over the noisy path in the time space of a validation measurement window.

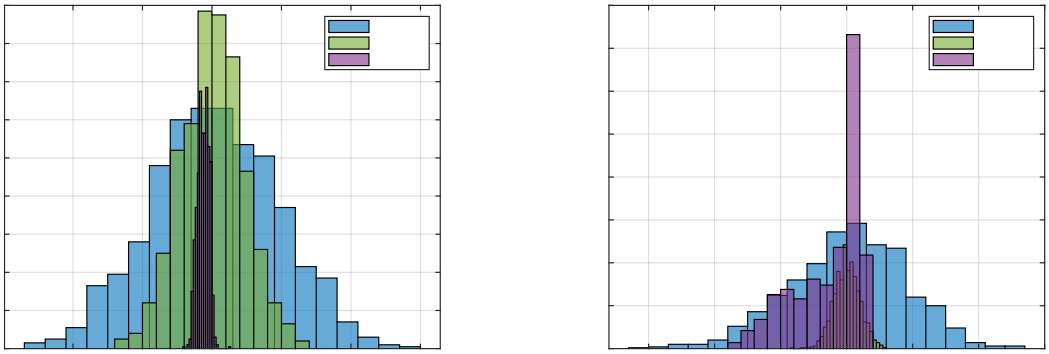
All CP is taken over trajectories in real space. In the Kalman case filtering value generation is immediate with its algorithm, on the other hand, the network must apply the unstandardization like Table *6.2* procedure 2.

The unstandardization process in our neural network proposal is produced after NN inferred process. In this step we recover the maximum (M_i) and minimum (m_i) value of each Z_i measurements series where picked up in standardization first step.

The following figures illustrate the position error histograms obtained in prediction for the first-time window of data from 1000 validation series in both checkpoints.

Fig. 6.11 (a) shows the Gaussian behavior of the measurements, Kalman, and the network at the first checkpoint. It is checked how in this first checkpoint the two systems reduce the error of the measurements and it is highlighted how the network presents better performance in this first checkpoint.

However, at the second checkpoint Fig. 6.11 (b) Kalman improves the performance of the network. In this case, while Kalman shows Gaussian behavior in its error, the network tends to stabilize its error as acquired with the training process. Also, it is verified that around 50% of the results are clustered around 0 but a group of solutions is distributed with negative values, showing an asymmetric distribution of error.



(a) (b)

Fig. 6.11. Histogram error: (a) First estimation after overlap measurements area in 1St time window of 80 measurements, (b) Last estimate in 1St time window of 80 measurements.

Table 6.4. Kalman and LSTM validation results.

Model	**Histogram RMSE [10^{-1}]**	
	1st checkpoint	2nd checkpoint
Measures | Kalman | LSTM	9.090 | 4.750 | 1.490	9.281 | 1.969 | 5.912

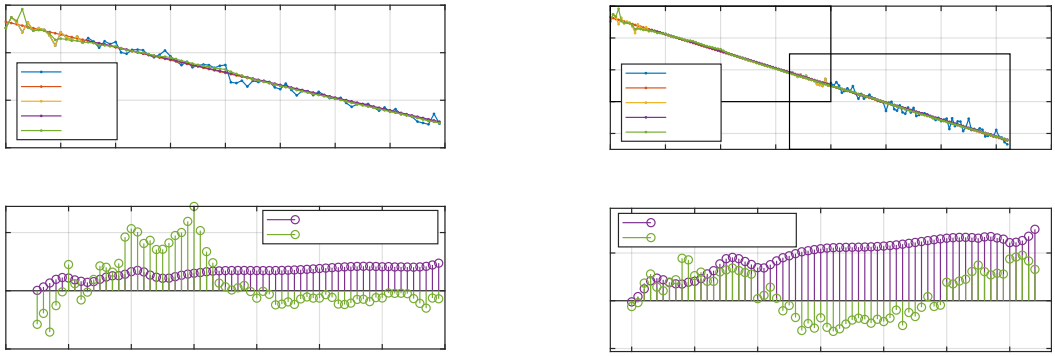
6.5.2. Filtering system simulation with new measurements

The experiments in this section and section 3 use a new path from the following initial conditions: $x0 = [-23.4897, -5.3815]$. The measurements are simulated using the parameters in Table *6.1*.

In this following case, the systems are continuously updated with new measurements. In the case of the LSTM model, the network determines internally if the measurement is relevant or not to forecast the next time step state to be forecast, while in Kalman's case this is used to reduce the filtering error. The selection process of new input values in the LSTM cells is controlled by the input gate, like we can see in [10], [17], [29]–[31]. In figure Fig. 6.12, Kalman's filter tends to minimize his error when he receives new measurements, but the LSTM model too, getting in this first phase, an improved error regarding the KF.

The first graph of Fig. 6.12 (b) shows the evolution time in the second time window of the LSTM model and the KF. While the second shows the error evolution in that time window.

In each time window, we can see the overlap/activation regions in yellow. While Kalman starts working after the initialization with the second measurements (green), the LSTM starts working after the activation area (purple). Kalman reduces his covariance exponentially as it gets new measurements, this can be seen with the error evolution in both time windows. On the other hand, it can be shown how the network error remains bounded but without a downward trend.



(a) (b)

Fig. 6.12. LSTM and Kalman with new feed measurements. (a) First, (b) Second, time-windows.

Table 6.5. RMSE with continues new measurements after overlap window.

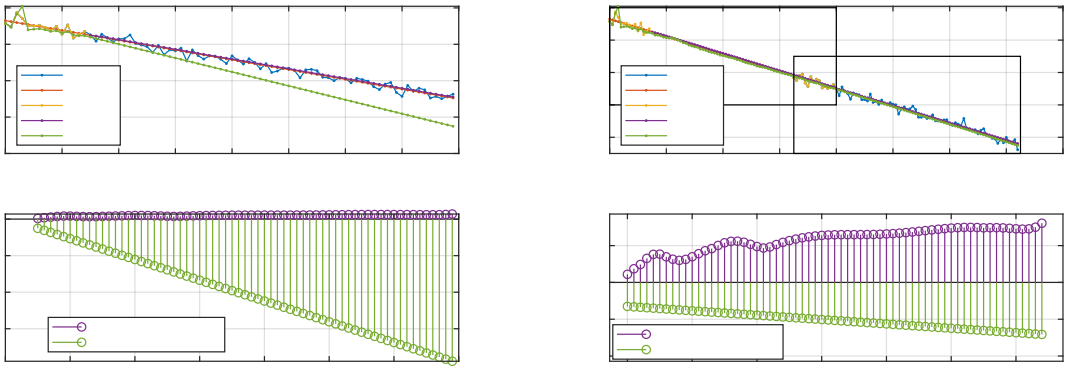
Model	**RMSE [10^{-1}]**	
	1St Window	2Nd Window
Kalman | LSTM	5.533 | 1.660	0.969 | 2.046

The RMSE analysis is shown in Table 6.5. In the first time window, can be appreciated a better performance of the network than Kalman. However as new measurements are taken, Kalman continuously improves its error while the LSTM error remains with similar values all the time.

6.5.3. Loss position measurements effect simulation

This section shows the system evolution in the first and second-time window when only one set of measurements (overlap/activation) is used to make an estimate and then it is feed with the previous estimate, both in the Kalman model and in the LSTM model. In the second time window, all the data from the first window are used to feed the KF, while only the data at the overlap region is used to activate the neural architecture. Later in both cases, we make an estimation without measurements. This process is aimed to simulate classical estimation problems when some measurements are lost.

The first window graph in Fig. 6.13 (a), shows how the KF has not enough measurements to reduce its error and it diverges from real trajectory when it does not receive new measurements, increasing its error during the prediction as a linear function of elapsed time, while LSTM architecture with few measurements manages to make good estimations and gets in that window an RMSE lower order of magnitude than Kalman. In Fig. 6.13 (b), we see how Kalman with first window data has managed to improve its behavior but will continue to increase its error with the estimates passage, while the LSTM architecture keeps its error bounded, remembering that has been activated only with overlapping window data.



(a) (b)

Fig. 6.13. LSTM and Kalman without new feed measurements. (a) First, (b) Second, time-windows.

Table 6.6. RMSE without new measurements after overlap window.

Model	**RMSE [10^{-1}]**	
	1St Window	2Nd Window
Kalman ǀ LSTM	42.534 ǀ 2.054	2.084 ǀ 2.4142

Analyzing RMSE in first and second time windows, we see how the network behaves better when measurements are lost after a few measurements, while in the second time window when Kalman has received enough measurements, Kalman improve the LSTM behavior.

6.6. Conclusions

In this paper, we implemented a neuro-estimator/filter architecture with recurrent LSTM layers and inspired by encoder-decoder systems for sequence-to-sequence learning problems able to estimate and filter a trajectory based on noisy position measurements of a uniform rectilinear motion.

We proposed a recursive model with overlapping sliding windows that allows avoiding the problem of network saturation with unbounded systems and maintains the trends from past times. To train our system, we use a panel data model standardized and pre-processed for prediction.

The model has been validated by comparing the filtering performance at two checkpoints with respect to the input sequence of measurements.

This model was compared in filtering and forecasting with a KF along with two time-windows, showing in the first one that the LSTM model improves the results in filtering and estimation with respect to Kalman, also showing evidence of bounded error in the estimation/filtering process being able to interpret internally the measurement noise.

We have verified that with few initial measurements the LSTM system manages to extract the general trend of the trajectory, while the KF with few measurements may not be able to reduce their estimation error and the system is susceptible to diverge from real trajectory in the absence of measurements to update the predicted values, Fig. 6.13 (a). The magnitude orders of errors and RMSE are equivalent throughout this study between Kalman and LSTM, but it is noticeable how the LSTM model shows a minor magnitude in the RMSE at the first estimates, Fig. 6.11 (a), Table 6.4.

It's important to also mention the fact that in the processes of unstandardization Table 6.2 for the neuronal architecture data for all experiments, we used (m) and (M) parameters obtained from the standardization of ideal trajectories X, associated with the series of measurements Z, with the aim of making a first approximation with the lowest possible error level of these neural systems. So, to a certain degree, the LSTM neural system is endowed with some additional information as compared to the Kalman model.

In conclusion, the presented LSTM model may be a good proposal for an alternative or hybridization with a KF, since KF provides the optimum solution in long time ranges and continuous measurements for a URM. In this way, our method has great potential for target tracking.

Author Contributions: Conceptualization, J.P.LL., J.G., J.M.M.; Formal Analysis, J.P.LL., J.G., J.M.M; Funding Acquisition, J.G, J.M.M.; Investigation J.P.LL.; Methodology, J.P.LL; Project Administration, J.G, J.M.M.; Resources, J.P.LL., J.G., J.M.M; Software J.P.LL.; Supervision, J.G, J.M.M.; Validation, J.P.LL., J.G, J.M.M.; Visualization, J.P.LL.; Writing-Original Draft Preparation, J.P.LL.; Writing-Review & Editing, J.P.LL., J.G, J.M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by public research projects of Spanish Ministry of Economy and Competitivity (MINECO), reference TEC2017-88048-C2-2-R and by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17), and in the context of the V PRICIT (Regional Program of Research and Technological Innovation).

Conflicts of interest: The authors declare no conflict of interest.

6.7. References

[1] S. H. Rudy, J. Nathan Kutz, and S. L. Brunton, "Deep learning of dynamics and signal-noise decomposition with time-stepping constraints," *J. Comput. Phys.*, vol. 396, pp. 483–506, 2019.

[2] H. H. Afshari, S. A. Gadsden, and S. Habibi, "Gaussian filters for parameter and state estimation: A general review of theory and recent trends," *Signal Processing*, vol. 135, no. January, pp. 218–238, 2017.

[3] T. Li and H. Fan, "A Computationally Efficient Approach to Non-cooperative Target Detection and Tracking with Almost No A-priori Information," *Oppen-access, arXiv*, 2021.

[4] D. Blacknell and H. Griffiths, "Radar Automatic Target Recognition (ATR) and Non-Cooperative Target Recognition (NCTR)," *Radar Autom. Target Recognit. Non-Cooperative Target Recognit.*, pp. 1–280, Sep. 2013.

[5] T. Li, H. Chen, S. Sun, and J. M. Corchado, "Joint Smoothing and Tracking Based on Continuous-Time Target

Trajectory Function Fitting," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 3, pp. 1476–1483, 2019.

[6] D. Schuhmacher, B. T. Vo, and B. N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE Trans. Signal Process.*, vol. 56, no. 8 I, pp. 3447–3457, 2008.

[7] B. T. Vo, C. M. See, N. Ma, and W. T. Ng, "Multi-sensor joint detection and tracking with the Bernoulli filter," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 48, no. 2, pp. 1385–1402, 2012.

[8] B. Ristic, B. T. Vo, B. N. Vo, and A. Farina, "A tutorial on Bernoulli filters: Theory, implementation and applications," *IEEE Trans. Signal Process.*, vol. 61, no. 13, pp. 3406–3430, 2013.

[9] J. Mohd Ali, M. A. Hussain, M. O. Tade, and J. Zhang, "Artificial Intelligence techniques applied as estimator in chemical process systems - A literature survey," *Expert Syst. Appl.*, vol. 42, no. 14, pp. 5915–5931, 2015.

[10] S. Hochreiter and J. Urgen Schmidhuber, "Long Shortterm Memory," *Neural Comput.*, vol. 9, no. 8, p. 17351780, 1997.

[11] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[12] C. Gan, L. Wang, Z. Zhang, and Z. Wang, "Sparse attention based separable dilated convolutional neural network for targeted sentiment analysis," *Knowledge-Based Syst.*, 2019.

[13] Y. Wang, M. Huang, L. Zhao, and X. Zhu, "Attention-based LSTM for aspect-level sentiment classification," *EMNLP 2016 - Conf. Empir. Methods Nat. Lang. Process. Proc.*, pp. 606–615, 2016.

[14] O. Arriaga, P. Plöger, and M. Valdenegro-Toro, "Image Captioning and Classification of Dangerous Situations," no. 1, 2017.

[15] C. T. C. Arsene, R. Hankins, and H. Yin, "Deep learning models for denoising ECG signals," *Eur. Signal Process. Conf.*, vol. 2019-Septe, no. Iaa 220, pp. 1–5, 2019.

[16] X. Song *et al.*, "Time-series well performance prediction based on Long Short-Term Memory (LSTM) neural network model," *J. Pet. Sci. Eng.*, p. 106682, Nov. 2019.

[17] Z. Zhao, W. Chen, X. Wu, P. C. V. Chen, and J. Liu, "LSTM network: A deep learning approach for short-term traffic forecast," *IET Image Process.*, vol. 11, no. 1, pp. 68–75, 2017.

[18] L. O. Orimoloye, M. C. Sung, T. Ma, and J. E. V. Johnson, "Comparing the effectiveness of deep feedforward neural networks and shallow architectures for predicting stock price indices," *Expert Syst. Appl.*, vol. 139, p. 112828, 2020.

[19] M. Zaheer, A. Ahmed, and A. J. Smola, "Latent LSTM allocation joint clustering and non-linear dynamic modeling of sequential data," *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 8, pp. 6040–6049, 2017.

[20] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems," pp. 1–19, 2018.

[21] X. Zheng, M. Zaheer, A. Ahmed, Y. Wang, E. P. Xing, and A. J. Smola, "State Space LSTM Models with Particle MCMC Inference," pp. 1–12, 2017.

[22] S. Kay, "Intuitive probability and random processes using MATLAB®," 2006.

[23] J. M. Wooldridge, "Econometric Analysis of Cross Section and Panel Data," 2010.

[24] S. Shapsough, R. Dhaouadi, and I. Zualkernan, "Using linear regression and back propagation neural

networks to predict performance of soiled PV modules," *Procedia Comput. Sci.*, vol. 155, no. 2018, pp. 463–470, 2019.

[25] N. E. Barabanov and D. V. Prokhorov, "Stability analysis of discrete-time recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 13, no. 2, pp. 292–303, 2002.

[26] L. Deng, M. H. Hajiesmaili, M. Chen, and H. Zeng, "Energy-Efficient Timely Transportation of Long-Haul Heavy-Duty Trucks," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 7, pp. 2099–2113, 2018.

[27] Q. Wu and H. Lin, "A novel optimal-hybrid model for daily air quality index prediction considering air pollutant factors," *Sci. Total Environ.*, 2019.

[28] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[29] X. Song, J. Huang, and D. Song, "Air Quality Prediction based on LSTM-Kalman Model," no. Itaic, pp. 695–699, 2019.

[30] Y. Xiao and Y. Yin, "Hybrid LSTM neural network for short-term traffic flow prediction," *Inf.*, vol. 10, no. 3, 2019.

[31] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, 2017.

Chapter 7: Forecasting nonlinear systems with LSTM: Analysis and comparison with EKF

Juan Pedro Llerena Caña [1]*, Jesús García Herrero [1] and José Manuel Molina López [1]

[1] Carlos III University of Madrid (Madrid-Spain), Applied Artificial Intelligence Group (GIAA); jllerena@inf.uc3m.es (J.P.LL.), jgherrer@inf.uc3m.es (J.G.) and molina@ia.uc3m.es (J.M.M.).

* Correspondence: jllerena@inf.uc3m.es

Received: date; Accepted: date; Published: date

Abstract: Certain difficulties in path forecasting and filtering problems are based in the initial hypothesis of estimation and filtering techniques. Common hypotheses include that the system can be modeled as linear, Markovian, Gaussian, or all at one time. Although, in many cases, there are strategies to tackle problems with approaches that show very good results, the associated engineering process can become highly complex, requiring a great deal of time or even becoming unapproachable. To have tools to tackle complex problems without starting from a previous hypothesis but to continue to solve classic challenges and sharpen the implementation of estimation and filtering systems is of high scientific interest. This paper addresses the forecast–filter problem from deep learning paradigms with a neural network architecture inspired by natural language processing techniques and data structure. Unlike Kalman, this proposal performs the process of prediction and filtering in the same phase, while Kalman requires two phases. We propose three different study cases of incremental conceptual difficulty. The experimentation is divided into five parts: the standardization effect in raw data, proposal validation, filtering, loss of measurements (forecasting), and, finally, robustness. The results are compared with a Kalman filter, showing that the proposal is comparable in terms of the error within the linear case, with improved performance when facing non-linear systems.

Keywords: LSTM, Filtering, Forecasting, Regression, Encoder-Decoder, Attention, System identification, Deep learning.

7.1. Introduction

Many problems in engineering and research require or are based in forecasting or filtering parameters along time, understood by forecasting the predicted values for future times in the sequence. These processes are often associated with sensor-recorded values with a certain degree of accuracy. When the noise level has been reduced from the desired parameters, this is a filtering case.

The problems of estimation and filtering are not new, a classic study field is the theory of stochastic observers. The Aström [1] and Lewis [2] books provide an introduction into stochastic estimator theory and have been referenced in thousands of publications. Classical estimation methods have innumerable successful applications and continue to be one of the starting points for estimation and filtering problems. For an overview of classical and Bayesian estimation techniques, H. H. Afshari et al.'s [3] work provides a systematic review of all classical and Bayesian estimation techniques and their possible applications.

One of the principal landmarks in stochastic observer theory is the optimal stochastic estimators formulation or Kalman filter (KF) [4]–[6]. These estimators are based in the state space systems and different versions, such as extended KF (EKF) [7]–[9], unscented KF (UKF) [10, 11], or robust KF (RKF) [12], generalize its use with nonlinear Gaussian problems as shown in Afshari et al. [3]. However, sometimes the systems can present complexities that may be unapproachable from a classical perspective. In other cases, the systems present behaviors with memory (non-Markovian), like people moving around among other people [13]. In these cases, classical solutions provide approximations that diverge from the wanted behavior.

The KF is a widely used system for filtering and state estimation. This estimator uses linear systems and Gaussian noise as starting assumptions to find a feedback gain (Kalman gain) that exponentially minimizes the system covariance. On the other hand, the systems that can be solved by Kalman or its extended version, EKF, are Markovian, in other words, for state estimation they only use contiguous states but without taking into account the behavior (states) at other times. This limits the use in problems with context, such as natural language processing or human behavior, among others.

In the face of these limitations, artificial intelligence paradigms provide an interesting opportunity to study. It is interesting how hybrids between classical and artificial intelligence systems have been achieved, such as those made by Satish. R et al. [10] or H. Caskun [14]. In [14], a neuronal estimator was fused with a KF for human image pose regularization. Works such as J. Mohd et al. [15] used the term "software sensors" to describe computational algorithms to estimate system states that are complex to measure, expensive, or non-observable. Thus, computational artificial intelligent (AI) techniques were shown to be an alternative to classical estimators in the face of certain problems. In this line we can find many works, such as those of [15, 16], in which they use several features of the input in their models.

New perspectives in machine learning techniques address several classical theories limitations problems as shown in Park's work [17]. Park modeled the potential trajectories of nearby vehicles from a grid that formed an occupation map and an encoder–decoder system based on long short-term memory (LSTM) cells. If we know the states to be estimated or modeled, we can find problems with time series estimation or systems modeling.

Time series forecasting works, to some extent, to identify/model the dynamical system that the observations describe. The LSTM cells architectures have proved their potential in front of traditional techniques, such as ARMA (AutoRegressive Moving Average), SARIMA (Seasonal Autoregressive Integrated Moving Average Model), and ARMAX (AutoRegressive-Moving Average with exogenous terms). A good example of this is Muzaffar and Afshari's work [18], where they compared the previous traditional techniques with a light LSTM architecture for the electric charge estimation case in ranges of different time sampling, under root mean squared error (RMSE) and mean absolute percentage error (MAPE) metrics, where the LSTM architecture showed better results than the traditional techniques in several experiments, and this proposed system is very susceptible to improvements to increase the performance.

Deep learning (DL) in forecasting, filtering, or classification problems attempts to fit internal network functions to an input data set to make inferences. Relying on the architecture of the neural network, the cost function, the training algorithm, hyperparameters, and especially the dataset, the network can be adapted to a greater or lesser extent to the desired output.

While Kalman seeks to minimize its covariance based on prior assumptions, a deep neural network does not assume any of Kalman's assumptions but attempts to adapt its hidden dynamics to the training data independently of their distribution or the dynamical relationship between them. This neural network flexibility provides an opportunity to generalize estimation and filtering problems under artificial intelligence paradigms.

A previous work [19] made a first approach to forecasting and filtering problems in an increasing linear dynamic system with noisy measurements from a DL perspective. In [19], the authors highlighted the neural network saturation problem in non-bounded system estimation. To solve this problem, a recursive data standardization method based on overlapping sliding windows and a neural architecture with LSTM cells is proposed.

This paper tackles the forecast-filtering problem of trajectories from deep learning paradigms. We propose a novel method of network density adjustment based on J. Llerena et al.'s work [19]. That method generalized the estimation and filtering problem without any initial hypothesis about the system or measurement type (linear or nonlinear, Markovian or non-Markovian, or Gaussian or non-Gaussian), performing a rigorous analysis of the problem and solutions with a high experimental burden to evaluate the estimator performance.

Unlike Kalman, this proposal performs the process of prediction and filtering in the same phase, while Kalman requires two phases. In this evaluation, we study three different dynamic system trajectories. We have selected a set of systems with a progressive transition for the reader, starting from the position estimation in a uniform rectilinear motion (URM) in 1D

7.4.1. (7.4.1.); next, a sinusoidal paths of a 1D object (7.4.2.); and finally the curved trajectories defined by a nonlinear dynamic model described by the Volterra–Lotka evolutionary equations (7.4.3.). The proposed neural estimator is evaluated for different cases under five experiments: data preprocessing effect on database (7.4.4.1.), filtering with complete sequences (7.4.4.2.), recursive filtering with new measurements (7.4.4.3.), loss in measurement estimation simulation (7.4.4.4.), and finally the impact on the filtering when receiving measurements far from the model (7.4.4.5.).

The neural estimators proposed are supported by an encoder–decoder system based on natural language processing methods, which increases its depth with the complexity of the systems.

Finally, the contributions of the present work can be summarized in the following items:

- An approach has been developed to adapt a neural architecture previously used for natural language processing to the specific problem of estimation and filtering without needing previous hypotheses about the type of system.
- The proposed method shows a comparable performance in terms of error with respect to KF in linear systems, while in the case of nonlinear systems it shows its potential to improve in terms of error and robustness.
- The principal advantage of our method lies in the simplicity of the neuro-estimator/filter as a model building learnt from data with respect to KF.
- The proposed method can address estimation and filtering problems for linear, nonlinear, Markovian, non-Markovian, Gaussian and non-Gaussian systems.

7.2. General problem formulation

We consider an unknown dynamic system f not necessarily linear or Markovian. From this system we only know noise measurements z of trajectories described from observable system states x in time t. Measurements z are connected with the system states by the h function. Generally, h can be considered nonlinear and dependent of a stochastic parameter $v(t)$.

$$\frac{dx(t)}{dt} = f(x(t)) \tag{7.1}$$

$$z(t) = h(x, v) = h(x) + v \tag{7.2}$$

Here, $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the state vector, f is a state vector field, and h is a function that selects a subset of specific states. If f is of Lipchitz type, it is possible to transform the continuous-time problem to a discrete-time one:

$$x_{k+1} = F^*(x_k) = x_k + \int_{t_k}^{t_{k+1}} f(x(\tau))\mathrm{d}\tau \tag{7.3}$$

A common way to discretize generally linear systems is to use the approximation $\dot{x} = \frac{x_{k+1}-x_k}{T_s}$, where T_s refers to the sampling time that we can also find as ΔT or T.

Removing the assumption of a Markovian system, the future states not only depend on the previous instant states x_k, but also have long-term temporal dependencies, and thus we can formulate it as follows:

$$x_{k+1} = F^*(x_k) = \int_{t_l}^{t_{k+1}} f(x(\tau))\mathrm{d}\tau \tag{7.4}$$

where t_l is a temporal instant less than k and generally unknown in non-Markovian systems, where the approach for the previous discretization can no longer be used. In this way classical dynamic system can be considered as a particular case of a non-Markovian system.

According to this notation, the forecasting state problem is formulated in relation to the previous states (7.4), which means that the forecasting consists of identifying states in future times (x_{k+1}). On the other hand, a filtering problem base identifies certain x_k states at the same moment in which z_k noise measurements are received (7.5).

$$x_k = h^{-1}(z_k - v_k) \tag{7.5}$$

However, in real problems, it is not possible to know the noise value, v_k, and the h function may not be invertible, so that the state vector has to be estimated from observations. If we name $\hat{\mathrm{F}}^+$ and $\hat{\mathrm{F}}$ the filtered and predicted estimators, respectively, the problem is how to generate these estimators from observations:

$$\hat{x}_k = \hat{\mathrm{F}}^+(z_0, \dots, \mathrm{z}_{\mathrm{k}-1}, \mathrm{z}_{\mathrm{k}}) \tag{7.6}$$

$$\hat{x}_{k+1} = \hat{\mathrm{F}}(z_0, \dots, z_{k-1}, z_k) \tag{7.7}$$

The objective of this process is to build the estimators with the minimum error from the ideal values.

7.2.1. Kalman solution

In Bayesian estimation theory, KF is the optimal solution for a linear dynamic system and Gaussian noise in the measurement and estimation process [1, 2]. For a stochastic nonlinear dynamic system (7.8), the first approximation derived from the KF is the EKF.

$$\begin{aligned} \dot{x} &= f(x, u, w) \\ z &= h(x, v) \end{aligned} \tag{7.8}$$

As in the linear KF [1–3], w shows the noise process and v shows the measurement noise. The system and measurement model can be nonlinear. The EKF idea is built around the linearization system over the estimated states $\hat{x}$. This means that f and h must be derived with respect to the states x, the model noise w, measurement noises v, and the input signal u. In our case, we consider an autonomous system:

$$\begin{aligned} A &= \nabla f(x,0,0)|_{(\hat{x},u,0)} \\ W &= \nabla f(0,0,w)|_{(\hat{x},u,0)} \\ H &= \nabla h(\boldsymbol{x}_k,0)|_{(\hat{x},u,0)} \\ V &= \nabla h(0,v)|_{(\hat{x},u,0)} \end{aligned} \tag{7.9}$$

The first bracket in the previous equations refers to the terms with respect to the functions derived from the system and measurements, while the second bracket refers to the values to be substituted in our Jacobian matrix.

The matrices A, W, H, and V are the equivalent to the linearized f,h system. A is the linear system matrix, H is the observation matrix, W is the process noise, and V is the observation noise, all in continuous space. If the system has an input signal u, we can find the input matrix B and the direct transmission matrix D; however, in autonomous systems, these matrices do not exist. When discretizing a linear continuous system to discrete space with a sampling time ΔT , some of the above matrices traditionally acquire another notation symbol: $A \rightarrow \phi$ and $B \longrightarrow \Gamma$.

When the continuous system has been linearized, the next step is to discretize and apply the same process as in the linear KF. This classical theory decouples, in two different phases, the problem of prediction and filtering.

Kalman filters and EKF have two steps, prediction and update. To identify these steps and the temporary state, Kalman notation uses a sub-index in the form $x_{\gamma|\delta}$. The first, γ, refers to the temporal state (current=k and previous=$k-1$) and the second, δ, refers to the filter step (prediction=$k-1$ and update=k).

The KF step formulation is formulated as follows when the system does not have noise in the estimation process and is autonomous when $\Gamma = 0$ or when the control signal $u_k = 0$.

Prediction step:

$$\begin{aligned} \hat{x}_{k|k-1} &= \phi \hat{x}_{k-1|k-1} \\ P_{k|k-1} &= \phi P_{k-1|k-1} \phi^T + Q_k \end{aligned} \tag{7.10}$$

Update step:

$$G_k = P_{k|k-1} H^T (H P_{k|k-1} H^T + R_k)^{-1} \tag{7.11}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + G_k(z_k - H\hat{x}_{k|k-1})$$
$$P_{k|k} = (I - G_k H)P_{k|k-1}$$

In this way, both problems with forecasting and filtering in Kalman are decoupled. In the Kalman case, the forecast is made on the current state k; thus, it is usually called prediction in place of forecast. First, a state space model (SSM) predicts the current time state vector $\hat{x}_{k|k-1}$ (prediction step), and then the prediction is improved $\hat{x}_{k|k}$ (current state vector in update step) with the current measure vector z_k.

The KF aim is to find a feedback gain G (optimal Kalman gain) that allows us to exponentially minimize the covariance P matrix (measure of the estimate accuracy) taking into account the covariance of the process noise Q ($W_k \sim \mathcal{N}(0, Q_k)$) and the covariance of the measured observations R ($V_k \sim \mathcal{N}(0, R_k)$), under the assumption that all noises are Gaussian, uncorrelated, and zero-mean.

7.2.2. Deep Learning Solutions

Many works related to forecasting or filtering problems can be found in the literature under system modeling, filtering/reconstruction, and prediction keywords around deep learning paradigms. In system identification we can highlight works related to the resolution of ordinary differential equations, such as that of Chen et al. [20]. Solving these equations lets us move through the state space that defines a dynamic system at the instant of time desired—in other words, predict the future states of the system or reconstruct them.

Some of the works on system modeling, such as Sierra and Santos [21], compare traditional techniques versus neural networks highlighting the relevance of using neural networks when the mathematical modeling is complex. Modeling solutions have been found that are robust to noise in the data. Rudy's [22] work proposes a new modeling paradigm that simultaneously learns the dynamics of the system and the noise estimate of the measurements in each observation, managing to separate additive noise in the observations of the states of different systems.

Artificial neural networks (ANN) for the modeling of nonlinear dynamical systems have proven to be a relevant solution. In Raissi [23], the performance of a neural system for the modeling of different nonlinear dynamic systems starting from synthetic data. The data refer to a time series describing the states of the systems under study. In this study, they used a simple neural architecture and compared the error of the predicted trajectories versus the density and depth of the neural networks, concluding that a deeper and denser network will not always show better results.

In the case of signal filtering, the Arsene work [24] showed a performance comparison in electrocardiogram (ECG) signal filtering between two deep learning filters with the two most popular trends at present, convolutional neural networks (CNN) and LSTM, versus wavelet filters. Finally, the CNN architecture achieved better performance than the LSTM and the wavelet filter, but the proposed LSTM architecture can be improved.

When the systems to be predicted show non-Markovian behavior, SSM are not suitable. A widely studied set are those related to natural language processing.

Different studies regarding natural language processing with deep learning provides exportable tools to other study areas. In relation to this work, we can remark on the encoder–decoder architectures or the attention models. Y. Zhu et al. [25] showed a novel comparative study of different LSTM encoder–decoder architectures and attention mechanisms. Finally, they proposed a combined method of an encoder–decoder with attention mechanisms and LSTM cells for prediction. They used two different datasets, from the Alibaba Open Cluster Trace Program and Dinda workload dataset. Finally, the experiments showed that their proposed model achieved state-of-the-art performance.

The common link between several of the above studies lies in the intention to extract time trends from data sets with LSTM neural cells. LSTM neural cells are not new [26], but they have proven to be powerful in catching long short temporal dependencies in multiple examples. This is the reason for its use in other than recurrent architectures, such as gated recurrent unit (GRU), bidirectional-LSTM (BI-LSTM), or bidirectional encoder representations from transformer (BERT) architectures, used with great success as a new context extraction technique in natural language processing, as shown in J. Delvin's paper [27].

The LSTM is an recurrent neural network (RNN) that allows long-term dependencies and overcomes the vanishing gradient issue [28]. Considering the relevance of this layer, detailed information of its structure can be found in works, such as those of [16], [25], [26], [29]–[32]. In X. Song [16], we can see a typical structure of a LSTM layer versus a traditional recurrent network layer. Each cell of the LSTM layer is composed by different functions as shown in Y. Liu [32]. The processes that an LSTM cell performs when it receives new data are described as follows.

Given an input x_k at time instant k and the hidden cell state h, the basic operation involves different sections of the neural cell, forget gate (7.12), input gate (7.13), candidate (7.14), and output gate (7.15). The hidden state h gives the LSTM cell the property to acquire memory, and this memory provides the opportunity to address non-Markovian problems. The forget gate f_k decides which information $c_{\mathrm{k}-1}$ is removed from the previous cell state. The input gate is responsible for identifying the input information x_k, which should be kept in the

candidate memory cell $\tilde{c}_k$. The current memory vector c_{k} is updated by linking the past information $c_{\mathrm{k}-1}$ with the candidate information $\tilde{c}_k$ (7.14). Finally, in the output gate (7.15), the hidden state h_k cell is confirmed with the cell state c_{k} and the o_k output information.

$$f_k = \sigma(x_k U_f + h_{k-1} w_f + b_f) \tag{7.12}$$

$$\begin{aligned} i_k &= \sigma(x_k U_i + h_{k-1} w_i + b_i) \\ \tilde{c}_k &= \tanh(x_k U_c + h_{k-1} w_c + b_c) \end{aligned} \tag{7.13}$$

$$c_k = f_k c_{\mathrm{k}-1} + i_k \tilde{c}_k \tag{7.14}$$

$$\begin{aligned} o_t &= \sigma(x_t U_o + h_{k-1} w_o + b_o) \\ h_t &= o_t * tanh(c_t) \end{aligned} \tag{7.15}$$

Here, U is the input weight, W is the recurrent weights and b is the bias. Subscripts represent the gates: $f = \text{forget}, \mathrm{i} = \text{input}, \mathrm{c} = \text{candidate}, \text{and } \mathrm{o} = \text{output}$. The activation function σ is the sigmoid function, and tanh is the hyperbolic tangent function. The first function is bounded between 0 and 1, and tanh between -1 and 1.

All the above cases are grouped under a regression problem in which the objective is to optimize/adjust the network function $\hat{F}_\theta$ to the $\mathbf{\Phi}$ dataset. To fit $\hat{F}_\theta$ to the $\mathbf{\Phi}$ dataset, the $\hat{F}_\theta$ function must be parameterized ($\boldsymbol{\theta}$) with a cost function $\mathcal{L}(\theta)$ to be optimized and an optimization methodology, where $\hat{F}_\theta$ means a network function parameterized by the internal $\boldsymbol{\theta}$ terms. These internal parameters are the weights and biases of each internal neuron.

As S. Rudy et al. showed in [22], we can mathematically define a recurrent neural network as the composition of g_i functions that define each i-layer of the network. In addition, these g_i functions are the result of the composition of the s_j functions that define each neuron.

$$\hat{F}_\theta(x) = \left(\prod_{i=1}^{l} C_{g_i}\right)(x) \tag{7.16}$$

Here, $g_i(x) = \left(\prod_{j=1}^{N_i} C_{s_j}\right)(x) \mid s_j = \sigma_j(xW_j + b_j)$ is an i-layer function. C_{s_j} is the s composition operator for each j activation function $\sigma_j : \mathbb{R} \to \mathbb{R}$ and $\theta = \{W_i, b_i\}_{i=1}^{l} | W_i \in \mathbb{R}^{N_i \times N_{i-1}}, b_i \in \mathbb{R}^{N_i}$ is the network parameterization function in terms of its weights W_i and biases b_i . $N_0, N_1, \ldots, N_l$ are the number of neurons in each layer, where $N_0 = d \mid d \in \mathbb{N}$ is the input layer and $l \in \mathbb{N}$ is number of network layers. The term ^ over the F function means "estimated", which is inherited from the classical notation from stochastic observers.

Taking LSTM cells in different layers, we must take into consideration the weights associated with the internal states $U_{\partial,i}$ and transitions of the LSTM cells. Finally, the parameter network functions are: $\theta = \left\{U_{\partial,i}, W_{\partial,i}, b_{\partial,i}\right\}_{i=1}^{l}$ where $W_{\partial,i} \in \mathbb{R}^{\partial \times N_i \times N_{i-1}}, b_{\partial,i} \in \mathbb{R}^{\partial \times N_i}$ and

spreading typical LSTM notation $\partial \equiv \{\text{forget} = \text{f}, \text{input} = \text{i}, \text{output} = \text{o}\ , \text{candidate} = \text{c}, \text{and non\_LSTM\_}gate = n\}$.

7.3. Proposal formulation

In this paper, we propose to approach the joint problem as forecasting-filtering trajectories without assuming a hypothesis of linear, Markovian, or Gaussian behaviors, based only on supervised information and in only one processing stage to build the estimator $\hat{x}_{k+1}$ from the available observations, $z_k, z_{k-1}, \ldots z_{k-L}$ based on a model built with representative training data.

$$x_{k+1} = \hat{F}^*(x_k) = \hat{F}^*(h^{-1}(z_k - v_k)) = \hat{F}(z_k) \tag{7.17}$$

For this purpose, the recursive method with overlapping sliding time windows with Llerena et al.'s work [19] is combined with the artificial neural architecture configuration process of Table 7.2. The general process can be seen at a high level in Table 7.1. The overlapping region between windows is used to activate the network, with activation being understood as a period for initializing the network to update its hidden states. This allows the network to activate its internal long-term memory with which to recall time trends of data from the previous time window. We have two cases of initialization, during the first-time window (no overlap window yet), lines 6–8 in Table 7.1 and, when overlaps between adjacent windows happen, lines 9–10. In the first case, as new measurements are received, they are piled up in an S-sequence until the size of the overlay/activation is defined as O. In the second case, the last measurements received in the previous time window are recycled to activate the network during the second (and successive) time windows.

The method makes it possible to address problems with continuous measurements in a recursive manner and also when a measurement is lost. If we look at the general process of Table 7.1 line 12 to 20, in the case of not receiving new measurements, the system uses the previous filtered estimation to feed the network and obtain the following state.

For this, three main blocks are differentiated: the generation of a synthetic database that allows us to control the system's performance, network building, and training, and finally inference with the trained network, like Table 7.1 shows.

The key to the generation of the synthetic database **Φ,** lies in matching noisy trajectories with ideal trajectories shifted one-time unit under Φ_i data packages. The noise paths Z_i are generated by adding a Gaussian noise with R_k variance to the simulated system states paths X_i^* to be measured. If the measured paths Z_i start at z_0 and end at z_k, the target paths X_i^* start at x_1^* and end at x_{k+1}^*, thus, maintaining the dimensionality one unit shifted. The size of the time window is therefore the L values. The length of the simulated trajectories is equal to

two consecutive non-overlapping time windows, so that the first-time window of each trajectory is used for the training subset and the second for the validation subset. Thus, the problem is formulated as a sequence-to-sequence learning system.

Table 7.1. General proposal process.

L= sliding time window length
O= overlap window length (activation area)
procedure GENERAL PROCESS (L, O, z_k)
for $k = 1 \rightarrow L$
    If start & 1st sliding window
        While Nº measurements< O
            $S_k = z_k$
        end while
    else if start
        S=$[z_{L-O}, z_{L-O+1}, \dots z_L]$
    else
        If new measure
            $S = z_k$
            $S \rightarrow$ standardization$\rightarrow S^* \rightarrow$Net & update internal states$\rightarrow \hat{x}^*_{k+1}$
            $\hat{x}^*_{k+1} \rightarrow unstandardization \rightarrow return(\hat{x}_{k+1})$
        else
            $S^* = x^*_k$
            $S^* \rightarrow$Net & update internal states$\rightarrow \hat{x}^*_{k+1}$
            $\hat{x}^*_{k+1} \rightarrow unstandardization \rightarrow return(\hat{x}_{k+1})$
        end If
    end if
end for
Move sliding window L-O & Start again
end procedure

To make step-by-step inference, a neural architecture is composed of LSTM cells. These neuronal units take advantage of their internal states as a memory to be able to relate measurements to previous and later states, allowing inferences from sequence to sequence, sequence to step, and step to step.

We assume, for this purpose, the neural network function $\hat{F}_\theta$ can be adapted to a function F that defines a dataset $\boldsymbol{\Phi}$, where $\boldsymbol{\theta}$ are the internal network parameters. The ∧ symbol over F_θ is inherited from the classical estimator's notation.

Then, the problem is to identify the parameters $\boldsymbol{\theta}$ of an ANN using exclusively supervised information, as in [19], which associates Φ_i packages of Z_i noise system paths with ideal X_i^* paths states.

$$\hat{F}_{\boldsymbol{\theta}}(z_k) \approx \mathrm{F}(z_k) \tag{7.18}$$

7.3.1. *Artificial neural network architecture*

The general network architecture proposed in Llerena et al. [19] consists of an encoder–decoder system based on good results with non-Markovian system models like [18, 23, 32]. Other fundamentals of design of this architecture focus on filtering problems, such as [24] or the identification of noisy systems [22, 23]. The encoder and decoder are composed of LSTM recursive structures. Using LSTM layers, it is possible to extract long-term and non-Markovian trends and show their potential in estimation problems [16], [33]–[35]. However, other types of dynamic systems have other particular conditions of information or number of measured states, and the architecture proposed in [19] does not have to be suitable with all systems; thereby, Table 7.2 proposes a configuration method to adapt [19]'s neural architecture to a specific case.

Starting from the structure proposed in [19], focused on the benefits in front of regression problems of each one the layers and proven performance in URM paths, we propose an algorithm in Table 7.2 to increase the depth of the encoder and decoder to adapt the results in front of other paths that are likely more complex in learning terms compared with URM paths.

Finally, at the output network side, we added a regression layer to implement the cost function $\mathcal{L}(\boldsymbol{\theta})$ (7.19) used to train the network system. Depending on the variability of the training set and the complexity of the system, the depth of the encoder–decoder and, in general, the network density must be adapted to obtain good training results.

Table 7.2. The network architecture configuration process.

SLIDING TIME WINDOW DIMENSION SELECTION
J. Llerena [19] ARCHITECTURE ADAPTION
Width = number of features
procedure ADAPT NETWORK TO SPECIFIC SYSTEM
    train $loos, MSE\ measures$
    while $RMSE\ (net) >> RMSE(data)$
        switch $Nº\ iteration$
        case 1:
            Hidden encoder and decoder layer = number of data whit sliding time window
        case 2:
            Increase number of units in the interconnexion layer.
        otherwise
            Add new LSTM layer in encoder with half hidden units than previous LSTM layer
        end switch
        go to → train
    else
        Save trained network
    end while
end procedure

7.3.2. Computational neural network framework

Under the supervised learning paradigms, we found that our problem consisted in identification systems or the regression problem. We can consider this problem as an optimization problem where we attempt to minimize the cost function $\mathcal{L}(\boldsymbol{\theta})$ by modifying the internal $\boldsymbol{\theta}$ parameters from function $\hat{F}_{\boldsymbol{\theta}}$ that we want to identify/adjust from the $\boldsymbol{\Phi}$ dataset. The typical cost function $\mathcal{L}(\boldsymbol{\theta})$ is the means square error (MSE). When we take the derivative of the MSE used in the updating the parameters during the backpropagation, the value 2 of the power can be cancelled if the term $\frac{1}{2}$ is added to the MSE. Thus, the mathematical arrangement for the definite cost function is obtained and called the half means square error (HMSE). To control for possible overfitting effects, an L_2 regularization is added to the net weights, with λ being the regularization factor.

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2S}\sum_{k=1}^{S}\sum_{j=\mathbf{1}}^{R}\left(X_{kj}^{*} - \hat{F}_{\boldsymbol{\theta}}\left(Z_{k,j}\right)\right)^2 + \lambda\sum_{i=1}^{l}\sum_{j=1}^{N_i}\left\|\theta_j\right\|_2^2 \tag{7.19}$$

S is the sequence length and R is the number of sequence parameters. On the other hand, this can be found in the literature [36]–[39], as the addition of Gaussian noise in the input data helps the regularization the network, for example with Tikhonov regularization [40]. Thus, using z-data with a certain level of $\mathcal{N}(0, \sigma^2)$ noise also helps the regularization effect in the network.

As an optimization methodology, the Adam algorithm is used, which has amply demonstrated its performance with recurrent neural architectures as can be seen in the comparison with other algorithms in Kingma and Lei's work [41].

Unlike Kalman, our system does not require Gaussian noise distribution, as the cost function does not assume any distribution. In addition, the network or cost function does not need to assume the system is linear, because the network function is fitted to the data behavior.

7.4. Case studies and experimentation

The following shows different case studies. For each one, we describe the synthetic data generation model, the classic estimator model and the neuronal structure used. All of them are accompanied by the configurations to help reproduce the results.

Among the classical estimators, KF is the optimal solution in the case of linear dynamical systems with Gaussian noise. When the system is not linear, its first approximation, EKF, is a widely extended method. To facilitate the comparison of our solution with the KF as a

reference system in the experimentation, the measurements are simulated with Gaussian noise.

For each study case, we conducted the following experiments:

1. Standardization effect.
2. LSTM model validation and filtering comparison.
3. Filtering system simulation with new measurements along the first and second-time window.
4. Simulation of missing measurements in the input to filtering system; we estimate in the first and second-time window on a signal test, applying only measurements in the overlap section, first window, and first window measurements for the second case.
5. Impact on filtering of measurements generated with parameters far from the design.

The first experiment was used to visually check that the data converted to the standardized space remained bounded. The systems were evaluated in filtering and estimation. The RMSE was used as an evaluation metric in different ways. For complete sequences, we used experiment 7.4.4.2. with (7.21) on each of the N validation trajectories over the k-time position associate with two different checkpoints. R is the number of states to be analyzed. If the system had a $R > 1$, the RMSE was determined for each of j states independently and in aggregate as the RMSE of the geometrical distance error $D_{i,k}$ (7.20). This can be seen in case study 7.4.. With partial sequences, continuous feed data, and loss data, we used experiments 7.4.4.3. and 7.4.4.4. For these cases, (7.22) was used as the evaluation metric, where L is the temporal size of the trajectories, R is the number of states, and O means the number of overlap data.

Experiment 7.4.4.5. tested the behavior of the systems in the face of new data deviating from the original design. The mean (7.24), median (7.25), and mode were used to evaluate the behavior with the RMSE (7.23) obtained from each of the N new trajectories obtained in each variation of the independent terms of the simulation systems. The mode of the ordered set E, will be the value E_i with the highest frequency in E, where $E = \{E_1 = \min_i(RMSE_3^i), E_2, \dots, E_{N-1}, E_N = \max_i (RMSE_3^i)\}_{i=1}^{N}$.

$$e_{i,k,j} = X_{i,k,j}^{*} - \hat{X}^{*}(Z_{i,k,j}); D_{i,k} = \sqrt{\sum_{j=1}^{R} e_{i,k,j}^2} \tag{7.20}$$

$$\mathrm{RMSE}_1 = \sqrt{\frac{1}{\mathrm{N}}\sum_{i=1}^{N} D_{i,k}^2}\ ;\ k = Checkpoint\ \{1,2\} \tag{7.21}$$

$$\mathrm{RMSE}_2 = \sqrt{\frac{1}{L}\sum_{\mathrm{k=O+\,1}}^{L} D_k^2} \tag{7.22}$$

$$\mathrm{RMSE}_3^i = \sqrt{\frac{1}{L}\sum_{k=1}^{L} D_{i,k}^2} \tag{7.23}$$

$$Mean = \frac{1}{\mathrm{N}}\sum_{i=1}^{N} E_i \tag{7.24}$$

$$Median = \begin{cases} E_{(N+1)/2} & if\ N\ is\ odd \\ \frac{1}{2}\left(E_{N/2} + E_{1+N/2}\right) & if\ N\ is\ even \end{cases} \tag{7.25}$$

Expression (7.20) is the estimation error and geometrical distance error, where $\hat{X}^*\left(Z_{i,k,j}\right)$ can be Kalman $\hat{X}^*{}_{i,k,j|\,k}$ or LSTM $\hat{F}_{\boldsymbol{\theta}}\left(Z_{i,k,j}\right)$, remembering that the superscript $*$ refers the sub-vector state to be estimated. The subscript i denotes trajectory i in a set of N trajectories, if the error (7.20) is calculated over a single trajectory, the term is removed as in (7.22). Finally, the subscript k is the time step, and j is the system state.

For the second experiment, we show a histogram of the estimation error of each test trajectories over the check points. If the system predicts $R > 1$ states (case study 7.4.3.), initially, this is shown as the error of each state and then the Euclidean distance between the ideal checkpoint $X^*_{i,k,j}$ and estimate system $\hat{X}^*{}_{i,k,j}$. Experiments 7.4.4.3. and 7.4.4.4. show the trajectory evolution and step-by-step error for specific initial conditions during two consecutive time windows. The error was determined for each state independently as in (7.20). Finally, experiment 7.4.4.5. shows the KF and LSTM mean, median, and mode evolution in 7.4.2. and 7.4.3. case studies as the independent terms of the trajectory simulation systems are changed.

To simulate each system trajectory, we used the Ode45 algorithm [42], while, for the estimation of states for each case study with Kalman techniques from the classical models, the formulation used is indicated in each of the systems. For training each ANN model, we trained over 80 epochs with 20 batches and an initial learning rate of 0.005. After eight epochs, we applied a 0.5 learning drop factor. Finally, we applied a $\lambda = 10^{-4}$ L_2 regularization factor.

All the algorithms were implemented on MATLAB [43]. The experiments were performed on a commodity machine with Windows 10 Home 64 bit hosted in Intel ® Core™ i7-8550U CPU @1.80 GHz 1.99 GHz with 12 GB RAM and 512 GB SSD from internal memory, graphic card Nvidia GeForce 940MX 64 bits.

7.4.1. Linear paths (Uniform Rectilinear Motion)

The model of linear paths is associated with a 1D uniform rectilinear motion, composed of the states of position p and speed v. To simulate state measurements, we only considered the position $H = [1\ 0]$ under gaussian noise $V_k \sim \mathcal{N}(0, \sigma_p)$. The simulated paths consider the ideal model, without process noise $W_k = [0, 0]^T$.

$$\begin{bmatrix} p \\ v \end{bmatrix}_k = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix}_{k-1} + W_k \tag{7.26}$$

$$z_k = Hx_k + V_k \tag{7.27}$$

The synthetic data is generated with Table III as described in Llerena's work [19].

7.4.1.1. Classical state estimator

As an estimator, we used a linear KF. In this case, the process noise is $W_k = [0, 0]^T$, and the position measurements have gaussian noise $\mathcal{N}(0, \sigma_Z)$ (7.27), as in Table 1 described in Llerena's work [19]. The system model corresponds with equation (7.26) and Table 1's parameters of [19]. KF requires two steps to obtain the unmeasurable state (speed) as $v_2 = (p_2 - p_1)/T$ and initialize the covariance matrix start, like this: $P_{2|\mathrm{k}-1} = \sigma_Z \begin{pmatrix} 1 & 100 \\ 100 & 2 \end{pmatrix}$.

$$\hat{x}_{k|k-1} = \phi \hat{x}_{k-1|k-1} \tag{7.28}$$

7.4.1.2. Artificial neural structure

As in work [19], the architecture referenced in Table 2 of that work is used. This architecture is composed of an input layer with 80 samples and one feature. The encoder has 400 hidden units, and the decoder has 200, both composed with LSTM cells. The interconnection layer between the encoder and the decoder corresponds to a fully connected layer with a rectified linear unit (ReLU) function.

7.4.2. Sinusoidal paths (Simple harmonic motion)

To generate sinusoidal paths, we considered a 1D system with simple harmonic motion that defines the transversal position with constant amplitude and frequency. The system states are given by the position x_1 and the speed x_2.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + W \tag{7.29}$$

$$z = Hx + v \tag{7.30}$$

To simulate state measurements, we only consider the first estate x_1, $H = [1\ 0]$ under gaussian noise $V_k \sim \mathcal{N}(0, \sigma_{x_1})$. The simulated paths consider the ideal model, without process noise $W_k = [0, 0]^T$.

The synthetic data is generated with Table 7.3 conditions:

Table 7.3. Synthetic data generation parameters: sinusoidal paths.

Data generation range		
Parameter	Minimum	Maximum
$\boldsymbol{x_1}$ [m]	-10	10
$\boldsymbol{x_2}$ [m/s]	-3	3
$\boldsymbol{\omega^2}$ $[\boldsymbol{rad/s}]^2$	6	
Simulation end times [s]	10.01	
Sampling time $\boldsymbol{T}$ [s]	0.01	
Number of window data	500	
Overlap $\boldsymbol{O}$ [Nº data]	90	
$\boldsymbol{V \sim \mathcal{N}(0, \sigma_Z)}$	0.4	

7.4.2.1. Classical state estimator

Starting from equations (7.29) and (7.30), using discretization (7.3) and applying Taylor's series developments, finally our linear system is discretized as follows:

$$\underbrace{\begin{bmatrix}\hat{x}_1\\ \hat{x}_2\end{bmatrix}_k}_{\hat{x}_{k|k-1}} = \underbrace{\begin{bmatrix} cos(\omega T) & \frac{sin(\omega T)}{\omega} \\ -\omega sin(\omega T) & cos(\omega T)\end{bmatrix}}_{\phi} \overbrace{\begin{bmatrix}\hat{x}_1\\ \hat{x}_2\end{bmatrix}_{k-1}}^{\hat{x}_{k-1|k-1}} \tag{7.31}$$

$$z_k = Hx_k + V_k. \tag{7.32}$$

By assuming that we only measured the first of the states, we used the linear trajectory system strategy to find the second state and be able to initialize a filter in the third measure. As the estimator minimizes the covariance in an exponential way, the cross covariances can be made large to converge quickly, and this helps the new poles of the feedback system have a high negative real part:

$$P_{2|k-1} = \sigma_Z \begin{pmatrix} 1 & 1000 \\ 1000 & 2 \end{pmatrix} \tag{7.33}$$

7.4.2.2. Artificial neural structure

Taking the method described in the process of Table 7.2, the architecture proposed for the sinusoidal paths is the one indicated in Table 7.4.

Table 7.4. Listing of neural network layer with sinusoidal paths: s=500 samples per input path.

Nr	Name and type	Activation/ prop.	Learnable	States
1	Sequence Input: 1x500	1	-	-
2	lstm_1: LSTM Hidden units: 500	State activation function: tanh Gate activation function: sigm	Input Weights: 2000x1 Recurrent Weights: 2000x500 Bias: 2000x1	Hidden States: 500x1 CellState: 500x1

3	lstm_2: LSTM Hidden units: 250	State activation function: tanh Gate activation function: sigm	Input Weights: 1000x500 Recurrent Weights: 100x250 Bias:1000x1	Hidden States: 250x1 CellState: 250x1
4	lstm_3: LSTM Hidden units: 167	State activation function: tanh Gate activation function: sigm	Input Weights: 668x250 Recurrent Weights: 668x167 Bias:668x1	Hidden States: 167x1 CellState:167x1
5	fc_1: Fully connected	100	Weights: 100x167 Bias:100x1	-
6	relu_1: ReLU	100	-	-
7	Do: Dropout 20%	100	-	-
8	lstm_4: LSTM Hidden units: 500	State activation function: tanh Gate activation function: sigm	Input Weights: 2000x100 RecurrentWeights: 2000x500 Bias:2000x1	Hidden States: 500x1 CellState:500x1
9	fc_2: Fully connected	1	Weights: 1x500 Bias: 1x1	-
10	Regression output	Loss function: HMSE	-	-

7.4.3. Smooth curved paths (Volterra–Lotka system)

The proposed model to generate smooth curved paths is the Volterra–Lotka predator–prey model. This model indicates the evolution of two species parameterized with the growth rates of the prey r_1, the success of the hunt of the predator that affects the prey a_1, the growth rate of the predator r_2, and the success of the hunt that affects predator a_2. The paths used are those defined by the union of the two states, also known as phase diagrams.

This is an autonomous system that does not require any input or external signal u and presents a great variety of smooth curved paths in the whole of its state space.

We added a process noise term to the system $W = [w_1, w_2]^T = [0,0]^T$.

$$\begin{cases} \dot{x}_1 = f_1(x,w) = r_1 x_1 - a_1 x_1 x_2 + w_1 \\ \dot{x}_2 = f_2(x,w) = a_2 x_1 x_2 - r_2 x_2 + w_2 \end{cases} \tag{7.34}$$

$$z = h(x,v) = Hx + V \tag{7.35}$$

This system has an equilibrium point in $EP = [\frac{r_2}{a_2}, \frac{r_1}{a_1}]$. Around this point, the system paths present a periodic evolution associated to a limit cycle attractor.

This study focuses on the set of initial conditions around 20% of the equilibrium point where the variety of trajectories is more pronounced.

Table 7.5. Synthetic data generation parameters: Volterra–Lotka paths.

Data generation range		
Parameter	Minimum	Maximum
State $\boldsymbol{x_1}$	$0.8\,{}^{r_2}/_{a_2}$	$1.2\,{}^{r_2}/_{a_2}$
State $\boldsymbol{x_2}$	$0.8\,{}^{r_1}/_{a_1}$	$1.2\,{}^{r_1}/_{a_1}$
$\boldsymbol{r_1, r_2, a_1}$	1	
$\boldsymbol{a_2}$	2	
Simulation end times [s]	20.05	
Sampling time $\boldsymbol{T}$ [s]	0.05	
Number of window data	200	
Overlap $\boldsymbol{O}$ [Nº data]	40	
$\boldsymbol{V \sim \mathcal{N}(0, \sigma_{Z_1}) = \mathcal{N}(0, \sigma_{Z_2})}$	0.09	

7.4.3.1. Classical state estimator

Using the approximation of (7.3), $\dot{x} = \frac{x_{k+1} - x_k}{T}$ the system is discretized as follows:

$$\begin{cases} x_{1,k+1} = x_{1,k} + \left(r_1 x_{1,k} - a_1 x_{1,k} x_{2,k} + w_{1,k}\right)T \\ x_{2,k+1} = x_{2,k} + \left(a_2 x_{1,k} x_{2,k} - r_2 x_{2,k} + w_{2,k}\right)T \end{cases} \tag{7.36}$$

$$z_k = H x_k + V_k \tag{7.37}$$

Since the system is non-linear, an EKF is formulated as an extension of the KF. In this way, the EKF is formulated with the following parameters:

$$A = \nabla f(x,0)|_{(\hat{x},0)} = \begin{pmatrix} 1 + (r_1 - a_1 x_2)T_s & -a_1 x_1 T_s \\ a_2 x_2 T_s & 1 + (a_2 x_1 - r_2)T_s \end{pmatrix}\Big|_{(\hat{x},0)}$$

$$\begin{aligned} W &= \nabla f(0,0,w)|_{(\hat{x},0)} = 0_{2x2} \\ H &= \nabla h(x_k,0)|_{(\hat{x},0)} = I_{2x2} \\ V &= \nabla h(0,v)|_{(\hat{x},0)} = V_{2x1} \end{aligned} \tag{7.38}$$

We consider the system to be fully observable in which we can simultaneously measure the two states that we consider as positions on a two-dimensional plane, known in other environments under the phase diagram name. The measurement noise corresponds to a gaussian noise $\mathcal{N}(\mu, \sigma_z)$ with mean $\mu = 0$ and variance σ_z.

$$P_{1|k-1} = \sigma_Z I_{2x2} \tag{7.39}$$

7.4.3.2. Artificial neural structure

Starting from the initial structure of the URM, the proposed structure for the Volterra–Lotka system is:

Table 7.6. Listing of neural network layer: s=200 is the number of samples per input path.

Nr	Name and type	Activation/ prop.	Learnable	States
1	Sequence Input: 2x200	2	-	-
2	lstm_1: LSTM Hidden units: 400	State activation function: tanh Gate activation function: sigm	Input Weights: 1600x2 Recurrent Weights: 1600x400 Bias:1600x1	Hidden States: 400x1 CellState:400x1
3	fc_1: Fully connected	16	Weights: 16x400 Bias:16x1	-
4	relu_1: ReLU	16	-	-
5	Do: Dropout 20%	16	-	-
6	lstm_2: LSTM Hidden units: 200	State activation function: tanh Gate activation function: sigm	Input Weights: 800x16 RecurrentWeights:800x200 Bias:800x1	Hidden States: 200x1 CellState:200x1
7	fc_2: Fully connected	2	Weights: 2x200 Bias: 2x1	-
8	Regression output	Loss function: HMSE	-	-

Although apparently the structure is similar to the URM, the density of the network is higher because it contains one more feature in the input and output layers, as well as a larger number of measurements to define the input/output layers.

7.4.4. Experimentation

In the following section, we show, in a compact way, each of the proposed experiments for the different study cases.

7.4.4.1. Standardization effect

In this section, we show the dataset information mapping before and after applying the standardization process. We used the standardization process described in [19] based on [16].

First, it is important to emphasize that the arrival spaces after the standardization are bounded, Fig. 7.1. Another perception that can be observed is that, for certain trajectories, the noise in the arrival space after the transformation can be attenuated (pronounced speeds, big amplitudes, or big closed paths) on the contrary increased (small speeds, amplitudes, and closed paths). This differentiation can be perceived by an intelligent system. These features combined with a bounded space are good hints to use ANN.

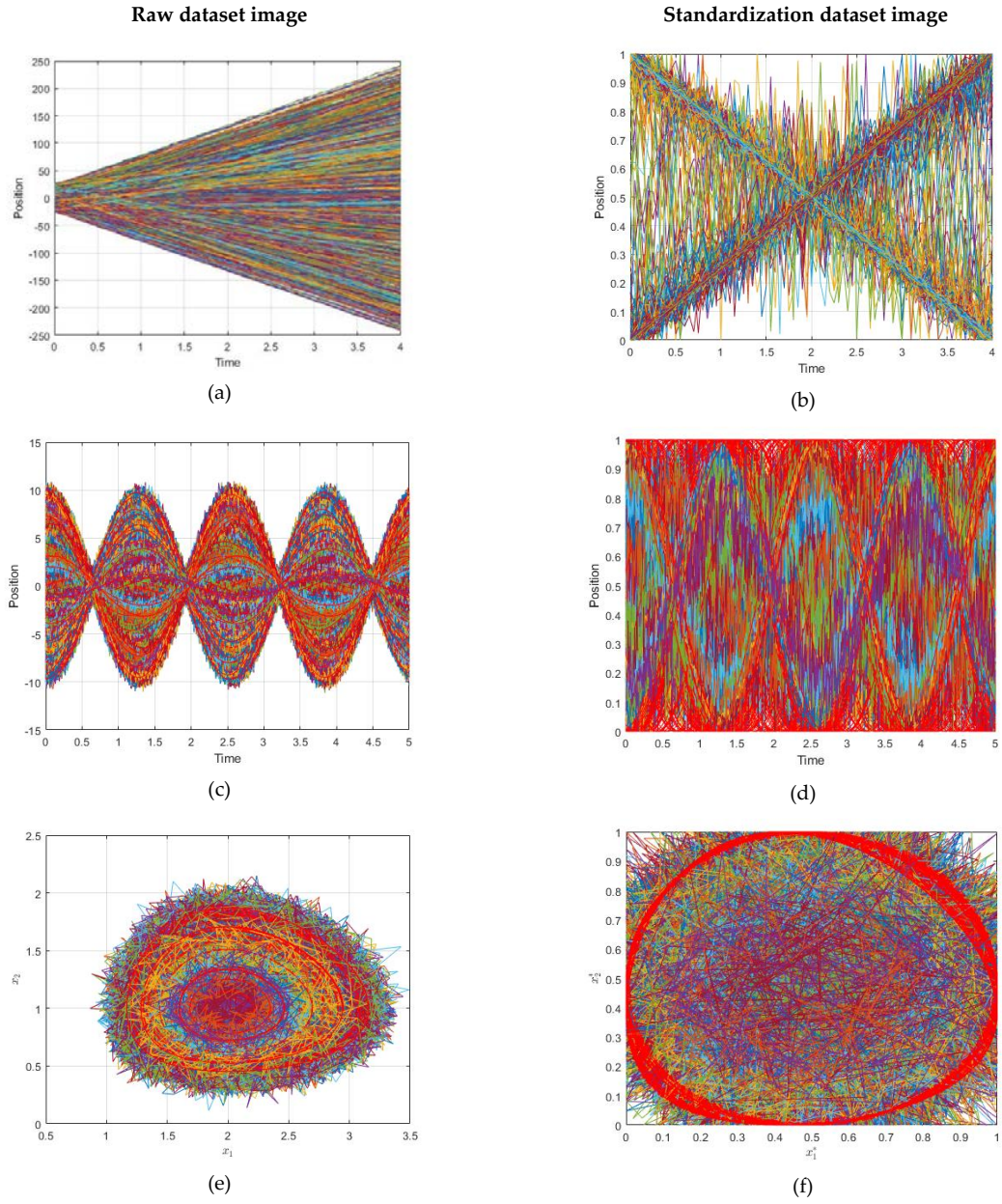


Fig. 7.1. (a), (c), and (e) a set of 10^3 ideal paths in real space with uniform rectilinear motion (URM), sinusoidal, and Volterra System. (b), (d), and (f) a set of 10^3 paths in standardized space with URM, sinusoidal, and Volterra System.

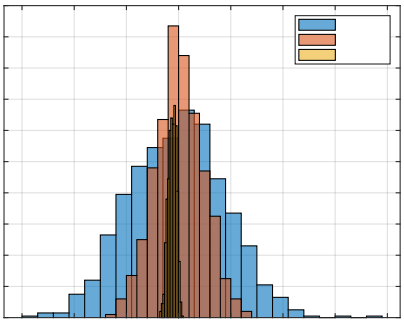
7.4.4.2. Architecture validation

The validation process of the different architectures is carried out using two checkpoints on each path. The first checkpoint is located just after the activation window and the second at the end of the data window. This is justified based on the KF covariance evolution, where it decreases exponentially in a linear system. Thus, KF will be less accurate at the beginning of receiving measurements than at the end.

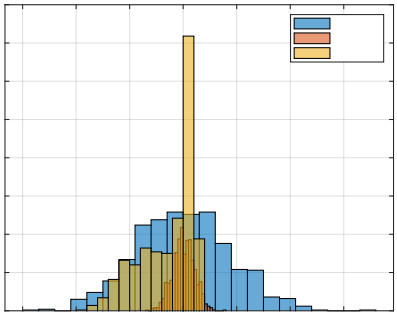
The checkpoints are taken over the measured, Kalman, and LSTM network outputs. The values obtained with each of the previous paths are compared with the ideal values, and the

error value is saved. These errors are shown as a histogram in Fig. 7.2**,** and the values of the RMSE obtained are shown in Table 7.7.

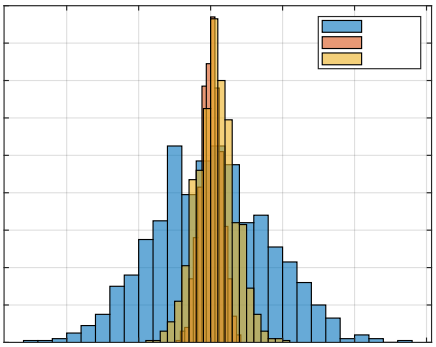
1st Checkpoint **2nd Checkpoint**



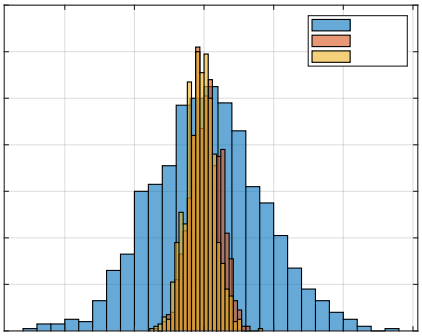
(a)



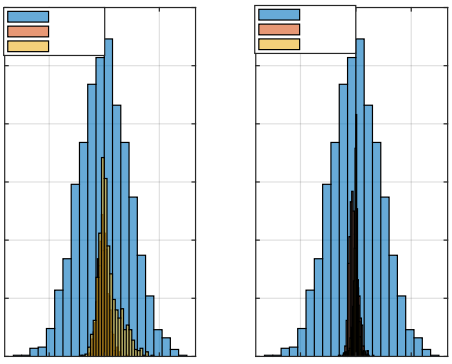
(b)



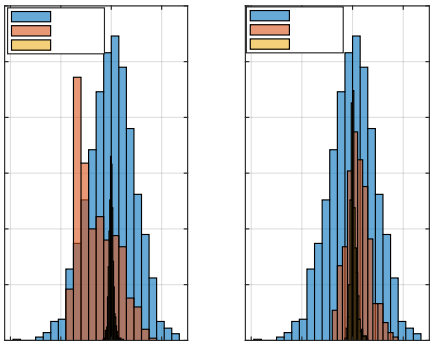
(c)



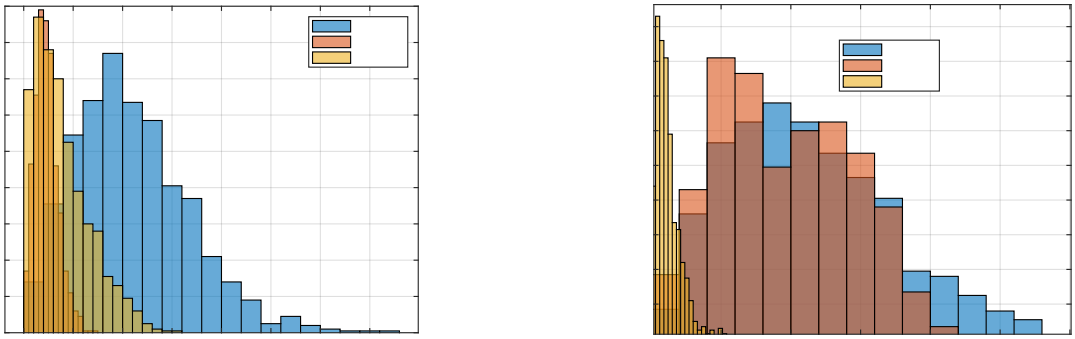
(d)



(e)



(f)



(g) (h)

Fig. 7.2. LSTM and Kalman histogram validation: (a) First and (b) second, checkpoint in the URM model. (c) First and (d) second checkpoint in the sinusoidal path model. (e) First and (f) second checkpoint in the Volterra system paths. (g) First and (h) second checkpoint in the Volterra system (Euclidian distance error).

Table 7.7. Kalman and LSTM validation results.

Path-Model	**Histogram RMSE** $[\mathbf{10^{-1}}]$ (Measurements ǀ Kalman ǀ **LSTM**)	
	First checkpoint	**Last checkpoint**
Lineal	9.086 ǀ 4.569 ǀ **1.444**	8.697 ǀ 2.038 ǀ **5.799**
Sinusoidal	3.971 ǀ 0.720 ǀ **1.395**	3.955 ǀ 1.092 ǀ **1.068**
Volterra state $\boldsymbol{x_1}$	0.893 ǀ 0.195 ǀ **0.424**	0.933 ǀ 0.948 ǀ **0.089**
Volterra state $\boldsymbol{x_2}$	0.847 ǀ 0.168 ǀ **0.107**	0.885 ǀ 0.501 ǀ **0.125**
Volterra paths (distance)	1.231 ǀ 0.258 ǀ **0.437**	1.286 ǀ 1.072 ǀ **0.153**

The error distributions of the sensor-measured data simulation show an invariant Gaussian behavior of the path position at the checkpoint. Given the nature of the RMSE, the values obtained correspond to the variance of the Gaussian noise.

We verified that the KF behavior implemented also presented a Gaussian distribution with less variance in the second checkpoint in linear systems cases (URM and sinusoidal). However, in the EKF case, we can see how the filter presents difficulties at the end of the paths but maintained the noise below the measurements.

In the case of the LSTM networks, we can see how the behavior was generally Gaussian except for the second checkpoint in the linear paths of the URM model. In the case of the second state of Volterra, it remained practically bounded, while in the sinusoidal trajectories, the first state of Volterra was reduced and was lower than in Kalman.

Fig. 7.2 (g) and (h), show the system error as a Euclidean distance of the estimated XY positions with respect to the ideal values in order to check the deviation of the filter. All distributions have a tail to the right; however, this metric allows us to highlight the amount of

data centered around the zero error. We verified how the performance of the LSTM network for this non-linear system showed great performance as the EKF approached.

Finally, we verified how the proposed system with LSTM networks reduced the noise of the measurements and presented an error comparable to the KF.

7.4.4.3. Filtering system simulation with new measurements

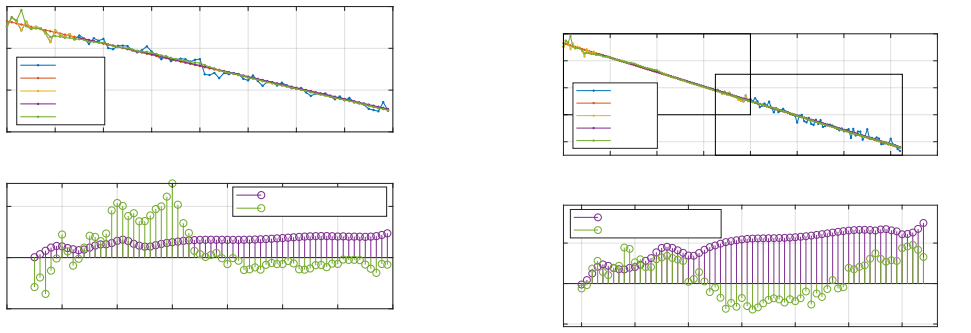
This experiment shows the behavior of Kalman and the proposed network when they are in continuous measurement feeds during the first and second time window when faced with a new set of data different from those used in the training and validation.

The initial conditions used in each system simulation are shown in Table 7.8. We used the same initial conditions for both experiments with continuous feed measurements and in the measurement experiment 7.4.4.4.

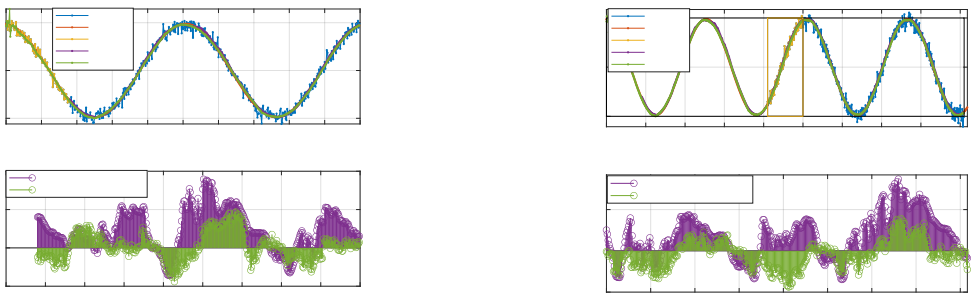
Table 7.8. The initial simulation conditions.

System-Model	Initial conditions $\bar{x}_0$
URM	$-23.4897, -5.3815$
Sinusoidal	$4.8647, -0.9199$
Volterra–Lotka	$3.0298\ , 0.8219$

First time window **Second time window**



(a) (b)



(c) (d)

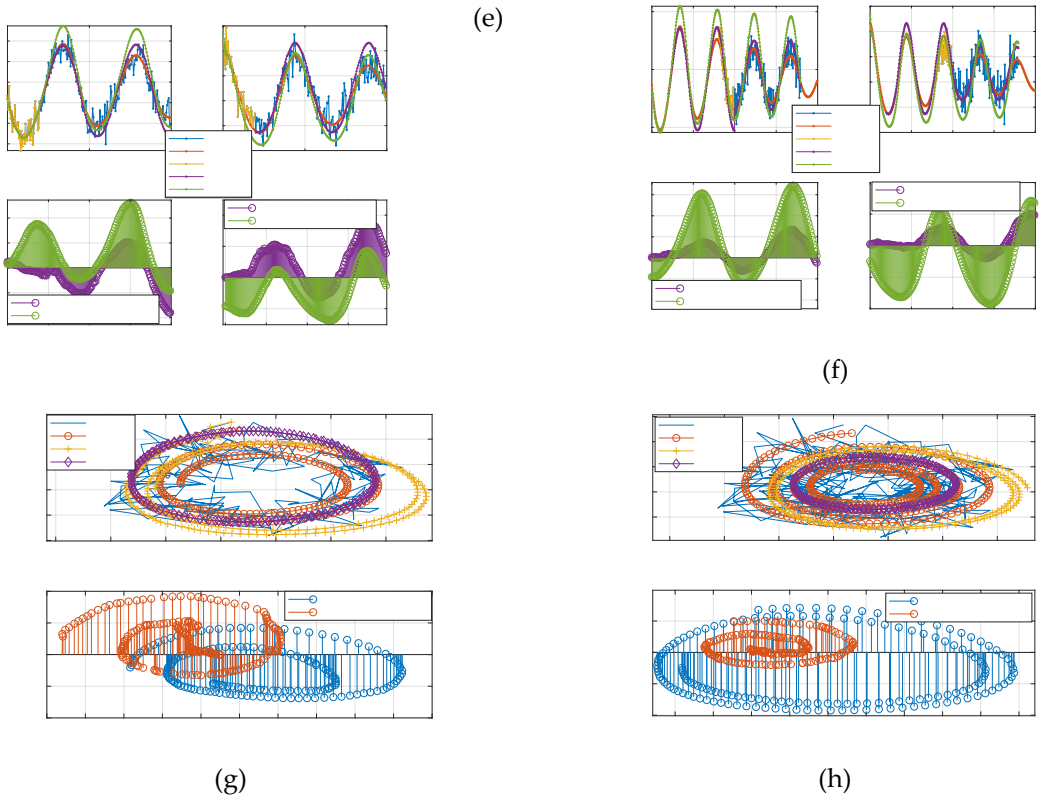


Fig. 7.3. Kalman and LSTM with new feed measurements. (a) First and (b) second time window URM path evolution. (c) First and (d) second time window sinusoidal path evolution. (e) First and (f) second, time window Volterra path evolution in both two states. (g) First and (h) second window, Volterra phase diagram evolution.

Fig. 7.3 and Fig. 7.4 in (a) to (f) show the overlapping regions in yellow—that is, the region without estimates, and is used to activate the networks and also to adjust the KF states in iterative way. After this time, the different systems were fed with new measurements to perform the filtering. In the linear case, this was checked as during the first two time windows, while the KF tended to reduce the RMSE, the network kept the error bounded to acquire the desired trend, Table 7.9.

In the sinusoidal case, we checked during the first two time windows as the KF tends to reduce its error. In the case of the neural network, it does not manage to improve on the Kalman results, but it remained with an acceptable trend and a comparable RMSE, Table 7.9.

In the case of Volterra's system, the trajectory was split into the components defined by the system states. During the first time window, the EKF and the network acquired the system trend but with a higher amplitude offset by the EKF than the LSTM, showing a behavior with less error than EKF in the initial moments but with a comparable RMSE. This effect is better observed in Fig. 7.3 (g) (phase diagram first window) where it is shown that, even maintaining a comparable RMSE, the EKF was much farther than the LSTM from the ideal values. During the second time window Fig. 7.3 (h) the effect was even more pronounced, and, this time, we

found that the LSTM had a behavior with less error than the EKF. We can see the joint states error in the error diagram of the second time window Fig. 7.3 (h), where the error in the evolution of the LSTM is shown compressed around (0,0), clearly more compact and reduced than the EKF and, in this case, an order of magnitude higher than the network.

Table 7.9. The RMSE with continuous feed measurements.

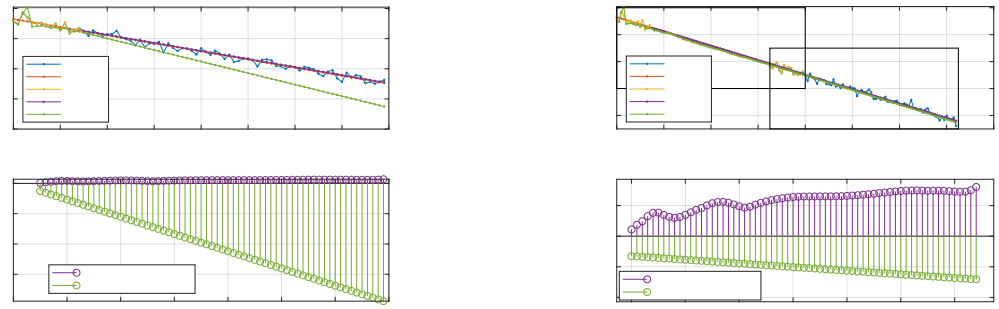
Model	**RMSE** [$\mathbf{10^{-1}}$] (Kalman | **LSTM**)	
	1St Window	**2Nd Window**
Lineal	5.533 | **1.660**	0.969 | **2.046**
Sinusoidal	1.325 | **1.444**	0.678 | **1.269**
Volterra state $\boldsymbol{x_1}$	1.118 | **0.723**	1.728 | **0.597**
Volterra state $\boldsymbol{x_2}$	0.800 | **0.861**	1.099 | **0.410**
Volterra paths (distance)	1.500 | **1.125**	2.157 | **0.725**

7.4.4.4. Effect of missing observations in the input sequence

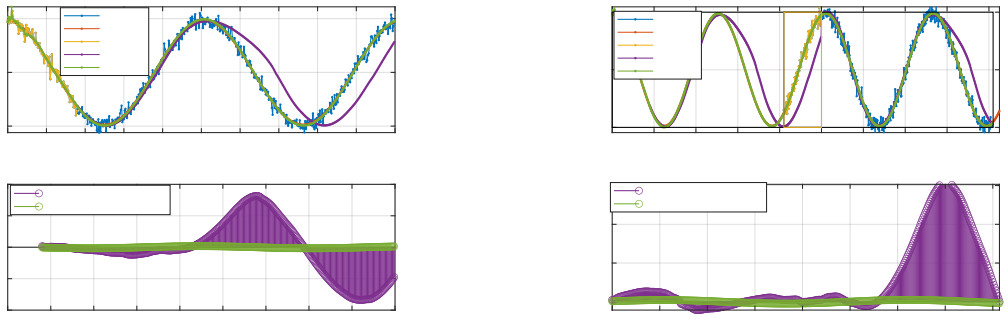
We simulated the loss of measurements after the overlap/activation region in two consecutive time windows. In the first window, we only used data from the overlap section for network activation and as feed measurements in the Kalman filters. In the second time section, KF used the set of measurements of the first-time window, while the neuronal model only used the overlapping region for the activation. When measurements are missing, the systems were fed with predictions based on the previous estimates from each system as Table 7.1 explains.

In the case of the URM system, we see how, with few measurements lost, KF can diverge from the real trajectory, while the network managed to extract the trend of the system and maintain a bounded error Fig. 7.4 (a). On the other hand, when Kalman was fed with a complete time window, it managed to extract a trend that reduced its error compared to the LSTM in terms of the RMSE. However, it may be the case that this is not sufficient and the system continues to decouple as long as the network keeps its error bounded. Fig. 7.4 (b) shows how the Kalman RMSE was lower than the LSTM but with a slightly increasing error trend indicating that it continues to decouple, while the LSTM remained bounded.

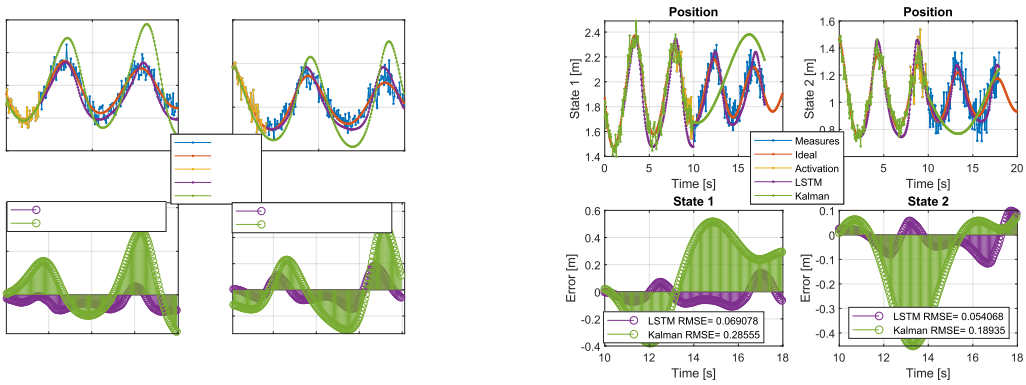
In the case of the sinusoidal paths, we verified how the well-adjusted KF managed to maintain the trends better than the LSTM during the first two-time windows. We also observed how the network managed to have a behavior like Kalman in the first estimation moments, but it decoupled in the absence of measurements and introduced a certain gap in the estimation.



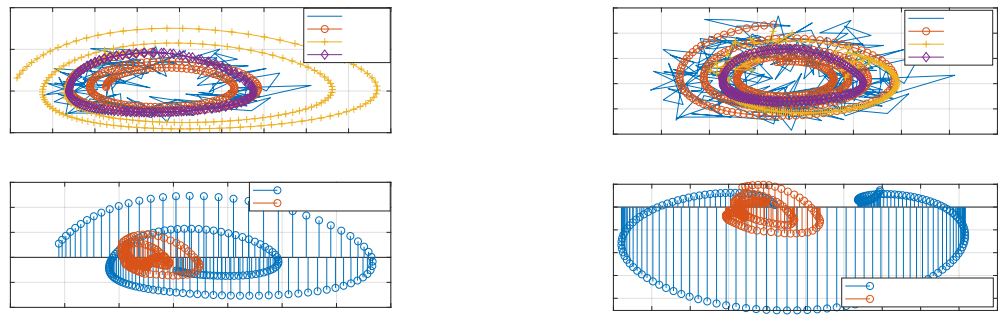
(a) (b)



(c) (d)



(e) (f)



(g) (h)

Fig. 7.4. Kalman and LSTM without feed new measurements. (a) First and (b) second time window URM path evolution. (c) First and (d) second, time window sinusoidal path evolution. (e) First and (f) second, time window Volterra path evolution in both states. (g) First and (h) second window, Volterra phase diagram evolution.

Finally, in the case of Volterra system, it can be seen how the EKF in the first and second time windows is much more vulnerable and can diverge from the ideal trajectory with respect to the proposed LSTM solution. This is easily observed in each state graphs in Fig. 7.4 (g) and (h), especially in the joint state diagrams in error part, where the error of the LSTM is clearly bounded around (0,0) while the EKF is not. Fig. 7.4 (g) and (h) show that the EKF was more vulnerable to decoupling in the absence of measurements compared with the neuronal system as observed in the evolution of systems in terms of the amplitude, phase, and finally higher error.

Fig. 7.4 (e) and (f) show that the EKF was more vulnerable to decoupling in the absence of measurements compared with the neuronal system, as observed in the evolution of systems in terms of the amplitude, phase, and definitely higher error. Fig. 7.4 (g) shows how in the first moments around (1.5 ,1) the EKF, the network, and the ideal measurements evolved together, while the neuronal network extracted the tendency of the equilibrium point and presented an evolutionary behavior on an invariant set, the EKF began to diverge from the limit cycle decoupling itself from the system and becoming unstable in terms of tendency and comparison with the ideal system.

Table 7.10. The RMSE with measurement loss simulation.

Model	**RMSE** [$\mathbf{10^{-1}}$] (Kalman ǀ **LSTM**)	
	1St Window	**2Nd Window**
Lineal	42.534 ǀ **2.054**	2.084 ǀ **2.414**
Sinusoidal	0.903 ǀ**17.455**	0.323 ǀ **10.898**
Volterra state $\boldsymbol{x_1}$	2.855 ǀ **1.188**	2.855 ǀ **0.690**
Volterra state $\boldsymbol{x_2}$	2.034 ǀ **0.817**	1.893 ǀ **0.541**
Volterra paths (distance)	3.901 ǀ **1.442**	3.803 ǀ **0.877**

7.4.4.5. Impact on filtering of measurements simulated with different parameters with respect to the design

To perform these experiments, we used an ideal model for training and to configure the KF, but we generated new paths with slight changes in the dynamic simulation model with respect to the ideal model.

This α variation was made over each constant's parameters ψ_i of the ideal model, between 5% and 200% of the ideal value. The variation was made with only one parameter to study their impact without changing the rest of the terms with the initial/ideal model. Finally, the new constant ψ_i^* is as equation (7.40), where i indicates the different constants in the dynamic model and j indicates the variation range.

$$\psi_i^* = \psi_i . \alpha_j \tag{7.40}$$

For this test, the mean value, the median, and the mode of the set of RMSE values were determined over 1000 new test paths generated over each modification of the constant parameters. This means that, when making 40 modifications, we finally generate 40.000 new paths per study case.

This test was performed on the sinusoidal case by modifying the system frequency and with the Volterra system for each of the four constant terms (7.34).

In Fig. 7.5 and Fig. 7.6, there are two essential regions in each of the graphs delimited by the variance of the measurements (blue lines). Over this border, the filtering was worse than the measurements; however, this could be due to missing measurements, and so it is interesting to study the evolution over the border of measurements and compare the differences between the classical system and the proposed LSTM system.

Sinusoidal system:

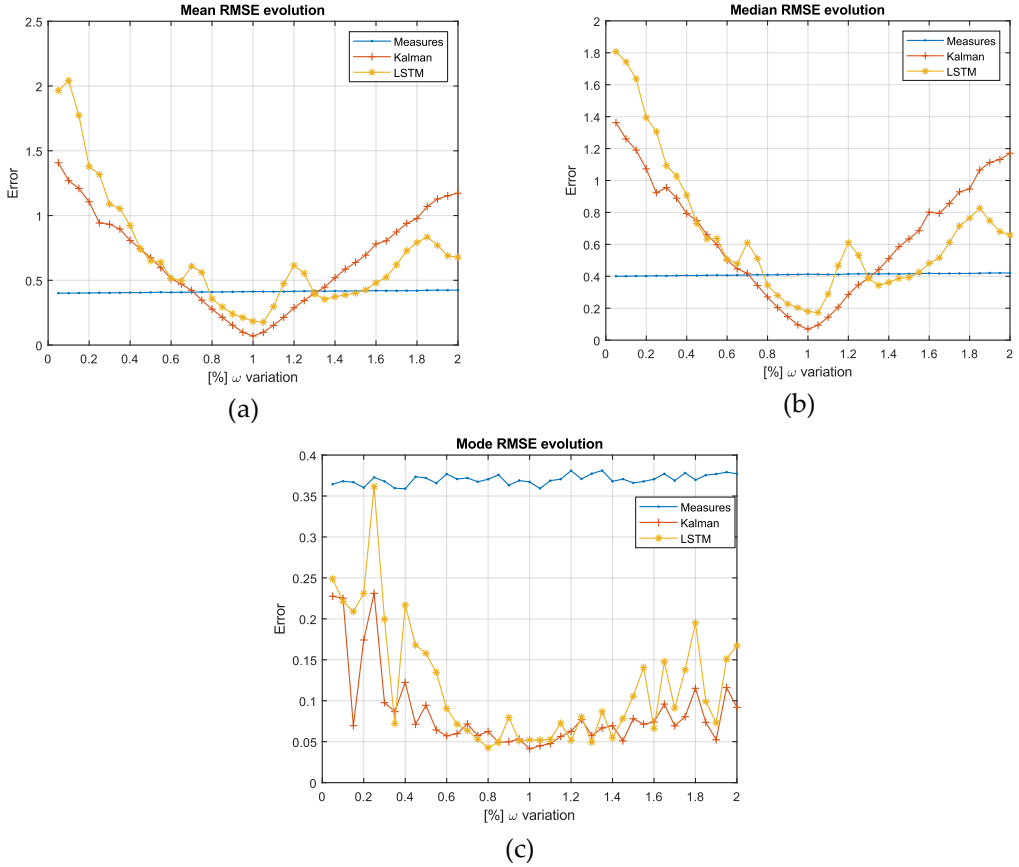
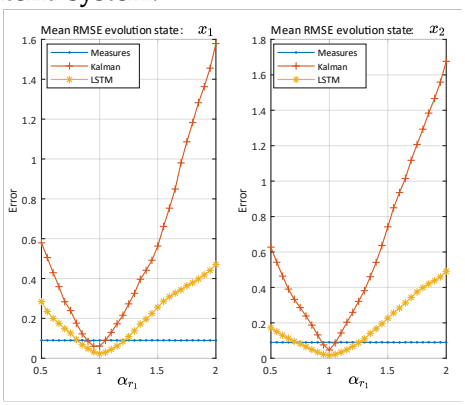


Fig. 7.5. RMSE evolution as the independent term changed in the sinusoidal measurements model: (a) RMSE mean, (b) RMSE median, and (c) RMSE mode.

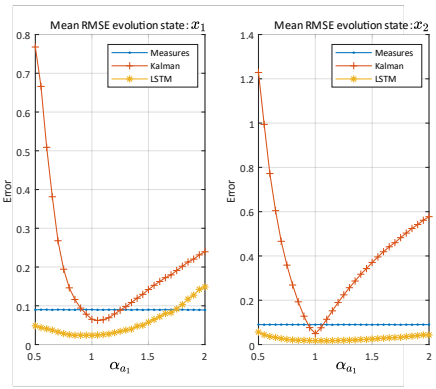
In the sinusoidal case, $\omega^2 = \psi$ was considered as the constant term. The general RMSE evolution in the average and median KF showed a linear-symmetric growth, while the network showed an irregular behavior, but with an increasing trend on both sides of α=1. In the lower

region of the measure's variance, Kalman had a lower value than the LSTM, reaching the border after the LSTM in both sides of the optimum. However, we found a region in the range of [1.25, 1.5] in Fig. 7.5 (a) and (b) in which the network continued filtering while Kalman did not. To the right of this region, Kalman performed worse than the network. In terms of the RMSE frequency (mode), we can see how both systems for the set of ranges studied were maintained in the filtering region and Kalman generally showed the best performance Fig. 7.5 (c).

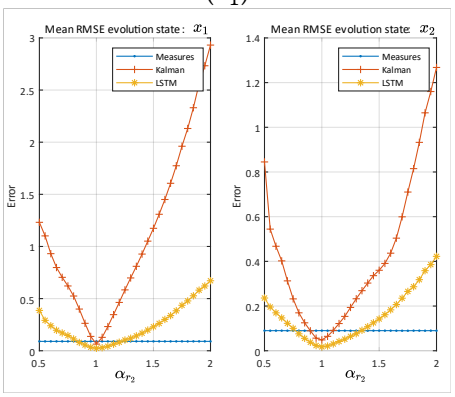
Volterra system:



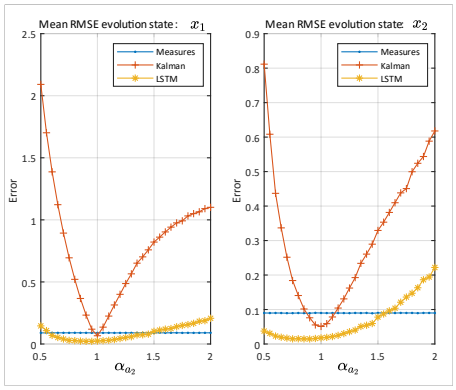
(a_1)



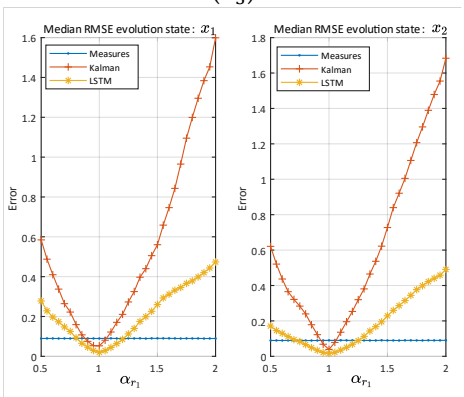
(a_2)



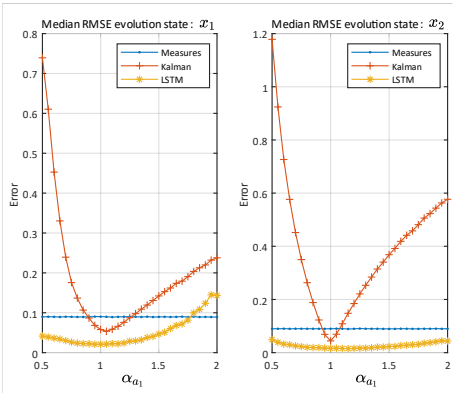
(a_3)



(a_4)



(b_1)



(b_2)

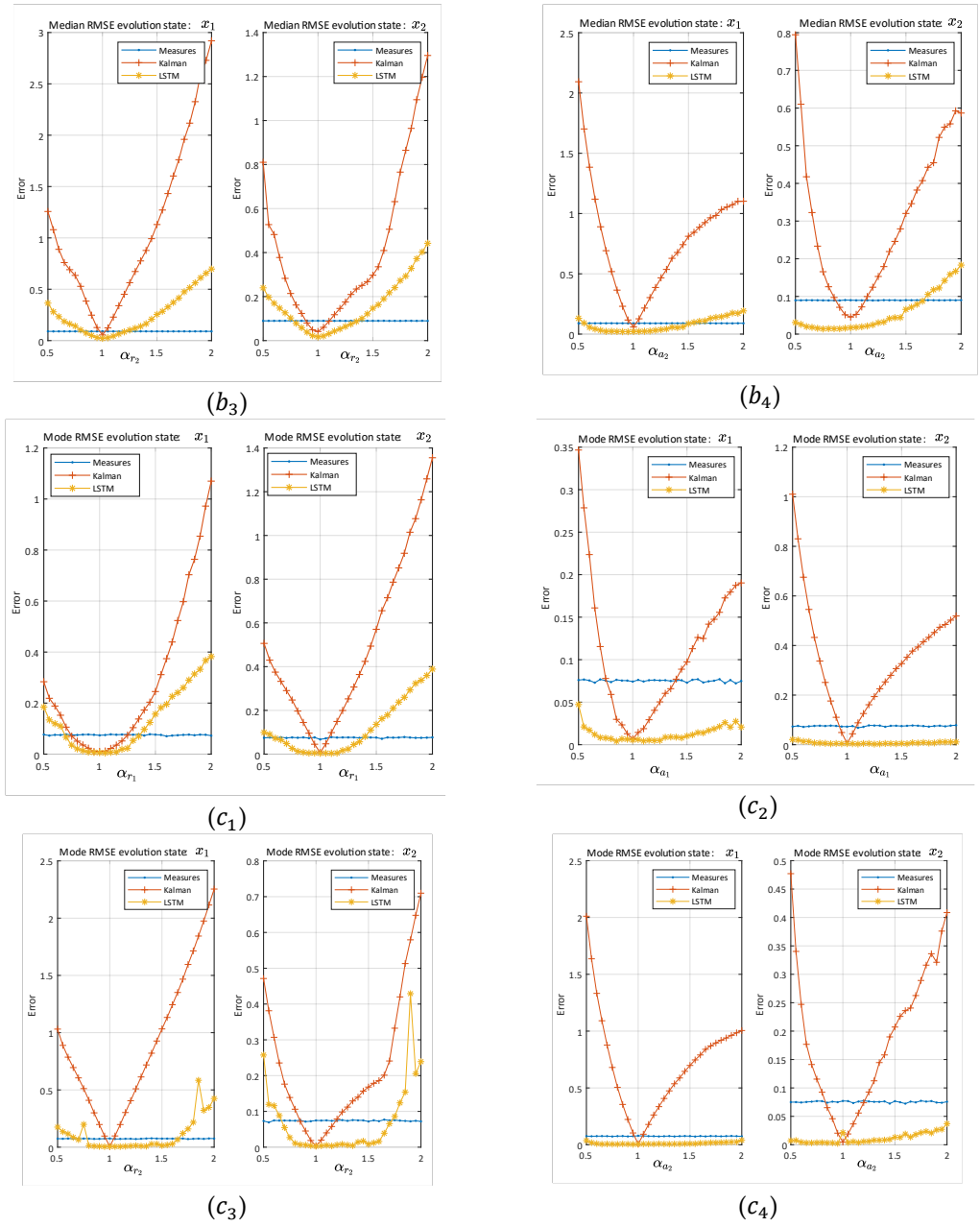


Fig. 7.6. The RMSE evolution as the independent term changed in Volterra model: (a) RMSE mean, (b) RMSE median, and (c) RMSE mode. Subscripts indicate Volterra constant terms: [1,2,3,4] = $[r_1, a_1, r_2, a_2]$.

Based on the statistical values of the mean and median RMSE with Volterra's system trajectory, the EKF sensitivity to changes in the independent terms are shown in Fig. 7.6 (a) and (b). The EKF quickly left the filtering region and showed an increasing trend on both sides of the optimum ($\alpha = 1$). On the other hand, the LSTM architecture was much less sensitive to these changes, becoming practically invariant in the second state (x_2) to a_1 modifications. The previous trend was generalized for all terms. The mode of the RMSE in Fig. 7.6 (c) showed the same behavior emphasizing the difference between the EKF and the network with the a_2

constant term modifications, where the network with even a slight increasing trend in the edges did not achieve, in the study range, the filtering border.

7.5. Conclusions

In this work, three neuro-estimator/filters were implemented through a common but different density encoder–decoder architecture, based on recurrent LSTM cells and using the Table 7.2 design process. These models were compared with a KF adapted to each specific case obtaining similar results in terms of the RMSE but, unlike Kalman, working in only one processing stage. The Kalman algorithm consists of two main processing stages, namely prediction and update, using ad-hoc models, while the proposed solution works in a single stage applying the model built after the training stage.

The study was limited with two consecutive time windows for two linear systems with linear and sinusoidal paths in a one-dimensional path space. In addition, it included a nonlinear autonomous system defined by Volterra–Lotka's equations, which describes a set of smooth, curved paths in a two-dimensional space. The simulated measurements were made by adding a Gaussian additive term in the state of the system case.

KF has proven to be the optimal process for linear systems; however, the proposed neural architectures, without taking any assumptions as Gaussian, linear, or Markovian processes, managed to show a comparable performance in terms of RMSE Table 7.7. Although it has been justified why our proposed system does not initially assume Gaussian systems or measurements (7.3.1), the system has not been tested with other noises to be compared with a reference system, such as KF or EKF. We verified that the system proposed in the case of linear trajectories, with few measurements, managed to acquire the desired trend in front of possible decoupling of the KF in absence of the measurements in Fig. 7.4 (a) and (b). When the system had non-linearity, the approaches used in the EKF may diverge from the ideal solution. The neural proposed system managed to improve the behavior of the EKF both in the filtering and in estimation in the absence of measurements Fig. 7.4 (e)-(h).

One of the principal advantages of our method lies in the simplicity of modeling the neuro-estimator/filter as KF. Finally, we studied the system behavior in the face of separate trajectories from the models for which the systems had been designed. To do this, we generated new paths modifying each constant term ψ_i of the dynamic models by a multiplicative value α. As expected, in all cases, the optimal value was found when the independent term matched between the model and generated values—that is, the multiplicative value $\alpha = 1$.

We proved, as in the case of a linear system (sinusoidal paths), Kalman grew linearly out of the filtering region after the neuronal system. The irregularity of the growth for the

neuronal system proposed for sinusoidal paths was shown to exist in regions where Kalman does not work while the network does (understanding by that "work" refers to the filter process).

As far as Volterra's system is concerned, the influence of each of its four independent terms (r_1, a_1, r_2, a_2) on EKF systems and the proposed LSTM solution were verified. We checked how the LSTM architecture can be maintained in the filtering area with a higher variation range than Kalman when each one of the independent terms is modified. In the case of a_1 and a_2, our system remained practically invariant as shown in Fig. 7.6 (a_2)-(b_2), (a_4)-(b_4)-second state x_2. On the other hand, the EKF with its linear approximations quickly left the filter region in Fig. 7.6. We can affirm that, for all the cases regarding parameter modification on the Volterra system and in the study domain as a whole, the LSTM solution was more robust than the EKF, with the filtering border beyond the EKF or even not having that border in certain cases.

Author Contributions: Conceptualization, J.P.LL., J.G., and J.M.M.; Formal Analysis, J.P.LL., J.G., and J.M.M; Funding Acquisition, J.G. and J.M.M.; Investigation J.P.LL.; Methodology, J.P.LL; Project Administration, J.G, J.M.M.; Resources, J.P.LL., J.G., and J.M.M; Software J.P.LL.; Supervision, J.G. and J.M.M.; Validation, J.P.LL., J.G., and J.M.M.; Visualization, J.P.LL.; Writing—Original Draft Preparation, J.P.LL.; Writing—Review and Editing, J.P.LL., J.G., and J.M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by public research projects of Spanish Ministry of Economy and Competitivity (MINECO), reference TEC2017-88048-C2-2-R.

Conflicts of Interest: The authors declare no conflict of interest.

7.6. References

[1] K. Åström and B. Wittenmark, "Computer-controlled systems: theory and design," 2013.

[2] F. Lewis, D. Vrabie, and V. Syrmos, "Optimal control," 2012.

[3] H. H. Afshari, S. A. Gadsden, and S. Habibi, "Gaussian filters for parameter and state estimation: A general review of theory and recent trends," Signal Processing, vol. 135, no. January, pp. 218–238, 2017.

[4] H. Musoff and P. Zarchan, Fundamentals of Kalman Filtering: A Practical Approach, Third Edition. American Institute of Aeronautics and Astronautics, 2009.

[5] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," In Pract., vol. 7, no. 1, pp. 1–16, 2006.

[6] K. Bogdanski and M. C. Best, "Kalman and particle filtering methods for full vehicle and tyre identification," Veh. Syst. Dyn., vol. 56, no. 5, pp. 769–790, 2018.

[7] J. H. Lee and N. L. Ricker, "Extended kalman filter based nonlinear model predictive control," Am. Control Conf., vol. 1, no. 9, pp. 1895–1899, 1993.

[8] J. García, J. M. Molina, and J. Trincado, "Real evaluation for designing sensor fusion in UAV platforms," Inf. Fusion, vol. 63, no. August 2018, pp. 136–152, 2020.

[9] R. Huang, S. C. Patwardhan, and L. T. Biegler, "Robust stability of nonlinear model predictive control based

on extended Kalman filter," J. Process Control, vol. 22, no. 1, pp. 82–89, 2012.

[10] S. R. Jondhale and R. S. Deshpande, "Kalman Filtering Framework-Based Real Time Target Tracking in Wireless Sensor Networks Using Generalized Regression Neural Networks," IEEE Sens. J., vol. 19, no. 1, pp. 224–233, 2019.

[11] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," IEEE 2000 Adapt. Syst. Signal Process. Commun. Control Symp. AS-SPCC 2000, pp. 153–158, 2000.

[12] Y. C. Tsai and Y. D. Lyuu, "A new robust Kalman filter for filtering the microstructure noise," Commun. Stat. - Theory Methods, vol. 46, no. 10, pp. 4961–4976, 2017.

[13] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces."

[14] H. Coskun, F. Achilles, R. Dipietro, N. Navab, and F. Tombari, "Long Short-Term Memory Kalman Filters: Recurrent Neural Estimators for Pose Regularization," Proc. IEEE Int. Conf. Comput. Vis., vol. 2017-Octob, pp. 5525–5533, 2017.

[15] J. Mohd Ali, M. A. Hussain, M. O. Tade, and J. Zhang, "Artificial Intelligence techniques applied as estimator in chemical process systems - A literature survey," Expert Syst. Appl., vol. 42, no. 14, pp. 5915–5931, 2015.

[16] X. Song et al., "Time-series well performance prediction based on Long Short-Term Memory (LSTM) neural network model," J. Pet. Sci. Eng., p. 106682, Nov. 2019.

[17] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, "Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture," IEEE Intell. Veh. Symp. Proc., vol. 2018-June, no. Iv, pp. 1672–1678, 2018.

[18] S. Muzaffar and A. Afshari, "Short-term load forecasts using LSTM networks," Energy Procedia, vol. 158, pp. 2922–2927, 2019.

[19] J. P. Llerena, J. García, and J. M. Molina, "An Approach to Forecasting and Filtering Noise in Dynamic Systems Using LSTM Architectures," Adv. Intell. Syst. Comput., vol. 1268 AISC, pp. 155–165, 2021.

[20] M. A. Hjortsø and P. Wolenski, "Some Ordinary Differential Equations," Linear Math. Model. Chem. Eng., no. NeurIPS, pp. 123–145, 2018.

[21] J. E. Sierra and M. Santos, "Modelling engineering systems using analytical and neural techniques: Hybridization," Neurocomputing, vol. 271, pp. 70–83, 2018.

[22] S. H. Rudy, J. Nathan Kutz, and S. L. Brunton, "Deep learning of dynamics and signal-noise decomposition with time-stepping constraints," J. Comput. Phys., vol. 396, pp. 483–506, 2019.

[23] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems," pp. 1–19, 2018.

[24] C. T. C. Arsene, R. Hankins, and H. Yin, "Deep learning models for denoising ECG signals," Eur. Signal Process. Conf., vol. 2019-Septe, no. Iaa 220, pp. 1–5, 2019.

[25] Y. Zhu, W. Zhang, Y. Chen, and H. Gao, "A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment," 2019.

[26] [26] S. Hochreiter and J. Urgen Schmidhuber, "Long Shortterm Memory," Neural Comput., vol. 9, no. 8, p. 17351780, 1997.

[27] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf., vol. 1, no. Mlm, pp. 4171–4186, 2019.

[28] S. Hochreiter, “Long Short-Term Memory,” vol. 1780, pp. 1735–1780, 1997.

[29] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A Search Space Odyssey,” IEEE Trans. Neural Networks Learn. Syst., vol. 28, no. 10, pp. 2222–2232, 2017.

[30] Z. Zhao, W. Chen, X. Wu, P. C. V. Chen, and J. Liu, “LSTM network: A deep learning approach for short-term traffic forecast,” IET Image Process., vol. 11, no. 1, pp. 68–75, 2017.

[31] Y. Wang, M. Huang, L. Zhao, and X. Zhu, “Attention-based LSTM for aspect-level sentiment classification,” EMNLP 2016 - Conf. Empir. Methods Nat. Lang. Process. Proc., pp. 606–615, 2016.

[32] Y. Liu, “Novel volatility forecasting using deep learning–Long Short Term Memory Recurrent Neural Networks,” Expert Syst. Appl., 2019.

[33] L. Deng, M. H. Hajiesmaili, M. Chen, and H. Zeng, “Energy-Efficient Timely Transportation of Long-Haul Heavy-Duty Trucks,” IEEE Trans. Intell. Transp. Syst., vol. 19, no. 7, pp. 2099–2113, 2018.

[34] Q. Wu and H. Lin, “A novel optimal-hybrid model for daily air quality index prediction considering air pollutant factors,” Sci. Total Environ., 2019.

[35] H. C. Ravichandar, A. Kumar, A. P. Dani, and K. R. Pattipati, “Learning and predicting sequential tasks using recurrent neural networks and multiple model filtering,” in AAAI Fall Symposium - Technical Report, 2016, vol. FS-16-01-FS-16-05, pp. 331–337.

[36] A. Graves, “Supervised Sequence Labelling,” 2012, pp. 5–13.

[37] G. An, “The Effects of Adding Noise during Backpropagation Training on a Generalization Performance,” Neural Comput., vol. 8, no. 3, pp. 643–674, Apr. 1996.

[38] A. Neelakantan et al., “Adding Gradient Noise Improves Learning for Very Deep Networks,” Nov. 2015.

[39] A. Graves, A. R. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2013, pp. 6645–6649.

[40] C. M. Bishop, “Training with Noise is Equivalent to Tikhonov Regularization,” Neural Comput., vol. 7, no. 1, pp. 108–116, Jan. 1995.

[41] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc., pp. 1–15, 2015.

[42] L. F. Shampine and M. W. Reichelt, “THE MATLAB ODE SUITE.”

[43] M. H. Beale, M. T. Hagan, and H. B. Demuth, “Neural Network Toolbox TM User ’ s Guide R2013b,” Mathworks Inc, 2013.

Chapter 8: LSTM vs CNN in real ship trajectory classification

Juan Pedro Llerena[1[0000-0002-3476-6261]] (✉), Jesús García[1[0000-0003-1768-2688]] (✉) and José Manuel Molina[1[0000-0002-7484-7357]] (✉)

[1] GIAA Group, Universidad Carlos III de Madrid, Madrid 28270, Spain.
{jllerena, jgherrer}@inf.uc3m.es, molina@ia.uc3m.es

Abstract. Ship type identification in a maritime context can be critical to the authorities to control the activities being carried out. Although Automatic Identification Systems has been mandatory for certain vessels if a vessel does not have them voluntarily or not, it can lead to a whole set of problems, so the use of tracking alternatives such as radar is fully complementary for a vessel monitoring systems. However, radars provide positions, but not what they are detecting. Having systems capable of adding categorical information to radar detections of vessels makes it possible to increase control of the activities being carried out, improve safety in maritime traffic, and optimize on-site inspection resources on the part of the authorities. This paper addresses the binary classification problem (fishing ships versus all other vessels) using unbalanced data from real vessel trajectories. It is performed from a Deep Learning approach comparing two of the main trends, Convolutional Neural Networks and Long Short-Term Memory. In this paper, it is proposed the weighted Cross-Entropy methodology and compared with classical data balancing strategies. Both networks show high performance when applying weighted Cross-Entropy compared to the classical machine learning approaches and classical balancing techniques. This work is shown to be a novel approach to the international problem of identifying fishing ships without context.

Keywords: Deep Learning, LSTM, CNN, Weighted Cross Entropy, Automatic Identification System, Fishing ship classification.

8.1. Introduction

Although the problem of Illegal, Unreported and Unregaled Fishing (IUU-Fishing) is not new [1], it continues to be a topical issue in international relationships [2, 3].

As the Food and Agriculture Organization of the United Nations (FAO) shows in [4], yearly in the world IUU-fishing extract around 26 million tons of seafood. Recent studies, such as that of U.R.Sumalia et al. [5], estimated between 8 and 14 million metric tons of unreported catches estimating financing of the illegal fishing market of US$9 billion to US$17 billion in the world. In addition, the study estimates the annual economic impact and the loss of tax revenue for the countries between US$2 and US$4 billion, showing a table with the detailed data.

Vessel Monitoring Systems (VMS) and Automatic Identification Systems (AIS) are powerful tools for authorities to address legislative challenges as IUU- fishing. Any VMS requires technology on the vessel, onshore, and communication between them. Specifically,

AIS systems provide real-time vessel type and position information. All other ships in turn can know the positions of their nearest neighbors.

Although these systems are mandatory, the use of radars is essential for a robust and functional VMS. Unlike the GPS systems associated with vessels, radars only provide the position of objects in the detection range, so they cannot differentiate the type of vessel. Having systems capable of extracting context information related to the trajectories that these systems can identify is of particular interest not only to improve the accuracy of VMS systems but also to identify irregularities in maritime traffic, to identify illegal activities, or even for rescue work.

In this paper we address the problem of vessel classification based on trajectories. Specifically, the binary classification of fishing ships trajectories versus other ships, studying the performance provided by two trendy deep learning (DL) approaches. These approaches transform the classification problem into a data learning problem. This learning problem is base to adjust the internal neural network parameters, so the low-level problem is translated into an optimization problem. To address the learning problem associated with approximations with highly unbalanced real-world data, we propose to use weights in the cost function during the learning phase and compare the performance with other classical techniques.

This paper is organized as follows: Section 8.2. shows the related works. Section 8.3. describes the methodology describing the data set, validation strategies, classification models and the learning problem. Experimental results are shown in Section 8.4. Finally, the conclusions are presented in Section 8.5.

8.2. Related works

In D. Sánchez et al. [6] authors focus on feature extraction of vessel trajectories from the Danish Coastal getting from the AIS system [7]. The authors propose a data preprocessing methodology that they validate using a set of classical machine learning (ML) approximation classifiers to solve the binary problem of fishing ships versus other vessels. These approximations show high sensitivity to the bias of the data in the proposed binary problem. The authors introduce the problem unbalanced data and evaluate the performance of classifiers applying different subprocesses including two balancing methods, Random Undersampling (RUS) [8] and Synthetic Minority Over-sampling Technique (SMOTE) [9]. Other researches such as [10, 11] also focus on the classification of fishing ships with classical approaches and searching features to describe vessel trajectories. However, these approaches seem to have reached an impasse as can be seen in [6], so use other approaches with high results in other fields such as Deep Learning (DL) is justified.

From the DL classification approach, there are two main branches. The first one uses a Convolutional Neural Networks (CNN) [12, 13] and the second one uses Long Short-Term Memory network (LSTM) [14, 15]. The previous works focus in one of the two branches, nonetheless a comparison between the two trends on the same problem can yield interesting solutions. For example, in estimation problems related to ships trajectories we can find comparisons between LSTM and CNN solutions. In [16] three solutions with CNN, LSTM and a hybridization between both, BDLSTM-CNN, are compared for the estimation of the maritime vessel flow. In this research the LSTM architecture has a lower Error Rate than CNN, but both are improved by the BDLSTM-CNN proposal in the prediction problem over 4 time steps.

The features used to define the flow are limited to the spatial grid of the study, being a bounded solution to the case study.

CNNs in classification problems have two principal steps, feature extraction (convolutional phases) and classification. These approaches have proven to be one of the most accurate techniques in multiclass problems due to their ability to learn relevant classification features. CNN architectures require a fixed input structure, for example, an image or in the case of trajectories a fixed time window. LSTM models are widely used with sequential data, mainly in non-Markovian systems such as natural language processing [17] or human behavior [18]. This is because their long-term memory properties allow remembering past trends of the states, understanding as state the temporal input variables to the network. Generally, DL-based classification systems use cross-entropy [19] as a cost function to optimize the internal network parameters. Related work on real-world data such as [20, 21] proposes the use of weights associated with classes to improve learning and avoid classification bias. This technique is known as Weighted Cross-Entropy (WCE). The research of Y. Sousa et al. [20] propose a new approach called CSEFMLP (Cost-Sensitive Cross-Entropy Error Function for MLP neural networks) based on the cross-entropy radius to weight the weights of the cross-entropy. This radius evaluates the contribution of each class on the cost function. The authors compare the performance of the proposal with other common balancing-classification techniques for different unbalanced databases. They finally conclude that the performance generally improves or at least similarly the performance of other strategies. In the work of M.R. Rezaei et al. [21] a binary classification problem with the Inception-V3 network is presented in which different increasing values of weights associated to the cost function are compared under usual classification metrics. WCE assigns more weight to the minority class, penalizing more incorrect predictions, and thus enabling better and faster training for the minority class. Research such as that of Shen Lu et al. [22] proposes a dynamic weighting cross-entropy technique for semantic segmentation. In this type of WCE the value of the weights changes as the background is differentiated from the non-background. Their research shows

the use of the proposed WCE method improves the degradation from the extremely unbalanced data. In recent studies such as [23] or J.P.Llerena et al. [24] authors used LSTM networks for state-space estimation and filtering in highly nonlinear systems. These studies show the LSTM learning capacity in complex environments. It also shows the possibilities of using Sequence-to-sequence architectures that allow step-by-step estimation opening the possibilities to classify step-by-step temporal sequences or trajectories.

8.3. Methodology

In this paper, we propose to compare the two principal deep learning approach trends in literature, CNN, and LSTM architectures, to classify an extremely imbalanced data problem and find the contribution to use WCE in relation to other balance techniques. Specifically, this paper use kinematics features from trajectories of two classes of vessels (fishing ships and other ships). For this purpose, the methodology section first describes the database, its structure, and partitioning for the learning problem with two different validation strategies. Then we briefly describe the classical approach model that we will use as comparison. Finally, this section is concluded with the proposed neural network architectures used and the justification for the weight selection proposal method in the NN-learning problem.

8.3.1. Real world binary dataset

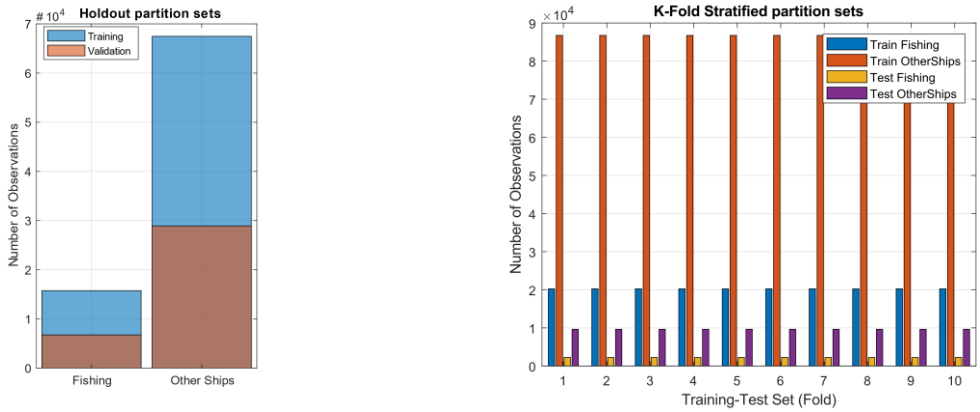
We use trajectories of AIS-recorded ships off the coast of Denmark [7] preprocessed according to [6]. Each AIS raw trajectory is defined as a succession of spatial positions and a sampling time. This research explains in detail the data preprocessing and the set of references on which they are based to finally extract the kinematic states of the vessels. In addition, the segmentation of the resulting trajectories is justified, and finally, a set of trajectories defined by continuous segments of 50 samples, normalized 10-feature space and N elements. In this work, we start from this data structure provided by its authors. Specifically, the database Φ is structured for a learning problem $\Phi = \{(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} | i = 1, \dots, N\}$ where the tensor features $\boldsymbol{x}_i^1 = \{t, p_{x,y}, v_{x,y}, \Delta v, |v_{x,y}|, d, \Delta\psi, \Delta t\}$, building a data panel of 10 features and 50 samples, $\boldsymbol{x}^1 \in \mathbb{R}^{NX10X50}$, superscript one over x means first data structure. Where t is the normalized sample times, $p_{x,y}$ is the normalization position component, $v_{x,y}$ speed components, Δv is the speed variation, $|v_{x,y}|$ is the speed vector modulus, d is the distance inter-samples, $\Delta\psi$ is the direction variation, and the end Δt is the time gap. In order to flatten the data panel $\boldsymbol{x}_i^1$, to be applied in classifiers with a flattened fixed input, we generate a flatten vector as [6] composed of normalization total time and five main blocks. In addition to the statistical characteristics of [6], the sum value is included in each feature block. Specifically,

$\boldsymbol{x}_i^2 = \{T.Time, speed, distance, Course\ variation, speed\ variation, time\ gaps\}$.
Each block composes by sum, mean, max, min, standard deviation, mode, and three quartiles, from each feature from the previous panel data x_i^1, getting a vector of 46 static characteristics for each trajectory, were $x^2 \in \mathbb{R}^{NX46X1}$. In this way, we define the trajectory i of vessel y_i, in two different ways, x_i^1 and x_i^2. So, we use two learning sets $\Phi_1 = \{(x_i^1, y_i) \in \mathcal{X}^1 \times \mathcal{Y} | i = 1, \dots, N\}$ and $\Phi_2 = \{(x_i^2, y_i) \in \mathcal{X}^2 \times \mathcal{Y} | i = 1, \dots, N\}$. Being N=118884, the number of trajectories. Although the AIS system can differentiate a total of 18 different classes of vessels, in this paper we focus on the fishing class as opposed to the rest, in other words, two different classes $\{A, B\}$, fishing ships, and other ship types respectively. Associating to each sample the class A or B we obtain that $N_A = 22495$ and $N_B = 96389$, where it is easily observed that the classes are unbalanced Fig. 8.1 and propagated for training and validation. While in a balanced set the samples would be split 50/50, in our case the samples are split 81.1% for B and 18.9% for A.

8.3.2. Data for validation methodologies

In this paper we use two of the main techniques for validation models Holdout and stratified K-fold. To apply each of these methodologies, the database is divided differently. For the holdout case the database is divided into two parts with random data selection, training, and test. Specifically, 70% of the data is selected for training and the remaining 30% for validation $N = \{N_{70\%}, N_{30\%}\} = \{83218, 35666\}$. Fig. 8.1 (a) shows the distribution of training and test data. The number of trajectories per class in holdout partition is $N_A = \{15746, 6749\}$ and $N_B = \{67472, 28917\}$.

On the other hand, to apply K-fold validation, the dataset is divided into K packages (folds) similar than holdout structure (training and test).



(a) (b)

Fig. 8.1. Data partition to validation methodology. (a) Holdout partition. Blue training data, brown validation data. (b) K-Fold stratified partition set.

The selection of the data for the K-folds is done randomly but keeping the same proportion by classes as in the starting set. This type of selection is called K-Fold stratified and show in Fig. 8.1 (b). A fair amount of research has focused on the empirical performance of cross-validation. In R. Bharat Rao and Glenn Funng [25] research different studies are discussed where it seems to have been agreed that a value of k=10 usually shows good bias-variance compensation results for classifiers. In addition, Sánchez et al. [6] authors include a K-Fold section where they use 10 folds. Based on previous research in this paper we use 10-folds. Each fold training-test set has $N_A = \{20245, 2250\}$ and $N_B = \{86750, 9639\}$ trajectories per class. In both cases of Fig. 8.1**,** the different numbers of trajectories per class show an imbalanced distribution.

8.3.3. Classical approach

A first attempt to differentiate fishing ships from other ships using features extracted from vessel trajectories is to use the classical decision tree algorithm. The algorithm is integrated into the MATLAB Machine Learning Toolbox [26], based on the classic text of Leo Breiman et al. [27]. This method requires a flattened vector $\boldsymbol{x}^2 \in \mathbb{R}^{NX46X1}$ with fixed input features, so it uses the preprocessed set Φ_2 described in section 8.3.1.

8.3.4. Deep Learning approach

Since the features extracted from the trajectories can be expressed in a temporal way Φ_1 and in a static way Φ_2, we propose to address the problem with the most common DL trends in the literature, LSTM, and CNN networks. Both proposals share the SoftMax output layer and cross-entropy cost function [19]. Thus, the classification problem becomes a biased learning problem.

8.3.4.1. CNN classifier structure

The CNN structure is inspired by the feature extraction stage in the architecture proposed by [28] for speech command recognition and the famous AlexNet architecture [29] for the classification step. We generate two CNN networks that differ at the input layer. On the one hand, the first one addresses the static database with 46x1x1 structure, while the second one uses the data panel with 10x50x1 structure. Finally, the architecture is composed of an input layer, a convolutional module C, two consecutive fully connected layers with a Rectified Linear Unit (ReLU) activation function of 4096 units and 30% dropout. The last block is composed of a fully connected layer with 2 units (binary classification) and a SoftMax layer, Fig. 8.2.

Module C include 3 consecutive convolutional blocks. Each convolutional block is composed of four layers, convolution, batch normalization, activation, and pooling. The convolutional layer has 12, 24, and 48 filters respectively of 3x3 filter size, astride of 1x1, and padding same to ensure the output has the same size as the input. Then a batch normalization

layer follows of ReLU activation layer and pooling layer. The max pooling layer is composed of pool size 3x3 and stride 2x2 and padding same. In the case of temporary CNN, the last convolutional block is duplicated.

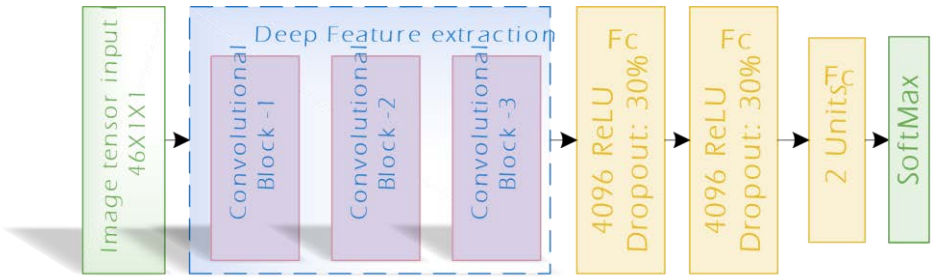


Fig. 8.2. CNN Architecture, first block: input layer. Second: convolutional deep feature extraction module. Third and fourth: Fully connected plus ReLU layer and dropout. Last two layers, fully connected layer and SoftMax layer.

It is used Adam optimizer [30] with an initial learning rate of 3.10^{-3} and learning rate drop factor 0.09. The minimum batch size is 200 and 40 epochs. In the case of K-Fold each of the K-Folds is trained in the same way but with 15 epochs.

8.3.4.2. LSTM classifier structure

In the case of the LSTM architecture, the network learns hidden transitions of the temporal sequences of the input features. The model used is employed in the work of [31] for fault detection in chemical processes, and can be seen represented in Fig. 8.3. The architecture is composed of a 10x50 sequence input layer, three consecutive LSTM layers with 20% dropout and 52, 40, and 25 hidden units respectively. Finally, a fully connected layer with two units and a classification SoftMax layer. The outputs of the first two LSTM layers are complete sequences, while in the case of the last LSTM layer only the last step sequence is selected to connect it to the fully connected layer.

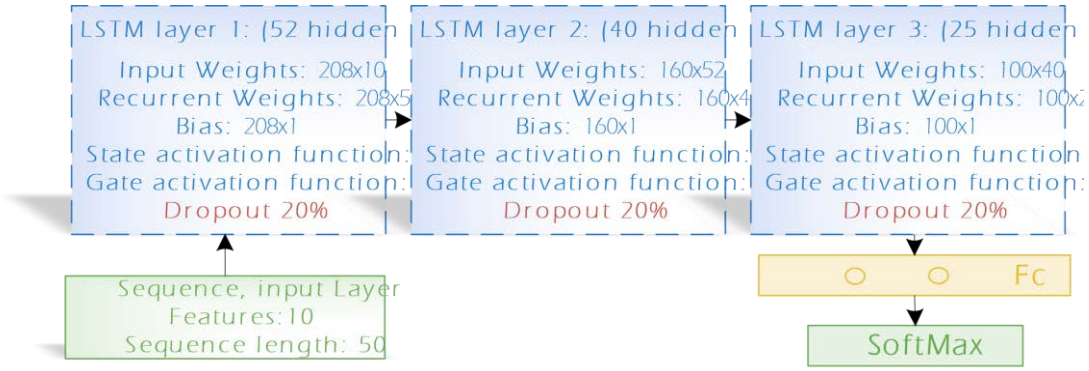


Fig. 8.3. LSTM Architecture, first block input layer (sequence), second to fourth block LSTM layers with dropout. Last two layers, fully connected layer and SoftMax layer.

The training hyperparameters are optimizer Adam, minimum batch size 550. Gradient thresholding is applied with value 1 using the L_2 norm.

8.3.4.3. Learning problem

The main relation between CNN and LSTM approach lies in the output SoftMax layer and the cost function. Specifically, the typical cost function $\mathcal{L}$ used in classification deep learning problems is cross-entropy, that compare two probability distribution over the same probability space.

Given a database $\Phi =$, in which each element i is described by a feature tensor x_i and has a $q_{i,k}$ probability of belonging to a categorical class k, designing an artificial neural network classifier $\hat{q}_{\boldsymbol{\theta}}(x_i)_k$, involves identifying the internal NN $\boldsymbol{\theta}$ parameters that minimize the error function $\mathcal{L}$ (8.1).

$$\mathcal{L}_{\mathrm{NN}}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} w_k\, q_{i,k} \ln(\hat{q}_{\boldsymbol{\theta}}(x_i)_k) \tag{8.1}$$

Where, K is the total number of classes, and N is the total number of samples.

The real probability of sample i belonging to class k are $q_{i,k}$ and $\hat{q}_{\theta}(x_i)_k$ the probability given by the prediction of the model/neural network with internal weights $\boldsymbol{\theta}$ of sample i (with inputs x_i) belonging to class k. On the other hand, $q, \hat{q}\colon \mathbb{R}^K \longrightarrow [0,1]^K$ and in the case of $\hat{q}$ provided by the SoftMax function (8.2).

$$\hat{q}(z_i)_k = \frac{e^{z_{i,k}}}{\sum_{k=1}^{K} e^{z_{i,k}}} \tag{8.2}$$

Where $z_{i,k} = f_{\boldsymbol{\theta}}(x_i)$ is the k element of the output vector at the network output function f_{θ}, before getting the SoftMax layer when introducing a x_i feature tensor. In other words, $f_{\boldsymbol{\theta}}(x_i)$ is the NN function. We use the hat over q ($\hat{q}$) inherited from the estimator notation since the network provides an estimate while q tells us the true value of the database. w_k is the weight associated with class k. For the case of two classes A and B:

$$\mathcal{L}_{\mathrm{NN}}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} w_A q_{i,A} \ln(\hat{q}_{\boldsymbol{\theta}}(x_i)_A) + w_B q_{i,B}\, \ln(\hat{q}_{\boldsymbol{\theta}}(x_i)_B) \tag{8.3}$$

Since the SoftMax function provides the probability of belonging to each A or B class, $\hat{q}_{i,A} + \hat{q}_{i,B} = 1$, and likewise the database probabilities satisfy the same relationship but in an unequivocal way $q_{i,B} = 1 - q_{i.A}, q_{i,k} \in \{1,0\}$:

$$\mathcal{L}_{\mathrm{NN}}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} w_A q_i \ln\left(\hat{q}_{i,\boldsymbol{\theta}}\right) + w_B(1-q_i)\ln\left(1-\hat{q}_{i,\boldsymbol{\theta}}\right) \tag{8.4}$$

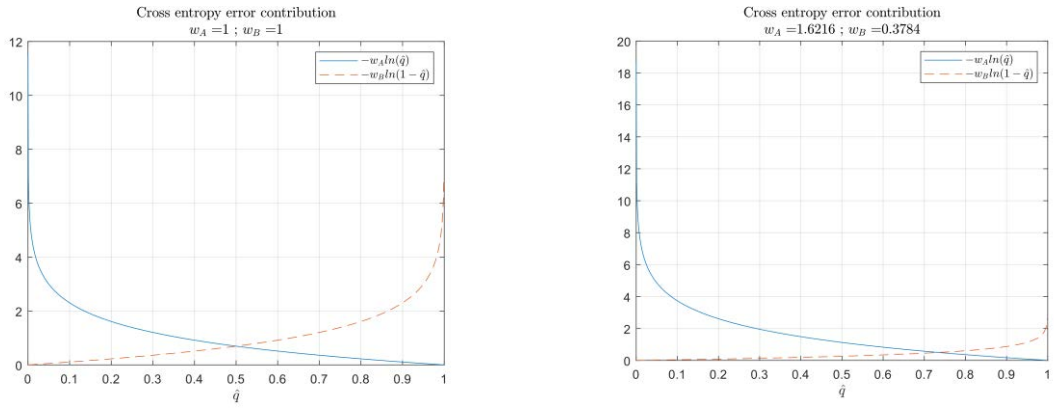
The first term of the summation in (8.4), contributes to the error of the positive terms (fishing ships) while the second term contributes to the error of the negative terms (other vessels). If the database is balanced means $N_A = N_B = 0.5N$, the probability of element i belonging to the classes A or B is the same [20], in such that both terms will act the same number of times $q_{i,k} = \frac{N_k}{N} = \frac{1}{2}$. However, if the database is unbalanced, $q_{i,A} \neq q_{i,B}$ the majority subset will contribute more times in the error function than the minority subset. This effect over the cost function can be balanced by weights (w_k), were:

$$\begin{cases} N_A = N_B, w_A = w_B = 1 \\ N_A \ll N_B, w_A > 1 \; y \; w_B < 1 \\ N_A \gg N_B, w_A < 1 \; y \; w_B > 1 \end{cases} \tag{8.5}$$

In order to have the same contribution of each class over the error function $w_A q_{i,A} = w_B q_{i,B}$, $w_A = \frac{N_B}{N_A} w_B$. The previous relationship indicates that the weight associated with a class k must be inversely proportional to the number of samples N of that class. In addition, to maintain the definition of cross-entropy (8.1), the sum of the weights must be equal to the number of classes $\sum_{k=1}^{K} w_k = w_A + w_B = K = 2$. Based on these restrictions, we use (8.6).

$$w_k = \frac{1/N_k}{\frac{1}{K}\sum_{k=1}^{K} 1/N_k} = \frac{K}{N_k \sum_{k=1}^{K} 1/N_k} \tag{8.6}$$

Fig. 8.4 shows the contribution of the classifier error to the cross entropy to applied equation (8.6). When the database is balanced the effect of the classification error is similar in booth classes. However, in our case the database is imbalanced with a ratio around to the 18,9% to the positive class and 21,9% for the negative class. With these relationships equation (8.6) gives the weights of Fig. 8.4 (b). This figure shows the error effect of the minority class, solid blue line, is stronger than the majority class, dashed red line.



(a) (b)

Fig. 8.4. Contribution to cross-entropy with weight adjustment (8.6). Solid blue line, positive class, dashed red line, negative class. (a) Weights for balanced database. (b) Weights for unbalanced database.

8.4. Experiments and results

The experimentation seeks to evaluate the performance of the two proposed deep learning approaches to the same fishing ship classification problem, introducing as a novelty the WCE to solve the classification bias produced by the imbalance data. The classification methods are a decision tree, CNN, LSTM and CNN by fix time window over kinematics data that we call CNN_T. During the training phase, unbalanced raw features (Without B.), two classical balancing methods RUS and SMOTE and the WCE formulated in 8.3.4.3. Learning problem section. We applied two validation strategies, holdout, and stratified K-Fold with $K = 10$. RUS and SMOTE balancing techniques are applied over each k-folds training sets, isolated synthetic disturbance validation folds or decimation data. The results are compared in $\mathbb{R}^2$ bounded space defined by accuracy (8.7) and F_1-Score (8.9) metrics. The first metric indicates an overall accuracy of the positives, however in an unbalanced sample a classifier can show a very high accuracy without detecting some classes, so the classifier's will also be unbalanced. For this reason, we use the harmonic mean between precision and recall (8.8) called F_β-Score or F_1. Where β is a weight indicating the importance, we give to precision versus recall, in our case the same ($\beta = 1$). The quantitative results are shown in Table 8.1 and Table 8.2. The union of these two metrics forms a plane $\mathbb{R}^2$ of precision in which the ideal classifier would be at coordinates (1,1).

$$Acc = \frac{Tp + Tn}{Tp + Tn + Fp + Fn} \tag{8.7}$$

$$Precision = \frac{Tp}{Tp + Fp};\ Recall = \frac{Tp}{Tp + Fn} \tag{8.8}$$

$$F_\beta - Score = (1+\beta^2)\cdot\frac{Precision\cdot Recall}{\beta.Precision + Recall} \tag{8.9}$$

Where Tp, Tn, Fp and Fn mean the set of combinations between true-false and positive or negative cases in the confusion matrix. In the confusion matrix, the first class is defined by the minority class A, fishing ships.

Table 8.1. Accuracy results (Holdout|K-Fold).

Feature type	Model	Without B. [%]	RUS [%]	SMOTE [%]	WCE [%]
Static: Φ_1	Tree	71.07|83.27	52.30|82.25	76.80|79.03	--------
	CNN	75.85|84.90	75.34|79.74	73.73|79.52	79.80|74.45
Kinematic: Φ_2	LSTM	81.08|83.75	69.79|80.20	78.94|83.65	81,38|77.12
	CNN_T	83.16|85.26	81.34|89.53	82.31|80.96	80.23|80.22

Table 8.1 shows the results of the accuracy of the four classification models against the different balancing techniques used. On the other hand, the results obtained in the F-score are shown in Table 8.2.

Table 8.2. F_1-Score (Holdout|K-Fold)

Feature type	Model	Without B. [%]	RUS [%]	SMOTE [%]	WCE [%]
Static: Φ_1	Tree	66.59|72.71	35.71|72.48	70.30|74.19	--------
	CNN	74.62|73.97	64.85|78.25	74.76|77.75	68.86|75.50
Kinematic: Φ_2	LSTM	-------|68.70	71.68|80.84	69.72|68.71	73.48|74.78
	CNN_T	67.20|72.50	60.34|89.40	63.87|79.20	70.42|78.71

Table 8.2 shows a strong bias for LSTM in holdout validation strategy when it is does not apply a balancing technique. This model classifies the validation set as the majority class, obtaining a similar accuracy as the percentage data of the majority class, but a null F_1- Score. The classifier with the highest accuracy in holdout and k-fold is CNN_T with unbalanced data and RUS respectively. For the F_1- Score case the static CNN and CNN_T acquire the best results with a RUS. While in the case of holdout the classifier with the best accuracy and F_1- Score relationship is the LSTM with WCE, these results are improved when K-Fold is used. Specifically, CNN_T with RUS shows significantly higher values than the other classifiers. WCE improves F_1- Score in 83% of cases and improves the accuracy of CNN and LSTM using Holdout validation. In the case of the tree classification, WCE has not been applied, so in Table 8.1 and Table 8.2, the results are shown with dashed lines. Using the accuracy and F_1-Score variables shown in Table 8.1 and Table 8.2 as variables of an $\mathbb{R}^2$ space, it is obtain Fig. 8.5. This figure shows the distribution of the different classifiers whit holdout and K-Fold validation strategies on the evaluation space-plane. In Fig. 8.5, 90% of the classifiers appear clustered around 70-90% accuracy and 65-90 F_1- Score, while 3 of the classifiers are positioned very far apart due

to lack of accuracy, F_1- Score or both. To differentiate between them, we zoom into the region of maximum classifier concentration.

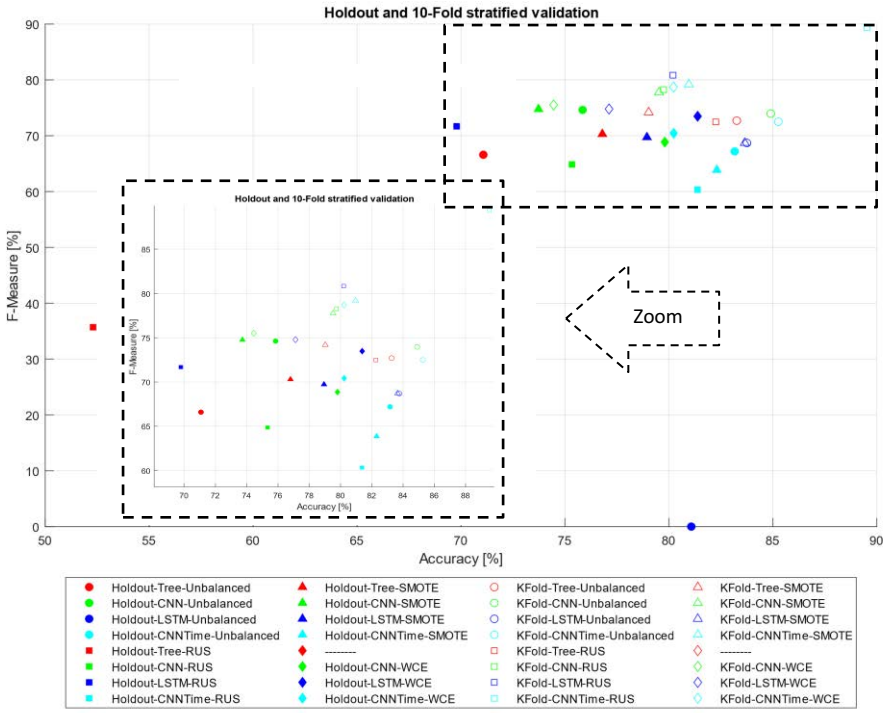


Fig. 8.5. Classifiers with different balancing techniques and validation strategies. Solid markers Holdout, solid edge without face color K-Fold. Edge and marker face color, {red, green, blue, cyan} = {Tree, CNN, LSTM and CNN Time}. Markers {circle, square, triangle and diamond} = {unbalanced, RUS, SMOTE, WCE}.

8.5. Conclusions

This paper has presented two main approaches to automatically classify fishing ships from other vessels using trajectory information. The two approaches used are classical and DL. In addition, in the DL approach, two of the main trends in the literature, LSTM, and CNN neural networks, have been compared using panel and flattened data structure. The classifier clustering around the same region of the comparison plane suggests a similarity of the evaluation methods. However, applying K-Fold eliminates outliers in the classifiers, clustering the results to a greater extent than holdout on the comparison plane. The performance of the DL approximations is superior to the classification tree approximations. It has been shown that in the case of the CNN architecture the influence of the bias in the data is not as pronounced as in the LSTM, which is completely biased in classification. However, applying balancing techniques produces a considerable improvement, achieving the best accuracy-F-Score ratio applying WCE.

The successful results of CNN with both static and sequential kinematic data structures suggest that the convolutional layers are capable to extract hidden features from the trajectories. Furthermore, applying RUS in the training process has shown good results with CNNs, which may be a consequence of the decimation of large numbers of segments from the same complete trajectories. WCE has been shown to improve or at least equal results in the case of holdout validation, but with K-Fold cross-validation other strategies have been shown to be superior in the face of bias. For an ideal classifier, the evaluation with Holdout and K-fold should be similar. However, as seen in Table 8.1 and Table 8.2 a high variability of results is observed due to random under-sampling with RUS method. Even so, the proposed WCE method, with the CNN_ T classifier has a 1% change between the two evaluation methods, so it is considered more stable than the rest of the proposed methods and classifiers. Although the general results show a superior performance of CNNs, the ability of LSTMs to work dynamically on non-fixed sequences of data or the possibility to deepen and densify the presented structure or even hybridize it with convolutional layers shows great opportunities for problems such as this work. Finally, this paper provides a new approach to the vessel classification.

Funding: This research was partially funded by public research projects of Spanish Ministry of Science and Innovation, references PID2020-118249RB-C22 and PDC2021-121567-C22 - AEI/10.13039/501100011033, and by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors, reference EPUC3M17.

Acknowledgments: We would like to thank David Sánchez and Daniel Amigo for sharing a set of trajectories used in their published work [6].

8.6. References

[1] S. Upton and V. Vitalis, “Stopping the high seas robbers: coming to grips with illegal, unreported and unregulated fisheries on the high seas,” Round Table Sustain. Dev. OECD, Paris, no. June, p. 18, 2003.

[2] T. H. Tai, S. M. Kao, and W. C. Ho, “International soft laws against IUU fishing for sustainable marine resources: Adoption of the voluntary guidelines for flag state performance and challenges for Taiwan,” Sustain., vol. 12, no. 15, 2020.

[3] E. Commission:, “International fisheries relations | Fact Sheets on the European Union | European Parliament.” [Online]. Available: https://www.europarl.europa.eu/factsheets/en/sheet/119/international-fisheries-relations. [Accessed: 08-Jun-2021].

[4] F. Food and Agriculture Organization of the United Nations, “The fight to save our oceans | FAO Stories | Food and Agriculture Organization of the United Nations.” [Online]. Available: http://www.fao.org/fao-stories/article/en/c/1136937/. [Accessed: 08-Jun-2021].

[5] U. R. Sumaila, D. Zeller, L. Hood, M. L. D. Palomares, Y. Li, and D. Pauly, “Illicit trade in marine fish catch and its effects on ecosystems and people worldwide,” 2020.

[6] D. Sánchez, D. Amigo, J. García, and J. M. Molina, “Architecture for trajectory-based fishing ship classification with AIS data,” Sensors (Switzerland), vol. 20, no. 13, pp. 1–21, 2020.

[7] Danish Maritime Authority: AIS data sets, “AIS data.” [Online]. Available: https://dma.dk/SikkerhedTilSoes/Sejladsinformation/AIS/Sider/default.aspx#. [Accessed: 07-Jun-2021].

[8] A. Fernández, S. García, M.-G.-R. C.Prati, B. Krawczyk, and F. Herrera, Learning from Imbalanced Data Sets. .

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” Ecol. Appl., vol. 30, no. 2, pp. 321–357, 2020.

[10] P. Kraus, C. Mohrdieck, and F. Schwenker, “Ship classification based on trajectory data with machine-learning methods,” Proc. Int. Radar Symp., vol. 2018-June, pp. 1–10, 2018.

[11] I. Kontopoulos, K. Chatzikokolakis, K. Tserpes, and D. Zissis, “Classification of vessel activity in streaming data,” DEBS 2020 - Proc. 14th ACM Int. Conf. Distrib. Event-Based Syst., pp. 153–164, 2020.

[12] H. Ljunggren, “Using Deep Learning for Classifying Ship Trajectories,” 2018 21st Int. Conf. Inf. Fusion, FUSION 2018, pp. 2158–2164, 2018.

[13] K. il Kim and K. M. Lee, “Convolutional neural network-based gear type identification from automatic identification system trajectory data,” Appl. Sci., vol. 10, no. 11, 2020.

[14] S. Hochreiter and J. Urgen Schmidhuber, “Long Shortterm Memory,” Neural Comput., vol. 9, no. 8, p. 17351780, 1997.

[15] W. Srisukkham, L. Pipanmaekaporn, and S. Kamonsantiroj, “A RECURRENT NEURAL NETWORK MODEL for DETECTING FISHING GEAR PATTERNS,” ICIC Express Lett., vol. 15, no. 6, pp. 627–637, 2021.

[16] X. Zhou, Z. Liu, F. Wang, Y. Xie, and X. Zhang, “Using deep learning to forecast maritime vessel flows,” Sensors (Switzerland), vol. 20, no. 6, pp. 1–17, 2020.

[17] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc., pp. 1–15, 2015.

[18] S. Swetha, V. N. Balasubramanian, and C. V. Jawahar, “Sequence-To-sequence learning for human pose correction in videos,” Proc. - 4th Asian Conf. Pattern Recognition, ACPR 2017, pp. 268–273, 2018.

[19] Y. Ho and S. Wookey, “The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling,” IEEE Access, vol. 8, pp. 4806–4813, 2020.

[20] Y. S. Aurelio, G. M. de Almeida, C. L. de Castro, and A. P. Braga, “Learning from Imbalanced Data Sets with Weighted Cross-Entropy Function,” Neural Process. Lett., vol. 50, no. 2, pp. 1937–1949, 2019.

[21] M. R. Rezaei-Dastjerdehei, A. Mijani, and E. Fatemizadeh, “Addressing Imbalance in Multi-Label Classification Using Weighted Cross Entropy Loss Function,” 27th Natl. 5th Int. Iran. Conf. Biomed. Eng. ICBME 2020, no. November 2020, pp. 333–338, 2020.

[22] S. Lu, F. Gao, C. Piao, and Y. Ma, “Dynamic Weighted Cross Entropy for Semantic Segmentation with Extremely Imbalanced Data,” Proc. - 2019 Int. Conf. Artif. Intell. Adv. Manuf. AIAM 2019, pp. 230–233, Oct. 2019.

[23] S. H. Rudy, J. Nathan Kutz, and S. L. Brunton, “Deep learning of dynamics and signal-noise decomposition with time-stepping constraints,” J. Comput. Phys., vol. 396, pp. 483–506, 2019.

[24] J. P. Llerena, J. García, and J. M. Molina, “Forecasting nonlinear systems with lstm: Analysis and comparison with ekf,” Sensors, vol. 21, no. 5, pp. 1–29, 2021.

[25] R. B. Rao, G. Fung, and R. Rosales, “On the dangers of cross-validation. An experimental evaluation,” Soc. Ind. Appl. Math. - 8th SIAM Int. Conf. Data Min. 2008, Proc. Appl. Math. 130, vol. 2, pp. 588–596, 2008.

[26] “Statistics and Machine Learning Toolbox - MATLAB.” [Online]. Available: https://es.mathworks.com/products/statistics.html. [Accessed: 19-Dec-2022].

[27] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, “Classification and regression trees,” Classif. Regres. Trees, pp. 1–358, Jan. 2017.

[28] M. H. Beale, M. T. Hagan, and H. B. Demuth, “Neural Network Toolbox TM User ’ s Guide R2013b,” Mathworks Inc, 2013.

[29] B. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Commun. ACM, vol. 60, no. 6, pp. 84–90, 2012.

[30] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc., pp. 1–15, 2015.

[31] X. Song et al., “Time-series well performance prediction based on Long Short-Term Memory (LSTM) neural network model,” J. Pet. Sci. Eng., p. 106682, Nov. 2019.

Conclusions and future research

This section presents a summary and conclusions of the works detailed throughout the different chapters of the thesis. In addition, a number of future lines of work are provided to continue the practical examples proposed at the end of each chapter.

It's important to note that the main contributions of this thesis are concentrated in chapters 4, 6, 7, and 8. Chapters 1, 2,3 and 5 are used as an introduction of the principal contributions. For each contribution, the error is evaluated and novel strategies and methods to mitigate it are provided.

Part I: Drones, Navigation and Vision-based precision landing

- **Chapter 1: Overview on drones**

Drones are just one branch of the UAV cluster, where the drive for new business opportunities is catapulting the technologies that comprise them. UAS are complex ecosystems composed of a variety of subsystems that perform specific tasks and are underpinned by powerful physical and mathematical foundations. The development of new applications requires a broad approach to their operation and internal communication mechanisms. In order to advance the development of safe applications that comply with current regulations, hyper-realistic simulation in SITL and HITL are shown to be powerful development tools.

- **Chapter 2: UAS INS/GNSS Navigation**

Navigation systems are based on the theory of estimation based on state space models and Kalman filter. Although the concept of navigation is broad and there are different navigation problems, the most widely used in the UAS field are INS/GNSS navigation systems. The configuration of these systems is not easy. Not only do they depend on the parameters provided by the sensor manufacturers, but the definition of the measurement models, prediction model, and process noise is critical. Mastering the operations for which a specific navigation system is intended is essential for fine-tuning these systems. This opens up new opportunities such as proposed in the paper "Tuning process noise in INS/GNSS fusion for drone navigation based on evolutionary algorithms" recently accepted at the 18th International Conference on Soft Computing Models in Industrial and Environmental Applications, SOCO 2023. In addition, although the vector structure of these systems allows

to increase their state vector, the influence of other states can be a great challenge for researchers and engineers.

- **Chapter 3: Machine Vision Systems of UAS**

Computer vision applied to UAS covers a wide range of fields. A clear example is vision-based navigation systems, image stabilization, or object tracking. These systems are multidisciplinary because they require in-depth knowledge of navigation theories, physics of motion, and computer vision systems. Although the current fashion in vision systems is to focus on deep learning, it is important to consider the classical approaches based on physical camera models, since they allow traceability of results, unlike the black box models of ANNs. Both approaches are complementary and can exploit each other's potential to solve specific problems. Among the various vision problems presented in the introduction, physical modeling and camera calibration, image stabilization, and object detection and tracking were highlighted. The problems of object detection and tracking in images will continue to be studied as a continuation and future work. These problems present interesting challenges where classical approaches are surpassed by systems capable of maintaining short to long term temporal trends or understanding context. However, it is important to remain cautious and continue their study beyond the demonstration of results.

- **Chapter 4: Error Reduction in Vision-Based Multirotor Landing System**

Although the chapter itself has a section on specific conclusions, here we focus on key points. First, it is important to emphasize how contextual information helps navigation systems improve accuracy. Vision systems are shown to be powerful sources of information that, in order to be used efficiently, require a process of adaptation so that the results at the output of the preprocessing are consistent with the rest of the subsystems in which it is desired to integrate. On the other hand, the error sources of the sensors and their integrations are propagated throughout the ecosystem, so a study of their behavior is essential for proper operation. The propagation of these errors, together with a high sampling frequency, produces an unwanted spin effect in the navigation system, which can be mitigated by rigorous study of the same and new landing strategies without the need to modify the flight controller. In future work, the systematic error of the vision system can be corrected by estimating the system bias, including the application of novel estimation strategies such as mismatch estimation. Another interesting area of study is the problem of landing surface detection and tracking in the face of resolution changes caused by altitude variations or environmental conditions. In addition, modeling the error of the vision system may help in future research on minimum time search systems applied to precision and emergency landings.

Part II: Deep Learning, forecasting, and filtering.

- **Chapter 5: Artificial Neural Networks.**

Artificial Neural Networks is an amazing field of study with exciting advances at the present time. This is continually expanding its application domain and major advances are being made on a daily basis. However, this poses an enormous challenge for researchers who wish to keep up with the state of the art. The selection of sections that make up this chapter has been carefully made to strike a balance between fundamentals and state of the art in a generalized manner. One of the conclusions we can draw from this chapter is that not always the architecture that is fashionable at the moment is the best solution to all problems. A constant critical vision is required without being blinded by the enthusiasm of the results. It is very important to keep an open mind in this field since approaches or ideas with modest results can be revolutionary in different contexts. There is undoubtedly much work to be done, for example, unveiling the internal dynamics of black box models can open up their application to critical systems.

- **Chapter 6: An approach to forecasting and filtering noise in dynamic systems using LSTM architectures.**

Understanding the prediction and filtering processes of Kalman filters along with their limitations is part of this work. This motivates the application of neural networks, specifically the LSTM models of the RNN. This is done based on the hypothesis that a LSTM is able to extract the temporal behavior of the data being able to model a dynamic system and the filtering of measurements. The results provided by the regression architecture, which we have named “neuro-estimator”, presents results comparable to the Kalman filter. However, for a linear system, when the filter reaches its steady state, the Kalman filter shows its potential as an optimal estimator. The behavior of the neuro-estimator is shown to be that of a filter, since it reduces the error of the measurements and is able to estimate future instants with past measurements or estimates. The initialization of these systems by overlapping sliding windows allows the networks to initialize their hidden states while maintaining their bounded space. When few prior measurements are available and measurements are lost, the Kalman system is susceptible to decoupling, while the results of the neuro-estimator remain bounded. This work shows the potential of networks in the prediction and filtering process, thus motivating the study on nonlinear systems where the estimation theory needs to be approximated.

- **Chapter 7: Forecasting nonlinear systems with LSTM: Analysis and comparison with EKF.**

Extending and relating the regression formulation to the forecasting and filtering formulation is one of the contributions of this work. Exploring the potential of neuro-estimators requires a strict definition of the estimation and filtering problem from the different approaches. A rigorous definition of neural architectures, the performance of hidden states and their comparison with Kalman filters is highlighted in this work. One of the main conclusions of this work focuses on the flexibility of the neuro-estimator models with respect to Kalman filters. When the system is purely linear, Kalman is optimal in the long run, but when the system starts to become nonlinear, the EKF is outperformed by the neuro-estimators, especially in terms of measurement loss. In addition, it is proposed to evaluate how robust the systems are when the measurements correspond to different models for which they have been designed. The error model of the Kalman filter gives it a proper flexibility when estimating slightly different dynamics than those for which it was designed, but the LSTMs are able to understand the trends of the measurements. This evaluation shows that Kalman filters quickly leave the filtering region, while neuro-estimators are able to continue filtering measurements even with large model variations. This work, together with the previous one, opens up a number of possible research. Among them is to go deeper into the Cramér-Rao quantile, or to evaluate its inference over time for possible embedded applications.

- **Chapter 8: LSTM vs CNN in real Ship trajectory classification**

This chapter is the result of exploring the potential of neural networks in the face of different machine learning challenges. The difference between the implementation of a regression problem and a classification problem is very small. Basically, the difference lies in the choice of the output layer and the learning cost function. Exploring the difficulties with real data is the purpose of this paper. In this work, we address the learning problem with highly imbalanced data sets. This motivates the proposal to compare different data balancing strategies with a strategy focused on learning balancing. In addition, two of the most widely used validation methods, Holdout and K-Fold, are applied to two of the predominant architectures in the literature, LSTM and CNN. The result of all these strategies and methodologies is a set of 32 classifiers, compactly presented in a table and a figure. The presentation of the results using two metrics provides an information-rich two-dimensional solution space. Among the most salient and duly extended conclusions in this chapter is that CNNs are less sensitive to information bias than LSTMs. On the other hand, the application of the proposed cross-entropy weighting in the learning process has shown significant results. Note that in all cases, the goal is to minimize the error defined by a cost function.

The next step in this line of work may be multiple classification. Moreover, the application on the UAS domain is immediate, bringing new opportunities in the generalization of a safety

flying space. On the other hand, the SoftMax output layer provides the probability of belonging to the different classes, this quality taken to the field of tracking with Interacting Multiple Model (IMM) filters can be of great interest where the model selection can be defined by the classification of the measurements.

In summary, error reduction is the common framework of all the researchers in this thesis, where the definition of error and how to deal with it is the difference between all of them.

Thesis Objectives Conclusions

The principal objectives of this thesis have been accomplished in an outstanding manner, not only with the publications that are part of the thesis, but also with those that are in production, pending publication, under review, or under study. In order to justify this conclusion, the following is a detailed description of where each of the principal and secondary objectives have been developed:

1) Define and prototype data fusion-based navigation systems: Chapter 2 focuses on fusion-based navigation systems. This knowledge is applied in Contribution three. In addition, the fine tuning of navigation systems using evolutionary and data analysis strategies is addressed in the proof-of-concept project, in the work under review "Tuning process noise in INS/GNSS fusion for drone navigation based on evolutionary algorithms" and in the book currently in production.

2) Define and prototype video analysis subsystems: During chapter 3, vision systems are explored, and the study culminates in chapter 4. In addition, in the book that is currently in production[9]., an example of object tracking with a vision-based drone and another of obstacle avoidance with monocular vision using occupancy maps are developed. On the other hand, it participates in the HADA project which focuses on the development of new automatic video segmentation systems. In parallel, it has supervised several works of national and international students related to computer vision systems.

[9] https://giaa.uc3m.es/giaa_drone_lab/

3) Design and prototype interpretation/reasoning subsystems: The systems in charge of interpreting and reasoning are intelligent systems that are shown in all the work carried out during the thesis. Specifically in Chapters 4, 5, 7 and 8.

4) Design and prototype deep learning methodologies for estimation and filtering: Two of the three reference articles of this thesis, focus on this objective by providing new approaches to the state of the art of estimation and filtering systems.

Related to secondary objectives:

I. Study indoor-outdoor UAV navigation techniques and mission planning: Chapter 1, 2 and 3.

II. To review in the literature problems associated with modeling dynamic systems with deep learning approaches. Part II.

III. To study estimation and filtering techniques. Chapter 2

IV. Explore advanced classification techniques with real data. Conference paper and extension journal paper in production, Chapter 8.

V. Explore hyper-realistic Software/Hardware In The Loop (SITL/HITL) simulation environments for experimentation with UAVs. Part I, proof of concept project SIMBAT, Book.

VI. Define models and systems that allow extracting information from the flight context. Chapter 4, 6 and 7

VII. To study sensor fusion and new deep learning architectures. Chapter 2, 6 and 7.

Thanks to the rigorous scientific methodology, the objectives of the thesis have been achieved and a small contribution in the field of support technologies for drone operations has been made.

Biography

Juan Pedro Llerena *has a bachelor's degree in physics from Complutense University of Madrid (UCM) specialized in physical devices and control. He did an inter-university master's degree between the National Distance Education University (UNED) and UCM specializing in systems engineering and control, motivating his interest in data fusion systems, artificial intelligence, computer vision, and UAVs. Currently, Juan Pedro is a Ph.D. Candidate and researcher in the Applied Artificial Intelligence Group (GIAA) at the Carlos III University of Madrid where his work focuses on the study of drone support technologies.*