Universidad
Carlos III de Madrid

Departamento de Sistemas y Automática

PROYECTO FIN DE CARRERA

TELEOPERACIÓN DE UN ROBOT MÓVIL MEDIANTE PERCEPCIÓN 3D

Autor: José Manuel Cabrera Canal

Tutor: Jorge García Bueno

Leganés, octubre de 2011

Título: TELEOPERACIÓN DE UN ROBOT MÓVIL MEDIANTE PERCEPCIÓN 3D

Autor: José Manuel Cabrera Canal

Director: Jorge García Bueno

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Tras tanto tiempo en la carrera, lo primero que me viene a la cabeza es la pregunta "¿Y ahora qué?". Parece como que el mundo se viene abajo y que nos espera una época oscura de 40, 50 años, vaya usted a saber cuántos de trabajo, pero lo cierto es que, en realidad, se nos abre un mundo lleno de posibilidades y aunque suene a frase hecha, estamos viviendo unos tiempos emocionantes para nuestra profesión, pues la tecnología avanza a un ritmo exponencial y hasta para los profesionales que nos dedicamos a este sector, supone un reto no quedarse atrás y ser capaces de innovar. Esta es la parte que me hace seguir adelante, el poder descubrir dónde y cómo estaremos mañana gracias al trabajo conjunto de nosotros, los ingenieros.

Quisiera agradecer, como es debido, a todas las personas que me han soportado, literalmente, pues no es fácil adaptarse a mi forma de entender la vida y las cosas. Hago especial mención a mis padres y hermanos cómo no, a mis dos amigos de siempre Álvaro y Dani, a Carmen, por su amor y apoyo inestimable y a Lillo, por ser como un hermano más, a parte de amigo.

Especiales recuerdos a mis compañeros de clase, de viaje y de salidas (y de discusiones), Javi, Aitor, Chumo, Juanto, Martín y muchas más personas que me han hecho pasar una gran etapa de mi vida y ayudado a tragarme el orgullo.

También dedico este proyecto a Carlos por esa magnífica Erasmus compartida en tierras normandas y a Dani, que como buen mexicano me enseñó a preparar una buena michelada.

Todo llega en la vida... ¡Por fin!

Resumen

Hoy en día, si pensamos en informática, cualquier persona nos diría que es todo aquello que tiene que ver con los ordenadores, internet y los móviles, pero afortunadamente, la informática es un mundo vasto y extenso y no existe disciplina humana en la que no se utilice o se aplique alguna herramienta de tipo informático, pues supone un ahorro considerable en tiempo y esfuerzo, así como en errores.

Dos de los campos que más llaman la atención es el de la robótica y el del videojuego, no ya por ser los más atractivos, sino por ser de los que más dinero generan, en concreto el de los videojuegos.

La robótica aplicada a vehículos comenzó allí por los años 50 y hoy en día ha alcanzado un grado enorme de sofisticación, con la aparición de los primeros vehículos controlados autónomamente.

La autonomía a la hora de gobernarse un vehículo es la nota predominante de la investigación hoy en día, pero hasta hace poco los vehículos robóticos no tripulados eran controlados a distancia, por lo que las tecnologías de comunicación inalámbrica cobran especial interés.

El mundo del videojuego puede parecer extrañamente relacionado con el tema propuesto, pero viene muy al uso pues es un dispositivo previsto inicialmente para jugar, el que será la piedra angular de este proyecto. El *Kinect*, supone una revolución en el uso de escáneres 3D, pues pone a un precio muy asequible un hardware que de otra forma el gran público no habría tenido acceso, abriendo la línea a infinidad de proyectos de carácter personal, que no hacen otra cosa que mejorar y avanzar en el campo de la investigación.

La visión por computador es otro campo muy ligado al de la robótica, utilizada para analizar el entorno que nos rodea, cuando el ser humano no está presente e incluso cuando lo está, proporcionándole información que este no es capaz de ver o percibir.

Así pues, el objetivo principal será la búsqueda de una simbiosis, una síntesis de todas estas tecnologías, adentrándonos en sus problemas y dificultades para dar solución a un problema ya bien conocido, pero visto desde un punto de vista muy personal, puesto que in-

tentaremos emular la conducción de un vehículo a escala de una forma original y divertida.

Palabras clave: visión, Kinect, escáner, 3D, robótica, vehículo, videojuego, vehículo autónomo, simbiosis, conducción, telecontrol.

Abstract

Today, if we think about computing, anyone would say that is all that has to do with PC's, internet and mobile, but fortunately, computing is a vast and extensive field and there is no human endeavor which did not apply any kind of computer technique, saving considerable time and effort, as well as errors.

Two of the fields that attract the most attention is the robotics and video games, not being the most attractive, but those who attracts the major cash, specially video games.

The robotics vehicles began back in the 50's and today they have reached an enormous degree of sophistication, with the emergence of the first vehicles controlled autonomously.

Nowadays, when it comes down to robotic vehicles, the autonomous control is the dominant note of the research, but until today, robotic vehicles were remote controlled, so that wireless communication technologies are of particular interest.

The game world may seem oddly related to the theme, but it comes in handy to use a device originally intended to play, which will be the cornerstone of this project. *Kinect*, is a revolution in the use of 3D scanners, because it offers a very affordable hardware that otherwise current people would not have had access, making way for an infinity of personal projects and improving the field of research.

Computer vision is closely linked to another field of robotics, used to analyze the environment around us, when the human being is not present and even when it is providing information that can not see or perceive.

Thus, the main objective is the search for a symbiosis, a synthesis of all these technologies, entering their problems and difficulties to solve a problem as well known, but seen from a very personal point of view, since it attempts to emulate driving in an original and fun way.

Keywords: vision, Kinect, scanner, 3D, robotics, vehicle, video game, autonomous vehicle, symbiosis, remote driving.

Acrónimos

Licencia GPL2 GNU Public License (GPL), versión 2

USB Universal Serial Bus

3D Tres dimensiones

BSD Berkeley Software Distribution

KDE es un proyecto de software libre para la creación de un entorno de escritorio e infraestructura de desarrollo para diversos sistemas operativos como GNU/Linux, Mac OS X, Windows, etc.

IDE Integrated Development Environment

CSS Cascading Style Sheets

HTML HyperText Markup Language

SDK Software Development Kit

SDRAM Synchronous Dynamic Random Access Memory

DDR2 Double Data Rate 2 memory

ARM Advanced RISC Machine

API Application Programming Interface

FreeBSD Free Berkeley Software Distribution

CAD Computer-Aided Design

RAM Random-Access Memory

JPEG Joint Photographic Experts Group

WPAN Wireless Personal Area Networks

MAC Media Access Control

IEEE Institute of Electrical and Electronics Engineers

Wi-Fi Wireless Fidelity

BT Bluetooth

Índice general

Índice de figuras

Capítulo 1

Introducción y objetivos

1.1. Introducción

La tecnología Kinect de Microsoft ha supuesto una total revolución en el mundo del videojuego, cambiando por completo el paradigma de juego, haciendo innecesario el uso de un mando o cualquier tipo de control remoto, pues permite al usuario interactuar con su propio cuerpo, mediante gestos o movimientos. Pero no todo queda en este campo, ya que la tecnología que incorpora ha sido un fuerte reclamo para investigadores, estudiantes y desarrolladores. Si bien es verdad que Microsoft lanzó en abril su propio entorno de desarrollo y, por tanto, el oficial, existen varias alternativas que comentaremos en los próximos capítulos.

En el campo de la construcción de sistemas robóticos existe un claro candidato por su simplicidad y por su disponibilidad en la universidad: Lego Mindstorms NXT, para el cual existen multitud de ejemplos y páginas web dedicadas, que pueden servir de inspiración.

La idea de realizar un proyecto de estas características surge del interés de intentar aplicar gran parte de lo aprendido durante la carrera y de complementarlo con aspectos complementamente ajenos para el autor, como es el campo de la visión por computador y de las comunicaciones *bluetooth*, pero sobre todo de hacer algo diferente, algo no al uso de lo que suelen ser los proyectos en informática (aplicaciones web, principalmente).

El objetivo es descubrir cómo relacionar y conectar tecnologías como *Kinect* y *Lego Mindstorms*, gracias al estudio de algoritmos de visión y de las comunicaciones inalámbricas, lo que implicará un estudio y construcción de una aplicación sencilla que utilice las características de este dispositivo.

1.2. Objetivos

El propósito de este proyecto es teleoperar un vehículo robótico construido con el set de piezas *Lego Mindstorms*, mediante una interfaz gestual, la cual traducirá los gestos del usuario en órdenes que se enviarán al robot vía *Bluetooth*

El objetivo no es tanto realizar una aplicación con un propósito general, que sea válido para cualquier tipo de robot, sino demostrar las capacidades de *Kinect* y de *Lego Mindstorms* y que los límites sólo los pone la imaginación. Indudablemente el código deberá ser reutilizable y/o adaptable para controlar cualquier tipo de robot o vehículo con unas características similares a lo que se pretende construir.

Más concretamente, en el proyecto se realizará un estudio detallado sobre *Kinect*, *Lego Mindstorms* y otras tecnologías relacionadas y se construirá una aplicación de demonstración del potencial de estas dos tecnologías. Para ello se aplicarán diferentes disciplinas, muchas de ellas vistas en la carrera, como son las comunicaciones inalámbricas, la visión por computador, etc.

Aunque no forma parte del objetivo principal del proyecto, se incluirá un apéndice con una pequeña investigación y demostración del mundo del modelado 3D utilizando Kinect, mostrando las posibilidades en este campo, que es quizás, el más atractivo e interesante para los desarrolladores. Y ya para finalizar, se intentará abrir nuevas líneas de investigación para el futuro.

1.3. Estructura de la memoria

Esta memoria se estructura en 5 grandes capítulos y varios apéndices. Los siguientes capítulos tratarán sobre el estado actual de las tecnologías o estado del arte y, posteriormente, se pasará a las fases de análisis y diseño.

Durante el análisis, se definirá el problema en cuestión, explicando bien cómo resolverlo dividiendo la estrategia en varios bloques fundamentales y para el diseño, se explicará la arquitectura hardware y software utilizada y se darán detalles de la construcción del vehículo (en el apéndice se incluye una más detallada).

Finalmente, tras los resultados y conclusiones obtenidos, se incluirán un apéndice, en el que el lector podrá consultar el manual de usuario, las instrucciones de montaje del vehículo, la instalación del dispositivo bluetooth (cómo realizar el pareamiento de claves) y, por último, una pequeña investigación sobre el mundo de las 3D con *Kinect*

Capítulo 2

Estado del arte

2.1. Resumen

Este capítulo recoge un estudio de las herramientas y tecnologías que serán utilizadas en el desarrollo del proyecto, así como un breve resumen sobre su futuro a corto y largo plazo. Además, se comentarán otros proyectos del estilo realizados hasta la fecha de publicación de este proyecto.

Como ya dijimos en el capítulo anterior, la idea principal es desplazar y controlar un vehículo de cuatro ruedas Lego Mindstorm a distancia, mediante una interfaz gestual, utilizando Kinect como receptor o captador de movimientos. Por supuesto, se dará un breve repaso a la librería de análisis de imagen OpenCV, la cual nos permitirá manipular las imágenes capturadas por Kinect y aplicar algoritmos que por su complejidad, habría sido imposible implementar en un tiempo razonable junto con toda la lógica de la aplicación.

En defnitiva, estos serán los puntos a tratar:

- Breve historia del *Kinect*
- Arquitectura interna
- Controladores o drivers para *Kinect*
- Lego Mindstorms
- Bibliotecas de referencia más utilizadas en este tipo de proyectos.
- Entornos de desarrollo disponibles
- Proyectos relacionados
- Hardware y software que utilizaremos

2.2. Kinect

2.2.1. Breve historia

Kinect es un dispositivo nuevo de juego introducido por *Microsoft* para la videoconsola *XBOX 360* y que permite al usuario interactuar con esta sin usar ningún tipo de mando o control y sin pulsar ningún botón.

En realidad, fue construído por la empresa israelí *PrimeSense*[1] y fue anunciado por primera vez en junio de 2009 bajo el nombre de *Project Natal* y aunque *Microsoft* se jacta de revolucionar el mercado de las tecnologías, pudo haber sido *Apple* quien se hiciera con esta, tan solo un año antes de dicho anuncio, pero numerosos problemas de confidencialidad y una obsesión por el secretismo por parte de *Apple* hicieron que *PrimeSense* probara suerte con su rival eterno.

En lo que a la historia del "hackeo"se refiere, fue un español quien consiguió acceder y controlar el *Kinect*; en noviembre de 2010, *Industrias Adafruit* ofreció una recompensa para un controlador de código abierto para *Kinect*. El 10 de noviembre, se anunció al español Héctor Martín como el ganador, que usó métodos de ingeniería inversa con *Kinect* y desarrolló un controlador para *GNU/Linux* que permite el uso de la cámara RGB y las funciones de profundidad.

Debido a esto, la empresa *PrimeSense*, se apresuró en sacar su propio controlador, que hasta el momento había mantenido en secreto y que incluye una funcionalidad mucho más amplia y precisa que el creado por Héctor Martín, proporcionando entre otras cosas un "SDKçompleto y funcionalidades muy útiles como la esqueletización.

En la actualidad, el aparato ha gozado de un gran éxito comercial y, sorprendentemente, no sólo en el mundo del videojuego, siendo éste objeto de numerosos estudios de carácter científico o personal, como el presente documento. En sí, Kinect compite con los sistemas *Wiimote* y *PlayStation Move*, que también controlan el movimiento para las consolas *Wii* y *PlayStation 3*, respectivamente.

2.2.2. Arquitectura interna

Kinect se basa en la detección de movimientos gracias al uso de dos sensores: una cámara de color y un sensor de profundidad. Así mismo, cuenta con un micrófomo, que permite la localización de la fuente acústica y el filtrado de ruido ambiente. El proyecto no requiere el uso del micrófono, así que nos centraremos solamente, en los otros dos elementos. El dispositivo también está pensado para poder orientarse en cualquier dirección.

[1] PrimeSense: `http://www.primesense.com/#2`

En la imagen 2.1 se puede apreciar la estructura interna del *Kinect*:

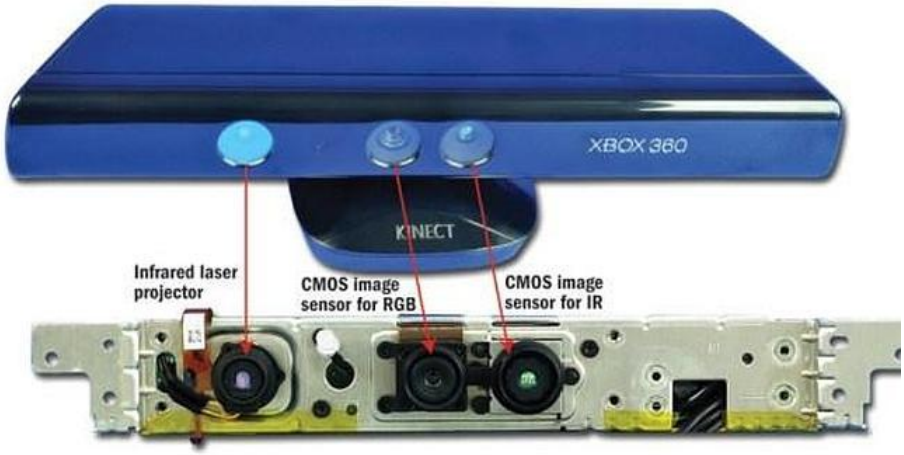


Figura 2.1: Componentes internos de Kinect

Y esquemáticamente, podemos ver cómo están relacionados sus componentes (2.2):

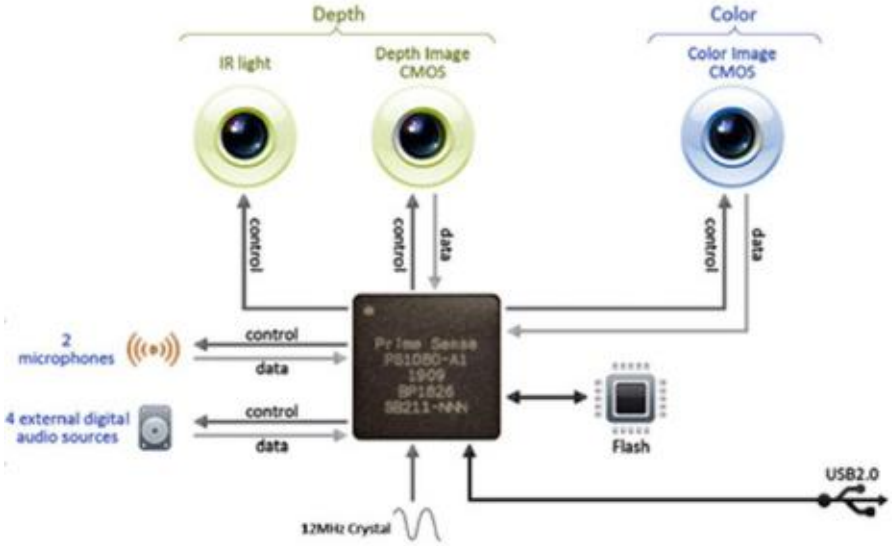


Figura 2.2: Componentes internos de Kinect

Como podemos ver, los dos sensores principales están situados en la parte frontal. La lente que se encuentra en el centro forma parte de la cámara a color y posee una resolución de 640x480 y los dos objetos dispuestos a cada lado de la cámara a color conforman el sensor de profundidad, que generará datos de profundidad. Estos dos objetos son: el emisor de infrarrojos (izquierda) y receptor (derecha de la cámara).

Los datos de profundidad se obtienen en base al tiempo que tarda en emitir el láser (opera en una frecuencia cercana al infrarrojo), rebotar en un objeto y llegar hasta el receptor. El siguiente esquema (figura 2.3) ilustra bien esta situación:

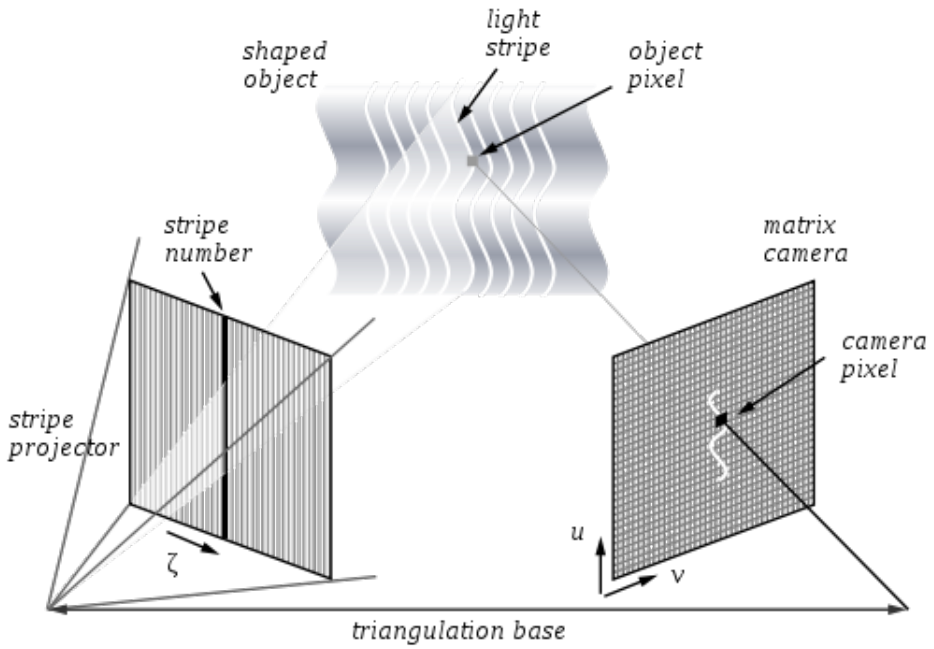


Figura 2.3: Esquema del sensor de infrarrojos

A continuación describimos cada componente hardware integrado en el *Kinect*:

- **Emisor láser:** Se trata de un objeto láser de clase 1 que usa un diodo láser de 830 nm sin modulación y nivel de salida constante. La potencia de salida medida a la salida del emisor es de alrededor 60 MW. La temperatura del láser se mantiene estabilizada gracias a un pequeño elemento *Peltier* (un pequeño refrigerador) montado entre el emisor y la placa de montaje. Este elemento puede calentar y enfriar el láser para mantener una temperatura constante, con el fin de estabilizar la longitud de onda del láser de salida, aunque a temperatura ambiente este se calienta poco. Existe una temperatura de corte no rebasable y no configurable para proteger el producto de 102°C.

- **Sensor de profundidad (receptor):** El sensor de imagen CMOS detecta los reflejos del haz infrarrojo, dentro de un patrón de puntos y segmentos, y genera mapas que informan sobre la intensidad de cada punto y segmento. Es un sensor de imagen monocromática y posee una tasad de refresco de 30Hz. La interfaz de control I2C del sensor realiza una transacción de un byte por cada frame o fotograma. También posee un filtro de luz infrarroja en la longitud de onda del láser emitido, el cual tiene una sensibilidad muy baja (mínima) a fuentes de luz visible.

- **Chip Marvell:** Este *chip* es para el procesamiento de audio, y no tiene nada que ver con el sistema de profundidad. Posiblemente se trate de uno de los chips de Marvell

ARMADA ARM serie, con velocidades de reloj de 400MHz a 1GHz. El reloj externo funciona a 26MHz. El chip possee una memoria SDRAM DDR2 de 512 Mbits y una memoria flash SPI que contiene el firmware cuyo código se asemeja al de un *ARM* de 32 bits. El Marvell también es responsable de la autenticación criptográfica del Kinect con respecto a la Xbox (mediante clave RSA y certificado), de esta forma Microsoft se asegura que el dispositivo sea original.

- **Ventilador:** El ventilador sólo se enciende cuando la temperatura interna supera los 70°C, medida cerca del conector del ventilador a la placa de la cámara y se alimenta de la fuente de 5Vdel conector USB .

- **Acelerómetro:** También cuenta con un acelerómetro *MEMS*, el *KXSD9* de Kionix. Debido a que la unidad tiene un rango (limitado) de movimientos, este acelerómetro forma parte del bucle de control de inclinación, que también incluye el controlador para motor paso a paso, el *A3906* de *Allegro Microsystems*.

- **Matriz de micrófonos:** Consiste en cuatro micrófonos que se utilizan para cancelar los ecos y el ruido de fondo, mientras que ayudan a determinar la procedencia de un ruido.

2.2.3. Características

Las características de *Kinect* son:

- Campo de visión horizontal: 57 grados
- Campo de visión vertical: 43 grados
- rango de inclinación física: +/- 27 grados
- Rango de profundidad del sensor: 1.2 – 3.5 metros
- Resolución profundidad: 320x240 a 16 bits de profundidad a 30fps
- Resolución color: 640x480 32-bit de color a 30fps
- Audio: Audio de 16-bit a 16 kHz
- Resolución de objetos en profundidad: Distingue objetos separados a partir de 1 cm
- Resolución de objetos a lo ancho: Distingue objetos separados a lo largo y a lo alto por una distancia mínima de 3 mm

Kinect devuelve datos “en bruto”de profundidad con valores que oscilan entre 0 y 2048.

2.2.4. Otros dispositivos: Asus XTION Pro Live

A mediados de este año, la empresa tecnológica *Asus* sacó al mercado lo que parece ser un claro competido del Kinect, pero con un precio, en principio, superior (175 euros). Este dispositivo es fruto del acuerdo entre *PrimeSense* (los creadores del Kinect) con dicha compañía, con la consiguiente compatibilidad de los drivers *OpenNI+NITE*. Fue durante la pasada edición del CES[2] que ASUS realizó una ambiciosa apuesta por sus sensores de movimiento *Xtion Pro* y *Wavi Xtion* y, apenas 7 meses después, ya tiene una nueva versión en el mercado. El *Xtion Pro Live* es un periférico que presume de ser compatible tanto con Windows como con Linux y que añade una cámara VGA y un par de micrófonos con respecto al modelo original. En principio, no se trata de un producto comercial al uso, sino que más bien está pensado para ayudar a los desarrolladores a diseñar mejores juegos y aplicaciones para el futuro *Wavi*, el modelo que sí tratará de encandilar al gran público en los meses venideros. El dispositivo de Asus luce así (figura 2.4):

Figura 2.4: Asus Xtion Pro Live

2.3. Controladores: OpenKinect, OpenNI+NITE, Microsoft Kinect SDK

En cuanto a los controladores, existen varias alternativas y multitud de extensiones y variantes, pero los *SDK* principales son tres: *libfreenect* (proyecto *OpenKinect*), OpenNI (de *PrimeSense*) y el *SDK* oficial de *Microsoft*. A continuación, describiremos cada uno con más detalle:

[2]Sitio web CES: `http://www.cesweb.org/`

2.3.1. OpenKinect: libfreenect

El proyecto *OpenKinect* surgió como una iniciativa de la empresa *Adafruit* ofreciendo una recompensa a quien lograra *"hackear"* por primera vez el dispositivo [3]. Como ya hemos dicho, fue el español Héctor Martín quien lo consiguió por primera vez y es hoy en día uno de los responsables del proyecto.

OpenKinect [4] es hoy en día una comunidad abierta de personas interesadas en dar uso al dispositivo de *Microsoft* con nuestros PCs y otros dispositivos. Trabajan creando bibliotecas libres, de código abierto y con soporte para Windows, Linux y Mac. La comunidad OpenKinect consta de más de 2000 miembros que aportan su tiempo y conocimientos al proyecto. El objetico principal actualmente es el software *libfreenect*, que es además, la biblioteca que usaremos para este proyecto. Todo código que utilice OpenKinect se pondrá bajo disposición de *Apache 2.0* o licencias GPL2 opcionales.

En concreto, el controlador *libfreenect* "simplemente"funciones de acceso a los recursos hardware del Kinect, es decir, los micrófonos, el sonido y las dos cámaras (profundidad y color). La elección de este controlador se tomó en base a que utilizando el de *PrimeSense* o el *SDK* de *Microsoft* no dejábamos espacio a la investigación del dispositivo en sus funciones más básicas y al aprendizaje de ciertas técnicas de visión, puesto que estas empresas ofrecen muchas otras funcionalidades además del acceso a los recursos, entre ellas, la famosa esqueletización y el seguimiento o *"tracking"* de objetos. Además, la instalación es relativamente sencilla y existen numerosos "wrappers.º adaptaciones a otros lenguajes de programación, pudiendo elegir el lenguaje de programación en el que queremos trabajar (todos hacen uso del código escrito en C). Luego, los lenguajes disponibles son:

- ActionScript.
- C++.
- C#
- Java
- Matlab
- OpenCV (escrita en C, pero utilizando objetos tipo IplImage, propios de OpenCV).
- Python
- Ruby

[3] Noticia concurso: `http://www.adafruit.com/blog/2010/11/04/`
[4] OpenKinect Project: `http://openkinect.org/wiki/Main_Page`

De los lenguajes conocidos por el autor (con conocidos me refiero a haber programado más de un año y tener cierta experiencia) están C, C++ y Java y en un principio se pensó utilizar este último, ya que es un lenguaje muy fácil de usar y al que el autor está muy acostumbrado, pero la lentitud demostrada al procesar las imágenes y no digamos ya intentar otro tipo de análisis, le hicieron decantarse por C++, que conserva la propiedad de la orientación a objetos de Java, descartando C por su complejidad a la hora de tratar temas de memoria. Los resultados obtenidos con C++ en cuanto a velocidad han sido sorprendetes.

Existen muchos proyectos realizados a día de hoy y pueden consultarse (y disfrutarse pues son código abierto y realmente muy buenos algunos) en Internet [5]

2.3.2. PrimeSense: OpenNI + NITE

Tras percatarse de la aparición del proyecto anterior, fruto del trabajo de la comunidad de *"hackers"* mundial[6], *PrimeSense* decidió liberar sus propios *drivers* para *OpenNI*, es decir, los controladores del propio creador del dispositivo. La tecnología de detección de movimiento para ponerla dentro de un Kinect, lanzó controladores de código abierto que funcionan muy bien con Kinect o con su kit de desarrollo propio, incluso con el nuevo dispositivo introducido por *Asus*.

PrimeSense se asoció con *OpenNI* y para la creación de su *SDK* de desarrollo. *OpenNI* es una organización sin ánimo de lucro creada para "promover la compatibilidad e interoperabilidad de dispositivos, aplicaciones y middlewar de Interacción Natural (IN)". Los controladores están disponibles en su sitio web [7]. La organización pone también a disposición un *"framework"* o entorno de trabajo de código abierto, el cual cubre todas las funcionalidades de bajo nivel (acceso a datos) y muchas otras de alto nivel.

Para garantizar la compatibilidad con las bibliotecas de *OpenNI*, *PrimeSense* creó *NITE*, que no es más que un conjunto de herramientas cuyo fin es crear combinaciones o secuencias de gestos con las manos. Realiza seguimiento sobre las manos, ya que se considera un gesto al movimiento de esta parte del cuerpo. Esta capa trabaja por encima de *OpenNI* y provee de implementaciones de todos los módulos de *OpenNI*. La arquitectura interna de NITE es la siguiente (fig. 2.5):

[5] Galería de proyectos: `http://openkinect.org/wiki/Gallery`
[6] Cómo se *"hackeó" Kinect*: `http://ladyada.net/learn/diykinect/`
[7] OpenNI: `http://www.openni.org/`

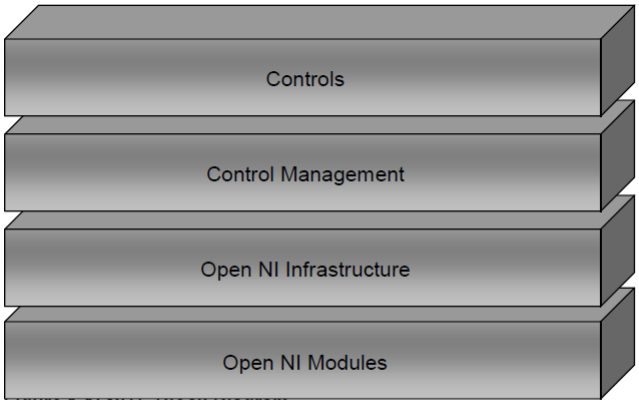


Figura 2.5: Diagrama de bloques NITE

Existen numerosos tutoriales, sencillos y con todo lo necesario para instalar y disfrutar de OpenNI+NITE [8]

2.3.3. Microsoft: SDK oficial

Nadie imaginó el enorme éxito que ha tenido Kinect, llegando a vender hasta la fecha X millones de unidades. Es por eso que *Microsoft* tampoco se quiso quedar atrás y sacó, por abril de este año su propio *SDK* de desarrollo, eso sí, en fase beta (en desarrollo).

Todo lo necesario se puede encontrar/descargar desde la web de *Microsoft* [9] y con respecto a sus competidores, suponemos que cuenta con perfecta integración con Visual Studio, Microsoft .NET y Windows. Al igual que *PrimeSense*, proporcionan también la funcionalidad de esqueletización. La ventaja principal está en que se instala muy fácil y rápidamente sin instalaciones complicadas (como ocurre con los otros dos anteriores).

2.4. Lego MindStorms

Lego Mindstorms es un juego de robótica para niños (y no tan niños) fabricado por la empresa *Lego*, el cual posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones, en forma interactiva.
Sólo la imaginación puede limitar lo que se va a construir, pues las combinaciones son prácticamente ilimitadas y todo está gobernado por un controlador que recibirá instrucciones.

[8] Tutorial: `http://golancourses.net/2011spring/02/09/a\-really\-simple\-tutorial\-for\-installing\-kinect\-openni\-on\-pc/`

[9] KinectSDK: `http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/`

Figura 2.6: Ejemplo Lego Mindstorms

La web *"NXTPrograms"* [10] es un sitio fantástico para tomar ideas y construir modelos ya programados y con instrucciones paso a paso.

2.4.1. Controladores ("brick")

Lego proporciona dos controladores (también llamados "bricks") diferentes, de los cuales hemos elegido el NXT, ya que era el único disponible en el laboratorio de trabajo. El NXT, es la versión mejorada del bloque RCX y puede programarse en multitud de lenguajes, siendo el más popular Objective C, que es el que proporciona Lego y Java, necesitando este último un firmware especial, con su propia máquina virtual dedicada.

[10] `http://www.nxtprograms.com/`

Figura 2.7: Comparación NXT y RCX

Existen también otros lenguajes para el NXT:

- NXT-G: Esta es la versión que lleva el NXT que se encuentra en los comercios y es la oficial de Lego [11].
- NBC (Next Byte Codes): es un lenguaje dirigido a programadores con una síntesis de lenguaje ensamblador.
- NXC: es un lenguaje de alto nivel similar a C. Utiliza el firmware original de LEGO y está disponible para Win32, Mac OSX y Linux.
- **leJOS NXJ**: leJOS NXJ facilita la programación del NXT con Java. Se trata de un completo firmware que sustituye el oficial de LEGO que funciona tanto en Windows como en Linux [12].
- leJOS OSEK (C/C++): consiste en un sistema operativo de tiempo real programable en ANSI-C y C++ [13].
- Python: Python es un lenguaje de programación de alto nivel fácil de aprender y muy extensible gracias a los múltiples módulos de los que dispone. Como es multiplataforma, no depende de un sistema operativo determinado para trabajar con el robot. Uno de

[11] Sitio oficial de Lego: `http://mindstorms.lego.com/en-us/support/files/default.aspx`

[12] LeJOS Java for Lego Mindstorms: `http://lejos.sourceforge.net/`

[13] leJOS OSEK Project: `http://lejos-osek.sourceforge.net/`

los módulos disponibles es *NXT Python*, que permite controlar el NXT sin necesidad de modificar el firmware original de Lego.

- Otros: LabVIEW, URBI, Robolab 2.9, NXT# - MindStorms for .NET, Logo for NXT, Matlab, Simulink, RoboRealm, etc.

La universidad disponía de un bloque NXT con el firmware estropeado y se decidió instalar el que lleva Java, es decir, **leJOS NXJ**, por razones de comodidad a la hora de programarlo y por la gran cantidad de ejemplos que hay en internet.

2.4.2. Otros componentes

Para el movimiento de los robots, Lego Mindstorms pone a nuestra disposición servomotores ajustables por ángulo de giro y también multitud de sensores con los que interactuar con el entorno, como sónares, detectores de color, micrófonos, etc.

El controlador NXT dispone de batería autónoma (pilas) y comunicación via Bluetooth y USB.

2.5. Bibliotecas de referencia

En el marco de las bibliotecas de referencia, habaremos de las que más impacto pueden y van a tener en el proyecto, en relación a las técnicas de visión, de construcción de interfaces gráficas y, por qué no, de recreación de entornos 3D (aunque esta última no tenga relación, queda vistoso intentar una recreación en tres dimensiones del entorno).

2.5.1. OpenCV

Dado que el proyecto tiene una componente de análisis de imagen, se hace necesario el uso de una buena biblioteca de visión por computador, con el fin de tener una buena cantidad (y también calidad, funciones que están sobradamente probadas y diseñadas para ser más eficientes en cuanto a recursos y tiempo se refiere) de algoritmos y para no andar "reinventando la rued".

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por *Intel*. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta software de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se dio bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estereo y visión robótica y "tracking.º seguimiento de objetos (Filtro de *Kalman*).

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multi núcleo. *OpenCV* puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel (IPP).

Para más información o para descargar la biblioteca, se puede consultar su sitio web [14]. Al igual que pasaba con *libfreenect*, existen multitud de *wrappers* para que el lenguaje no sea una limitación y sí una alternativa.

2.5.2. Qt

Qt es una biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con una interfaz gráfica de usuario así como para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores. Es producido por la división de software *Qt de Nokia*, que entró en vigor después de la adquisición por parte de *Nokia* de la empresa noruega *Trolltech*, el productor original de *Qt*, el . *Qt* es utilizada en KDE, un entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros. *Qt* utiliza el lenguaje de programación *C++* de forma nativa y también es usada en sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos.

2.5.3. OpenGL

OpenGL (*Open Graphics Library*) es una especificación estándar que define un API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por *Silicon Graphics Inc. (SGI)* en 1992 y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo. También se usa en desarrollo de videojuegos, donde compite con *Direct3D* en plataformas *Microsoft Windows*.

[14] OpenCV: `http://sourceforge.net/projects/opencv/`

2.6. Entornos de desarrollo (IDEs)

Este es quizás, uno de los puntos que más se deben tener en cuenta, pues puede condicionar la forma de trabajo y facilitarnos o dificultarnos nuestra labor. Afortunadamente, contamos con variedad de IDEs con características muy completas que no harán otra cosa que allanar el camino (sobre todo temas de compilación, referencias a librerías, etc.).

2.6.1. QT Creator

Qt Creator es un *IDE* creado por *Trolltech* para el desarrollo de aplicaciones con las bibliotecas*Qt*, requiriendo su versión 4.x. Los sistemas operativos que soporta en forma oficial son:

- GNU/Linux 2.6.x: para versiones de 32 y 64 bits con Qt 4.x instalado. Además hay una versión para Linux con gcc 3.3.
- Mac OS X 10.4: o superior, requiriendo Qt 4.x
- Windows XP y superiores: requiriendo el compilador MinGW y Qt 4.4.3 para MinGW

Este *IDE* es fantástico pues permite construir potentes interfaces gráficas configurables en todos los sentidos en muy poco tiempo y garantiza una muy buena estabilidad y tiempos de respuesta. Además posee un fichero de configuración donde poder referenciar fácilmente qué librerías queremos utilizar o referenciar, qué archivos forman parte del proyecto (en caso de que no los haya agregado el solo) y un directorio de recursos donde poder introducir imágenes o lo que queramos. Un aspecto a destacar es la personalización de los componentes de la interfaz mediante hojas de estilo, de forma parecida a como lo hace CSS para HTML. Una limitación es que sólo está pensado para programar en C++ (también en C, menos las librerías Qt que son sólo C++).

2.6.2. Netbeans

Netbeans es el *IDE* por excelencia e integra múltiples lenguajes en un único editor. Esto es un punto a destacar de cara al desarrollo del proyecto, ya que muy posiblemente el código del programa que va en el ordenador estará realizado en un lenguaje diferente al del que vaya en el controlador de *Lego*. Como carencia, señalar el pobre editor de interfaces gráficas que tiene y que además es sólo para Java.

2.6.3. Eclipse

Otro clásico de los *IDEs*, que ofrece las mismas posibilidades que Netbeans, salvo la del diseño de interfaces gráficas mediante un módulo gráfico, a menos que sea por un plugin y que suelen ser de pago. Como ventajas o desventajas no aporta ninguna con respecto a Netbeans, así que le consideraremos al mismo nivel.

2.6.4. Geany

Geany es un *IDE* muy ligero, prácticamente un editor de textos que ofrece subrayado de sintaxis, autocompleción, integración de la línea de comandos. Para hacer cambios rápidos o proyectos sencillos es bastante cómodo pues no necesita cargar muchos recursos y ocupa poco espacio en memoria.

2.7. Proyectos de referencia

En realidad, la idea de controlar un vehículo a distancia es anterior a decidir usar *Kinect* y de mirar cualquier otro tipo de ejemplo en Internet. La web de OpenKinect proporciona muchos buenos ejemplos donde poder inspirarse, por no decir ya si buscamos en plataformas de vídeo online como *Youtube* o *Vimeo*. Muestro tres como referencia, pero existen infinidad de aplicaciones [15].

2.7.1. RGBDemo v.0.4

El primer ejemplo que pude probar, fue una simple demo de la imagen de profundidad, que venía con el driver de OpenKinect, en los laboratorios de robótica de la universidad y con mi propio ordenador; pero más tarde descargué el programa *RGBDemo*, cuyo autor pude conocer en persona, ya que trabaja en la universidad (Nicolas Burrus [16]) y me dio ciertas pautas para poder continuar con el proyecto. Este programa, aprovecha el potencial de Kinect para hacer una reconstrucción tridimensional del entorno e incluye los ficheros de calibración para hacer corresponder los puntos de la imagen a color con los de la imagen de profundidad.

[15] KinectHacks: `http://www.kinecthacks.com/`

[16] Web de Nicolas Burrus: `http://nicolas.burrus.name/index.php/Main/HomePage`.

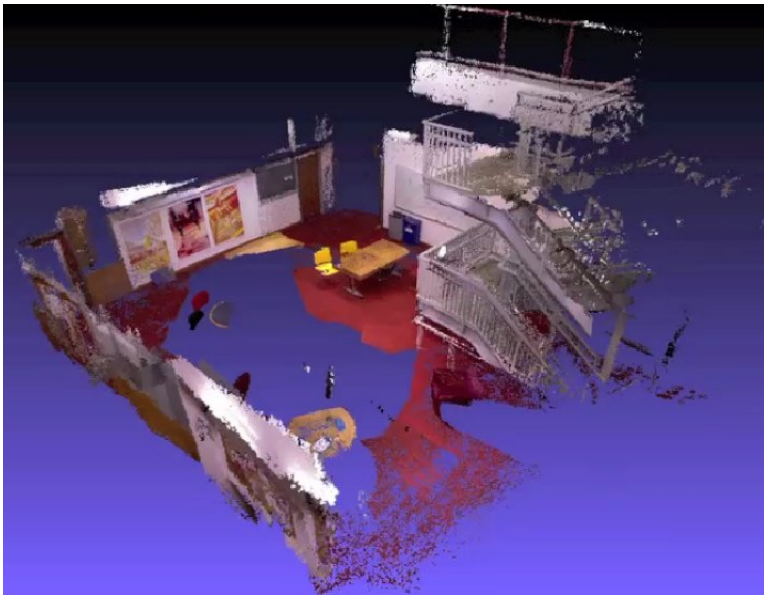


Figura 2.8: Recreación en 3D de una habitación

2.7.2. KinectFusion HQ

Este es uno de los mejores proyectos [17] realizados hasta la fecha con *Kinect* y otras tecnologías de visión y modelado 3D. Permite, al igual que el anterior, reconstruir en tiempo real cualquier espacio por el que vaya pasando el *Kinect* y genera un mapa tridimensional con texturas mapeadas sobre la imagen y con efectos de realidad aumentada como (físicas de partículas sobre objetos reales del entorno, es decir, detecta los objetos y las partículas chocarán o rebotarán en consecuencia).

2.7.3. Proyectos parecidos a este en concreto

Existía ya un proyecto parecido a este, pero mucho más simple, pues consiste en controlar un vehículo de dos ruedas utilizando los brazos, que modificarán la aceleración de la rueda que les corresponda en función de la profundidad. Utiliza OpenNI y C++ [18]. Este *"hack"* fue descubierto posteriormente a haber definido ya la forma de control de nuestro vehículo, así que podemos decir que quedó descartado como "fuente de inspiración".

[17] Para más información: `http://www.youtube.com/watch?v=RSh8Voanp3c`

[18] Vídeo: `http://kinect.dashhacks.com/kinect-hacks/2011/03/25/kinect-gesture-controlled-lego-mindstorms-tribot\%E2\%80\%AC`

Capítulo 3

Análisis del sistema

3.1. Introducción al análisis

Una vez puestos al día en lo que al estado de las tecnologías se refiere, conviene pasar a la fase de análisis del sistema, en la cual definiremos claramente el problema a resolver, las posibles estrategias y las técnicas existentes para este tipo de problemas. Dado que el sistema final está determinado por el tipo de vehículo, tomaremos como referencia dos modelos de coches y estudiaremos las ventajas y desventajas que aporta cada uno, así como las posibles formas de control, que también marcarán la interfaz de usuario. En resumen, se puede decir que la elección del vehículo determina la interfaz gestual y esta la experiencia de control del usuario.

3.2. Descripción del problema a resolver

Si tuviéramos que dar una definición sucinta y clara del objeto de este proyecto, podríamos decir que tratamos de construir un sistema de teleoperación via Bluetooth de un vehículo *Lego Mindstorms*, mediante una interfaz gestual, aprovechando el potencial de la tecnología *Kinect*. En otras palabras, controlar un coche *Lego* a distancia con nuestro propio cuerpo, transmitiendo los datos por un canal de radio *Bluetooth*.

Con interfaz gestual, nos referimos a un programa que, utilizando una cámara o cualquier otro dispositivo al uso, capture, analice y procese nuestros movimientos, identificando bien de qué parte del cuerpo se trata. Si bien podíamos haber utilizado una webcam cualquiera (el proceso de análisis hubiera sido mucho más complicado), se escogió finalmente *Kinect* como dispositivo de captura, por su condición novedosa y por el abanico de posibilidades que nos ofrece.

A continuación, se propone un esquema gráfico de lo que se plantea construir.

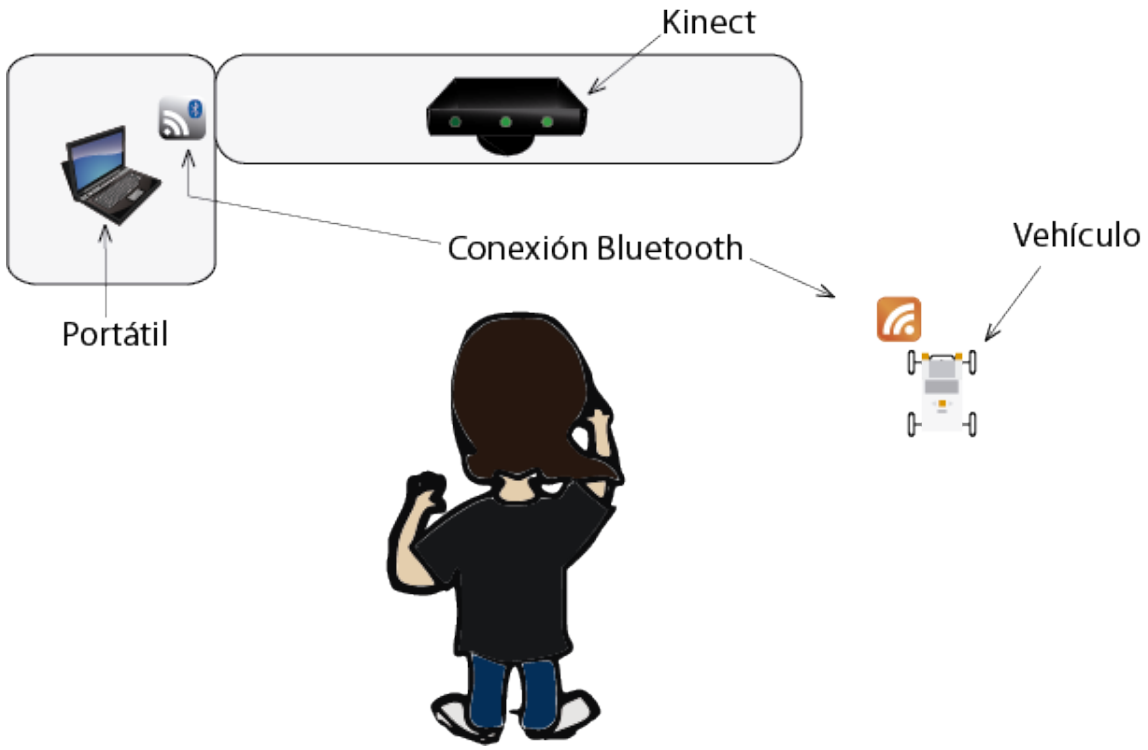


Figura 3.1: Esquema gráfico del problema a tratar

El esquema sugiere dividir el problema en tres partes principales: una dedicada al análisis de las imágenes capturadas, otra a la definición de los gestos y, por último, una para la comunicación entre el portátil y el vehículo, la cual debe ser en tiempo real, para garantizar un control fluido y en un solo sentido, pues solamente se van a enviar órdenes vía radio y no es necesario confirmarlas (penalizaría la respuesta del coche). En las siguientes subsecciones, se analizarán las posibles formas de control del vehículo (proponiendo tres modelos de coches y descartando uno de ellos), las técnicas de visión más adecuadas a nuestro caso y un esquema de comunicación más detallado.

El objetivo es constuir un vehículo completo, de tres o cuatro ruedas, con una interfaz gestual original y compleja, que implique técnicas de visión más avanzadas.

3.3. Modelos de control gestual

En el ámbito de los modelos de control, se nos ocurre plantear tres, para dos tipos de coches diferentes. El primero de ellos consistirá en un vehículo de tres ruedas, una de ellas libre rueda loca) y el segundo en uno de cuatro ruedas, todas ellas controladas por servomotores. Cada modelo debe presentar un control intuitivo, es decir, una serie de poses, gestos o movimientos que podríamos aplicar perfectamente si hubiese un interfaz físico (un volan-

te, joystick, panel de teclas, etc.). Cada modelo irá acompañado de una imagen a modo de esquema que nos dará una buena idea de lo que pretende.

3.3.1. Primer modelo

El primer modelo centra su diseño en un vehículo de tres ruedas. Las dos primeras se mueven gracias a dos servomotores, uno por cada una y una tercera que consiste en una "rueda loca", la cual tiene libertad de giro y ningún servomotor. En la figura se muestra, a la izquierda, un esquema simplificado del control del vehículo y, a la derecha, la arquitectura del coche.

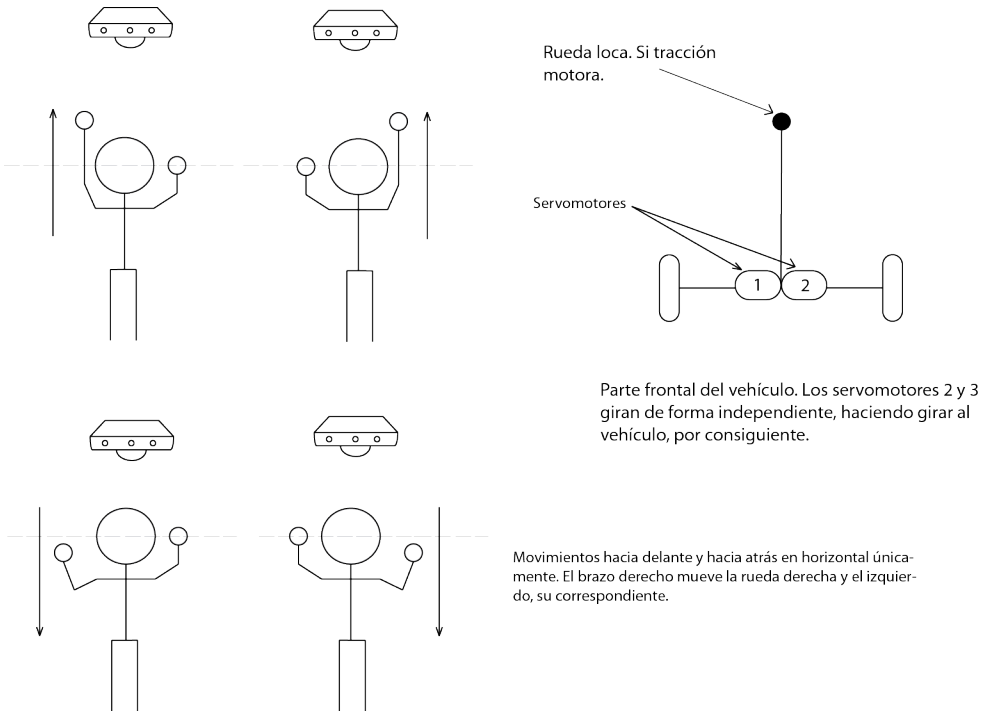


Figura 3.2: Primer modelo

El control es sencillo, pues cada brazo controla la velocidad de una de las ruedas. La velocidad dependerá de cuán cerca esté la mano del *Kinect*, aumentando a menor distancia de la fuente y disminuyendo conforme la acercamos a nuestro cuerpo. Si aceleramos más la rueda de la izquierda, o lo que es lo mismo, movemos el brazo iquierdo hacia delante y el derecho en el otro sentido (más pegado al cuerpo), entonces provocaremos una fuerza que hará que el vehículo gire a la derecha y viceversa.

- **Ventajas**
    - Intuitivo: Sólo son posibles dos gestos: mover brazo izquierdo y mover brazo derecho y resulta muy natural controlar el vehículo de esta forma.

- Facilidad de programación: La programación es sencilla, ya que sólo hay q detectar la mano izquierda y la derecha y obtener su profundidad para acelerar las ruedas.
- Construcción del vehículo sencilla: El vehículo requiere pocas piezas para funcionar y se pueden encontrar las instrucciones en varias páginas web. Además, montarlo no requiere ningún tipo de pericia por parte del usuario.

- **Inconvenientes**
  - Objetivo del proyecto: Además de ser una ventaja, supone al mismo tiempo un inconveniente, en el sentido de que necesitamos una aplicación basada en Kinect, sobre la cual podamos aplicar otras técnicas relacionadas con la visión que vayan más allá de una sencilla segmentación por profundidad.

Esta opción quedó descartada, ya que a pesar de ajustarse a las necesidades del proyecto, no permite profundizar más otros campos, como es de la visión, pero sirvió de base y estudio para los dos modelos que se explicarán a continuación.

3.3.2. Segundo modelo

Este segundo modelo supone un paso intermedio en la búsqueda de un sistema que se adapte a los objetivos del proyecto y que además nos permita, como ya venimos diciendo, profundizar en otras áreas. En este caso el vehículo dispone de cuatro ruedas, las dos traseras cada una con un servomotor independiente y las dos delanteras gobernadas por un mismo motor. A pesar de que las dos ruedas traseras cuentan con un servomotor independiente, es lógico pensar que se moverán al mismo tiempo, pues un único motor para las dos induciría menos potencia. De nuevo, el gráfico muestra, a la izquierda, una idea de cómo controlar el coche y, a la derecha, el esquema interno del vehículo.

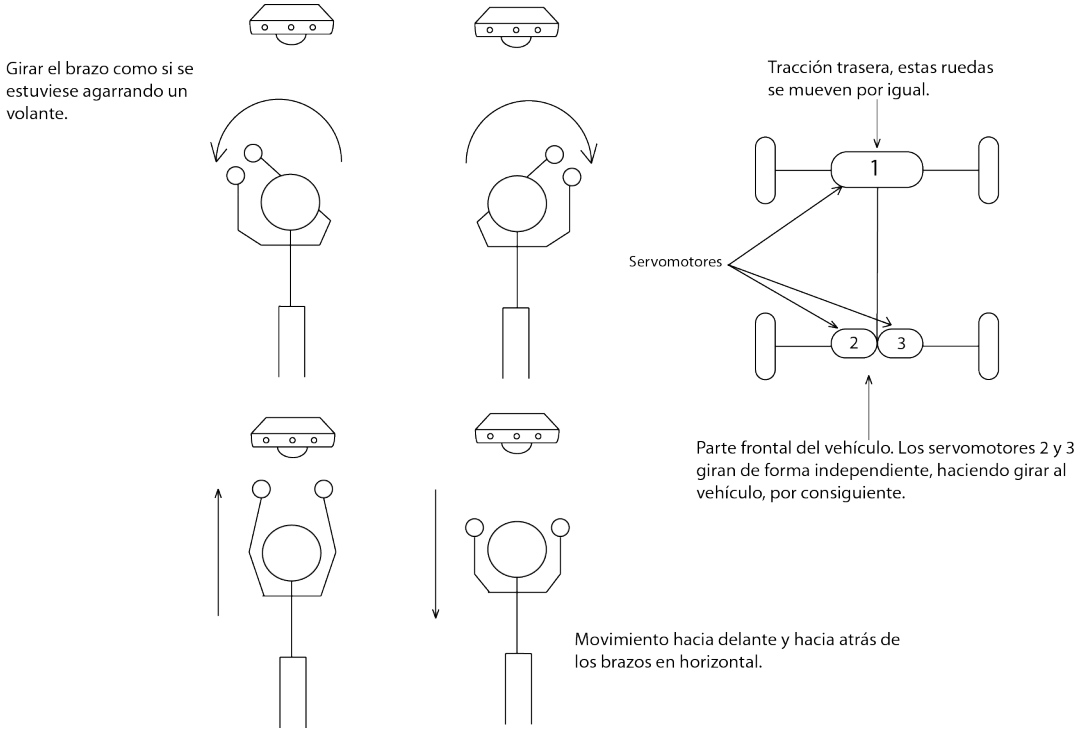


Figura 3.3: Segundo modelo

Como podemos ver, el control se asemeja al del timón de un avión, donde alejar o acercar el mismo se traduce en un aumento o decremento de velocidad, respectivamente. Del mismo modo, mover el brazo izquierdo hacia la derecha en arco produce un giro hacia la derecha y viceversa para el otro brazo. Esta técnica es más compleja, no ya desde el punto de vista de la visión, que es igual que el caso anterior, si no de la geometría del problema, pues habrá que calcular ángulos y posiciones relativas entre los brazos. Como todo sistema, tiene sus ventajas e inconvenientes, por supuesto vistos desde el punto de vista del autor y conforme a los objetivos del proyecto.

- **Ventajas**
  - Intuitivo: Existen cuatro movimientos básicos ahora, que se pueden combinar de forma que avance y gire al mismo tiempo. El control es intuitivo porque recuerda al de un coche o al de un avión.
  - Facilidad de programación: Al igual que el anterior, la programación no requiere mucha complicación, salvo el cálculo de giro y la posición relativa de un brazo respecto del otro.
  - Interfaz gestual más compleja: Lo que ofrece un control más realista.
- **Inconvenientes**

- Más gestos: El control reconoce más gestos, lo que requiere un poco más de práctica.
- Objetivo del proyecto: De nuevo, el proyecto se queda corto en cuanto a las técnicas de visión que se pueden emplear, pues como mucho se podrá aplicar una segmentación por profundidad y color y técnicas de seguimiento de objetos.
- Construcción del vehículo compleja: La construcción del vehículo es más compleja, requiere de una guía de montaje [1] y mucha paciencia.

Así pues, este modelo ofrece un diseño de vehículo más completo, pues se asemeja más a un coche real, pero sigue implicando pocas técnicas de análisis visual, por lo que se descartará el modelo de control. A continuación, propondremos una versión final que implicará este modelo de vehículo junto con un sistema de control totalmente original y que da lugar a nuevas técnicas de visión.

3.3.3. Tercer modelo: opción final

Continuando el apartado anterior y conservando la estructura del vehículo, surge la necesidad de imaginar una nueva interfaz gestual, que implique no solo un simple movimiento de brazos, sino también, por ejemplo, los dedos de la mano. Además, cada mano controlará un aspecto distinto del vehículo (velocidad y giro y no todo como venía ocurriendo hasta ahora (con todo nos referimos a que un brazo podía controlar velocidad y giro al mismo tiempo).

La idea de utilizar los dedos en el control viene de indicar la velocidad o "marcha"dependendiendo del número de dedos extendidos. Esta técnica recuerda a la caja de cambios manual de un coche, pero con la diferencia de que no se nos va a calar el coche. Así pues, se hace necesario idear una técnica de reconocimiento de dígitos en imágenes. Por supuesto, esto implica utilizar las ya mencionadas técnicas de segmentación.

El objetivo también es aplicar un seguimiento o *"tracking"* a las manos, de forma que el sistema sepa siempre qué objeto en la imagen se corresponde con qué mano.

[1] Guía de montaje `http://www.nxtprograms.com/NXT2/race_car/index.html`

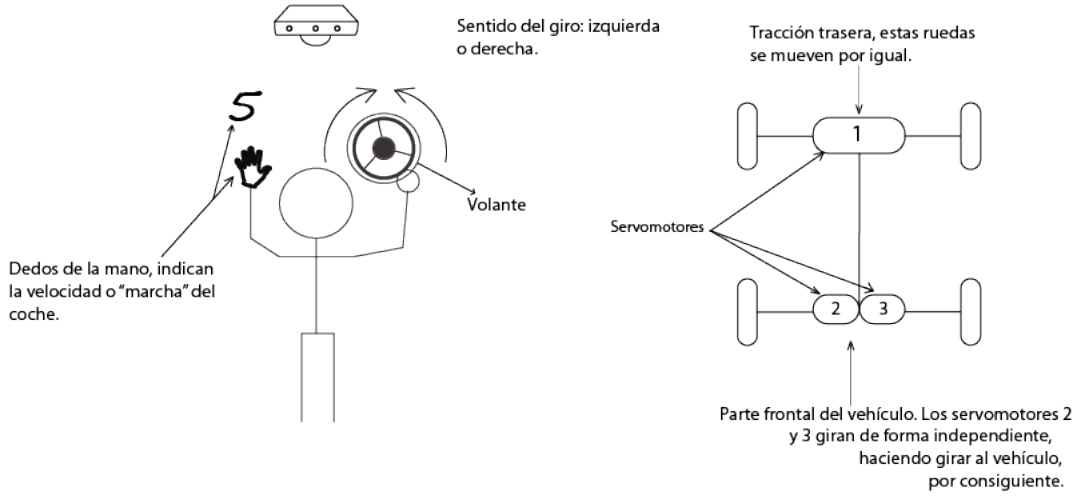


Figura 3.4: Esquema definitivo

En el esquema podemos ver como el usuario tiene la mano abierta, lo que significa que está transmitiendo la quinta marcha al vehículo, es decir, la más rápida si consideramos una escala del uno al cinco. En cuanto al giro, la idea más lógica de control para una única mano es la de trazar arcos sobre el centro del volante, que actúa a modo de referencia. Más tarde, en el capítulo dedicado al diseño, explicaremos en detalle cómo se extraen los dedos y el ángulo de giro.
Las ventajas e inconvenientes de este modelo son:

- **Ventajas**
  - Ajustado a los objetivos: La propuesta requiere de un mayor nivel de complejidad en la programación y unido a un diseño de vehículo realista, hacen de este modelo el candidato ideal.
- **Inconvenientes**
  - Menos intuitivo: El control no es tan intuitivo como en los otros dos casos; el usuario tiende a mover los dos brazos a la vez con la esperanza de controlar el vehículo, pero con un poco de práctica al final resulta práctico y divertido.
  - Complejidad: La programación se vuelve más compleja, pues habrá que idear la forma de extraer los dedos extendidos de un mano y aplicar técnicas de *"tracking"* sobre objetos en movimiento.

Resumiendo estas tres propuestas de control y, por tanto, del diseño del vehículo, concluimos que la mejor se adapta a nuestras necesidades es esta última de forma tal que nos va a permitir investigar en el mundo de la visión, las comunicaciones y, por qué no, de la mecánica.

3.4. Comunicaciones

La comunicación entre el portátil y el vehículo es, al igual que la interfaz gestual y el análisis visual, uno de los pilares de este proyecto. Es fundamental asegurar una buena comunicación y en tiempo real que no limite el control.

Lego Mindstorms NXT 1.0 ofrece dos vías de comunicación: cable *USB* y *Bluetooth*, cuyas ventajas e inconvenientes se analizarán a continuación. Antes que nada, daremos un breve repaso a la unidad fundamental de comunicación entre aplicaciones, el socket: software objeto que conecta una aplicación a un protocolo de red.

Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos

- Que un programa sea capaz de localizar al otro.
- Que ambos programas sean capaces de intercambiarse cualquier secuencia de octetos, es decir, datos relevantes a su finalidad.

Para ello son necesarios los tres recursos que originan el concepto de socket:

- Un protocolo de comunicaciones, que permite el intercambio de octetos.
- Un par de direcciones del protocolo de red (dirección IP, si se utiliza el protocolo TCP/IP), que identifican la computadora de origen y la remota.
- Un par de números de puerto, que identifican a un programa dentro de cada computadora.

Los sockets permiten implementar una arquitectura cliente-servidor. La comunicación debe ser iniciada por uno de los programas que se denomina programa “cliente”. El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa “servidor”. En este caso, es indiferente quien inicie la conexión y, por tanto, sea el servidor, así que se tomará como cliente al programa contenido en el ordenador y como servidor al programa del vehículo, que esperará a que el ordenador tenga la aplicación lanzada y listo para transmitir datos.

3.4.1. Cable USB

La tecnología *USB* (*Universal Serial Bus*), consiste en un interfaz físico o puerto que sirve para conectar periféricos a un ordenador. Tenemos la posibilidad de conectar el cable *USB* del vehículo al ordenador, aunque como ya podemos prever, esto supone una gran limitación al movimiento, pues depende de la longitud del cable y de que no se cruce en el camino del coche. A su favor, podemos decir que la velocidad de transmisión es muy alta, entre 1.5 y 480 Mbits por segundo para las versiones anteriores a la 3.0, siendo esta diez veces mayor. Además, la transmisión es más fiable y rápida que por *Bluetooth*, por ejemplo.

3.4.2. Bluetooth

Bluetooth es una especificación industrial para redes inalámbricas de área personal (WPANs) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda *ISM* de los 2,4 GHz. La principal ventaja que aporta frente al cable USB está en el propio concepto: es una tecnología inalámbrica. Esto nos garantiza libertad absoluta de movimiento. Si bien, también tiene limitaciones en cuanto a la distancia, pues dependiendo de la clase del dispositivo tendrá un alcance u otro. Dado que el estar de pie, presente frente al Kinect es una limitación de por sí (no podemos seguir al coche con el portátil y el Kinect en la mano), parece adecuado limitar el rango de acción del dispositivo a diez metros, es decir, utilizar un dispositivo de clase 2, con una potencia de 4dBm. y 2.5MW.

Aunque ya se tenía clara la tecnología de comunicaciones que se iba a utilizar, la limitación física que impone el cable USB es demasiado grande, por lo que el *Bluetooth* se vuelve imprescindible (ya que no disponemos de WIFI y tampoco interesa un rango de acción muy grande).

El protocolo implementado para la transmisión de los datos se describirá con más detalle en el siguiente capítulo, aunque ya podemos detallar que para establecer una conexión será necesario conocer la dirección física del *brick* de *Lego*, es decir, la dirección MAC (*Media Access Control*, que consiste en un identificador de cuarenta y ocho bits (tres bloques hexadecimales), que identifica de forma única a una tarjeta o dispositivo de red. Los primeros veinticuatro bits los configura el IEEE y los veinticuatro últimos, el fabricante.

Como protocolos de transporte, existen dos grupos, *L2CAP* y *RFCOMM*, pero utilizaremos este último por ser el más sencillo, ya que consiste en un conjunto simple de protocolos de transporte, construido sobre el protocolo anterior y que proporciona sesenta conexiones simultáneas para dispositivos BT emulando puertos serie RS-232. El protocolo está basado en el estándar *ETSI TS 07.10.*

3.4.3. Protocolo de envío

Independientemente del medio de transmisión, se debe pensar/definir un protocolo que dicte unas normas a la hora de enviar los datos por un canal de comunicación. El problema que nos atañe no tiene mayor complicaciones en este sentido, pues sólo habría que enviar la velocidad del vehículo y el giro, por lo que el protocolo o más bien "miniprotocolo" resula trivial.

Así pues, se define de la siguiente manera (los datos que se enviarán):

$$*velocidad\#giro@$$

Donde "velocidad" y "giro" son los parámetros a enviar.

3.5. Técnicas de visión

En este apartado se explicarán las técnicas más sencillas y habituales en lo que al análisis de imagen se refiere y se propondrá una estrategia para responder al **último modelo de control gestual** (tercer modelo) propuesto en secciones anteriores y se complementará en la fase de diseño del sistema.

3.5.1. Obtención de imágenes digitales

Mientras que las imágenes analógicas que se obtienen con una cámara son imágenes continuas, una imagen digital en dos dimensiones es una imagen discreta, por lo tanto, el primer paso una vez se obtiene la imagen es la digitalización. Por este motivo, una imagen se representará como una matriz donde cada elemento corresponde a un valor de iluminación.

En este caso se tratará obviamente con imágenes digitales, pero la imagen no provendrá de una cámara al uso, sino de la reconstrucción a partir de los datos de profundidad obtenidos del sensor láser. Dicho de forma gráfica, *Kinect* obtiene estos datos de la siguiente forma:

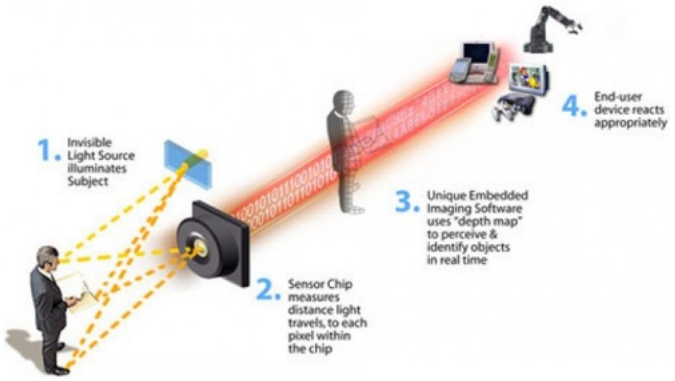


Figura 3.5: Funcionamiento interno de Kinect

Este dibujo, explica el funcionamiento interno del Kinect y el paso previo que supone a la obtención de la imagen, devolviendo una matriz de datos puros o en bruto (*"raw data"*). Esos datos deberán ser normalizados para poder construir una imagen en escala de grises. Estos valores tienen un rango limitado y se sitúa entre 0 y 2047 y para obtener una imagen en escala de grises, deberá aplicarse un proceso que se denomina "normalización".

El objetivo, para este campo del proyecto, **es aislar las dos manos de la escena**, aprovechando estas características, de forma que se obtenga algo parecido a lo siguiente:

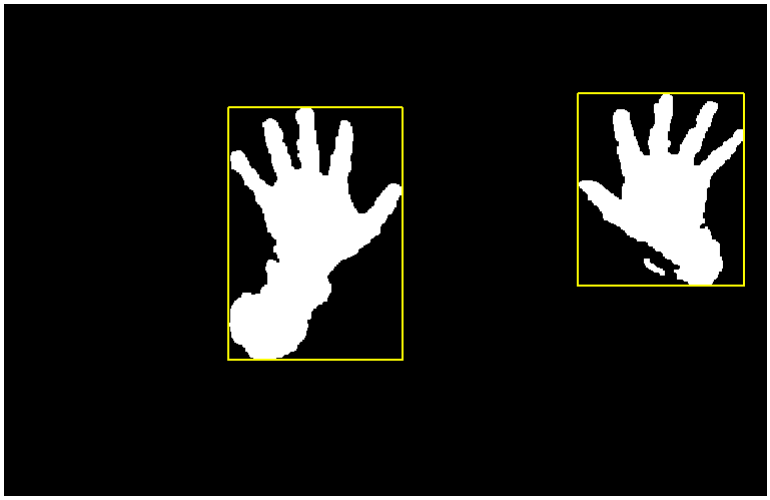


Figura 3.6: Segmentación de las manos por color

Todo lo que estuviera por detrás de las manos en la figura, tendrá un nivel de gris superior al definido como umbral de segmentación. Los valores más altos tras normalizar la imagen (entre 0 y 255) se mostrarán más claros (255 = blanco) y los más pequeños, aparecerán próximos al negro.

3.5.2. Técnicas habituales

Como ya hemos dicho antes, **una imagen digital consiste en una matriz** donde cada elemento se corresponde con un valor de ilumación, o lo que es lo mismo, con un color. Matemáticamente:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Donde *x* e *y* representan un punto *(x, y)* en la imagen, *M*, *N* el número de filas y columnas, respectivamente y *f(x,y)* una función de color para un punto dado.

Antes de comenzar a describir las técnicas más habituales, conviene ponerse al día de los términos más básicos:

- **Píxel:** Se trata de la unidad mínima de información que compone una imagen digital. No existen variaciones de color dentro de un píxel, pero sí que aporta información sobre su posición dentro de la imagen. El tamaño de un píxel depende de la resolución de la imagen.
- **Resolución:** Se trata de la cantidad de píxeles por unidad de medida.
- **Profundidad de color:** Este concepto está relacionado con el número de bits asignados a cada píxel, es decir, es proporcional al número de colores que pueden ser asignados a un píxel de la imagen. En la imagen normalizada que buscamos, la profundidad será de 255, pues sólo existen 255 tonos de gris.
- **Espacio de color:** Hay diferentes tipos de escalas que ayudan a determinar el color final de un píxel; se trabajará principalmente con dos: la escala RGB y su variante BGR, esta última para conversiones entre formatos de imágenes, ya que *Qt* utiliza este formato.

Repasados estos conceptos, damos paso a las técnicas más habituales en proyectos de visión, las cuales nos permitirán obtener imágenes como la de la figura anterior.

3.5.3. Espacio de color RGB

Se define el color de un píxel como la composición de las intensidades de los colores primarios que lo forman (Red, Green, Blue). Para indicar con que proporción se mezclan los colores se asigna un valor a cada color, por ejemplo el valor 0 indicaría que el color no interviene en la mezcla, mientras que si cada color primario se codifica con 8 bits, el 255 significaría que el color interviene de manera total en la mezcla. Así por ejemplo, el color (255,0,0) sería un rojo puro, el (255,255,255) sería composición total de los tres colores, cuyo resultado es blanco, y el (0,0,0) sería el color negro.

Normalización

Normalmente, la normalización se hace necesaria, para tener una cierta independencia de las propiedades de la imagen, como lo son el brillo y el contraste, y así poder comparar huellas por su índice de calidad (QI), pero en este caso, al no tratar con imágenes a color obtenidas de una cámara, el único objetivo de la normalización es poder representar los valores proporcionados por *Kinect* entre 0 y 2047 como tonos de gris, es decir, como valores entre 0 y 255. Se trata de un proceso matemático sencillo y muy habitual. Las ecuaciones que nos permiten esto son las siguientes: Sea *I(x,y)* la imagen de entrada,

$$N(x,y) = \frac{(N^{\circ}Nivelesgris - 1)}{(\text{máx}\,(I) - \text{mín}\,(I))} * (I(x,y) - \text{mín}\,(I))$$

Donde:

- **I(x,y)**: valor en bruto o puro (entre 0 y 2047), obtenido de la imagen en la coordenada (x,y).
- **min(I),max(I)**: mínimo y máximo nivel de valores en la imagen respectivamente.
- **N(x,y)**: nivel de gris de la imagen normalizada en la coordenada (x,y).

Si ya se conoce el valor máximo (2047) y mínimo (0), entonces se podría pensar en simplificar el cálculo sustituyendo directamente en la ecuación:

$$N(x,y) = \frac{(255-1)}{2047} * I(x,y)$$

Esto es acertado para el propósito de nuestro proyecto, sin embargo, debido a que la mayoría de valores se agrupan en torno a la media, no es efectivo de cara a obtener una imagen detallada de grises. El problema es que estableciendo como valor máximo 2047 y mínimo 0, todos los valores agrupados en torno a la media van a dar un valor normalizado de gris prácticamente idéntico, por lo que los objetos situados más o menos a la misma altura parecerán que están a la misma profundidad, cuando no es así. Lo conveniente para aplicaciones que requieren una buena reconstrucción en 3D del entorno es detectar el intervalo donde están la mayoría de puntos, de forma que los límites de este pasen a ser los de la ecuación. La diferencia entre una imagen de grises normalizada aplicando una normalización normal entre 0 y 2048 y otra con esta última técnica es muy clara:

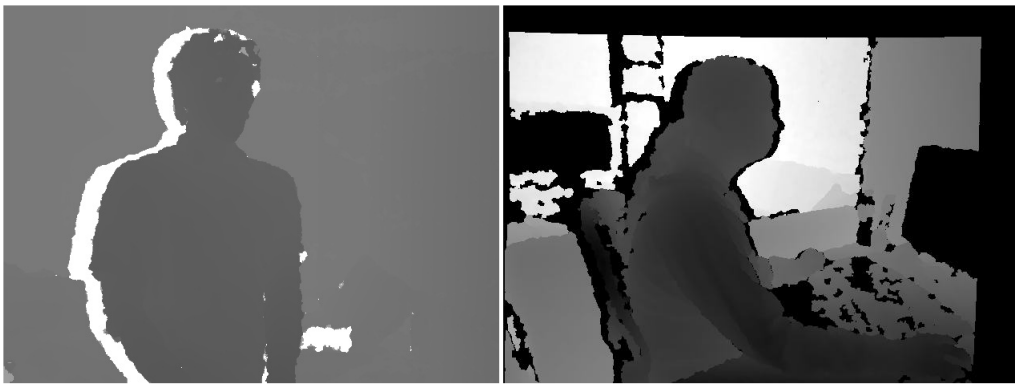


Figura 3.7: Normalizaciones diferentes: comparación

Si bien las imágenes no representan la misma escena, nos sirve para contrastar las tonalidades de gris de cada lado de la imagen. En el lado izquierdo, se puede apreciar una normalización de los datos en el intervalo [0, 2047] y en el lado derecho, una cuyo intervalo es mucho reducido y próximo a la media, lo que produce mejores resultados cuanto más cerca se esté del *Kinect* La normalización también actúa a modo de segmentación, pues valores altos para un intervalo pequeño, producirán colores cercanos al blanco.

Segmentación

La segmentación en el campo de la visión artificial es el proceso de dividir una imagen digital en varias partes (grupos de píxeles) u objetos. El objetivo es simplificar y/o cambiar la representación de una imagen en otra más significativa y más fácil de analizar. Se usa tanto para localizar objetos como para encontrar los límites de estos dentro de una imagen. Por consiguiente, la segmentación de la imagen es el proceso de asignación de una etiqueta a cada píxel de la imagen de forma que los píxeles que compartan la misma etiqueta también tendrán ciertas características visuales similares. Existen muchas técnicas de segmentación, las cuales listamos a continuación:

- Métodos de agrupamiento (Clustering).
- Métodos basados en el histograma.
- Detección de bordes.
- Métodos de crecimiento de regiones.
- Métodos del conjunto de nivel.
- Métodos de particionado gráfico.
- Transformación divisoria (watershed).
- **Método del valor umbral.**
- Segmentación basada en modelos.
- Segmentación multiescala.
- Segmentación semiautomática.
- Redes neuronales.

De todas ellas, dada la complejidad de algunas, sólo explicaremos la segmentación por umbral, técnica idónea para un problema sencillo de segmentación de imágenes en escala de grises, donde lo que interesa es filtrar por profunidad o lo que es lo mismo, aplicar un umbral para un determinado valor de gris. Para más información acerca de las otras técnicas, existe abundante bibliografía al respecto y *OpenCV* implementa muchas de ellas.

Un ejemplo de umbralización binaria (en blanco, todo aquello que está por encima del umbral o límite y en negro, todo lo demás) aparece en la figura 3.6. **Se utilizará esta técnica de umbralización (binaria) para el procesamiento de las imágenes de grises.**

La segmentación será una pieza indispensable en el proceso de análisis de la imagen, pues descartará todos aquellos objetos que no son de interés. lo más importante en la segmentación

es conocer la distancia en metros y traducirla a su nivel de gris correspondiente para así poder indicar al usuario la distancia aproximada y que este se coloque donde le plazca. Muchas veces la segmentación produce pequeños corpúsculos o grupos de píxeles reducidos aislados del cuerpo principal, que pueden confundir a la hora de analizar los objetos presentes en la imagen. Es por esto que suelen utilizar las técnicas que a continuación detallamos (siguientes subsecciones).

Erosión (erode)

La erosión consiste en examinar cada píxel y cambiarlo de *"activado"*(1) a *"apagado"*(0) si alguno de sus vecinos está a 0. Normalmente se utilizan como vecinos los ocho que rodean al píxel examinado, aunque para algunas aplicaciones se pueden utilizar conectividades de 4 vecinos (los 2 verticales y los 2 horizontales) e incluso conectividades de 2 vecinos (los verticales o los horizontales).

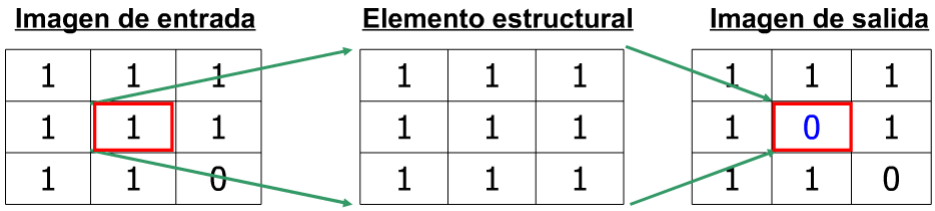


Figura 3.8: Proceso de erosión en imágenes binarias.

Por ejemplo, si partimos de la siguiente imagen en blanco y negro:

Figura 3.9: Imagen original binarizada

La imagen erosianada será:

Figura 3.10: Imagen erosionada

Dilatación (dilate)

La dilatación es el proceso inverso de la erosión, esto es, consiste en expandir los objetos de la imagen. La dilatación generalmente aumenta el tamaño de los objetos, rellenando agujeros y conectando las áreas que están separadas por espacios más pequeños que el tamaño del elemento estructural.

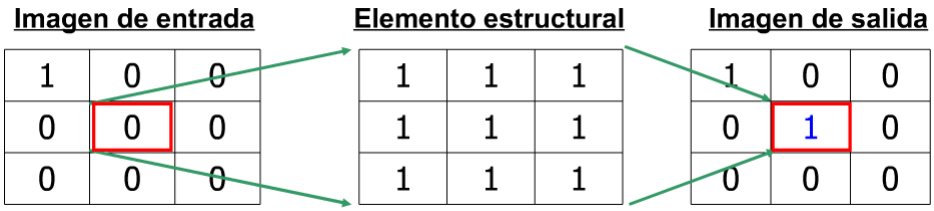


Figura 3.11: Proceso de dilatación en imágenes binarias.

Basándonos en la misma imagen de origen que para la erosión:

Figura 3.12: Imagen original binarizada

La imagen erosianada será:

Figura 3.13: Imagen dilatada o expandida

Apertura (opening)

Al resultado de una erosión más una dilatación se le denomina apertura (“opening”). El nombre proviene de la tendencia de esta secuencia de operaciones a separar (abrir) puentes de unión entre objetos próximos o a abrir cavidades próximas al borde.

Figura 3.14: Apertura

En la imagen se ven un ejemplo de la combinación de ambas técnicas, se lee de izquierda a derecha y de arriba a abajo.

Cierre (Closing)

La operación opuesta (dilatación más erosión) es denominada cierre ("closing") y puede usarse para conectar objetos muy próximos o para rellenar pequeños huecos.

Figura 3.15: Cierre

En la imagen se ven un ejemplo de la combinación de ambas técnicas, se lee de izquierda a derecha y de arriba a abajo.

Cálculo de la distancia y posición del usuario

Partiendo del siguiente principio: **"no puede haber ningún objeto delante del usuario, es decir, el usuario con los brazos extendidos a lo largo del cuerpo debe ser el objeto más cercano a la cámara"**. El cálculo de la posición del usuario se basará pues, en el valor de gris más obscuro obtenido de la imagen normalizada.

El usuario, tiene que poder conocer la distancia exacta a la que se encuentra de la fuente y es necesario presentársela en un formato que él pueda entender. Como podremos intuir, mostrar un "está usted a un nivel de gris 35" no quiere decir nada para este, pero sin embargo, mostrar dicha información traducida a metros sí que puede serle de gran utilidad, sobre todo para colocarse delante del objetivo, pues en diversas pruebas realizads con el *Kinect* se estableció como distancia idónea del cuerpo respecto a la fuente los 90cm. De este modo, al extender las manos hacia delante, se puede ver claramente su contorno y ocupan la mayor parte de la imagen, lo que facilita el análisis posterior.

La relación entre los valores "en bruto" obtenidos y la distancia en metros viene dada por la siguiente fórmula, obtenida de la página oficial del proyecto *OpenKinect*.

$$Distancia\ en\ metros\ (vbprof) = \frac{1}{(vbprof * -0{,}0030711016) + 3{,}3309495161}$$

Donde *vbprof* quiere decir "valor en bruto" o en inglés, "raw data" y puede tomar un valor entre 0 y 2047.

Dicho proyecto enlaza con otra fórmula, más precisa y obtenida igualmente de forma empírica, resultado de la investigación de Stéphane Magnenat [2].

$$Distancia\ en\ metros\ = 0{,}1236 * \tan\left(\frac{vbprof}{2842{,}5} + 1{,}1863\right)$$

Esta aproximación tiene un margen de error de unos 10cm a 4m de distancia, y menos de 2 cm de radio a 2,5 m.

La validez de estos cálculos se comprobó de una forma rudimentaria, pero con la precisión suficiente, mediante una cinta métrica, ya que obtener medidas precisas (calibrar) que muestren la relación entre los valores en bruto y la distancia en metros, requeriría de un profundo estudio, con un aumento considerable del tiempo dedicado. Dado que estos resultados tienen un margen de error pequeño (menos de 2 2 cm a 2.5 metros de distancia) se toman por válidas.

3.5.4. Análisis de contornos

El análisis de los contornos obtenidos luego de las fases de normalización y segmentación se considera la parte más importante del análisis visual, ya que las otras fases no son sino una "preparación" para poder trabajar cómodamente (eliminamos y trasnformamos las imágenes para poder extraer mejor sus características), pues de esta etapa saldrá la información que el vehículo espera recibir: la velocidad y el giro.

Se considera un contorno a una sucesión de puntos cerrada que, normalmente, se corresponde con el borde de un objeto concreto. La librería *OpenCV* nos permite calcular los contornos de una figura y es ya tarea del programador filtrar aquellos que no se ajusten a unas determinadas características. La técnica para extraer el contorno de las dos manos es sencilla, pero efectiva y consiste en:

1. Cálculo de contornos: se almacenarán en una lista.

[2] Publicación de Stéphane Magnenat: `http://groups.google.com/group/openkinect/browse_thread/thread/31351846fd33c78/e98a94ac605b9f21?lnk=gst&q=stephane&pli=1`

2. Descartar todos aquellos contornos con un área inferior a una dada: recordemos que un contorno es una sucesión de puntos cerrada, es decir, un polígono, normalmente irregular.

3. De los contornos resultantes, consideramos como la mano izquierda el que esté más a la izquierda y a la inversa para la derecha.

Las fases de erosión y dilatación sobre la imagen binarizada, dejan dos contornos perfectamente definidos (eliminan zonas aisladas y pequeños grupos negros), por lo que el filtro de áreas dejará siempre dos contornos y la selección de la mano izquierda y derecha resulta trivial (Figura 3.16).

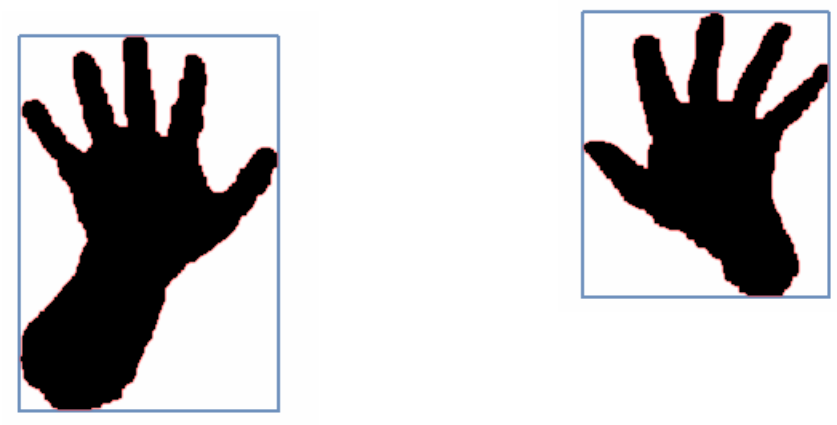


Figura 3.16: Extracción de contornos

Los contornos y la caja que los contiene (también llamada por su término en inglés, "bounding box") son la base para la explicación de cómo se extraerán los dedos de la mano y cómo se cálculará qué ángulo debe tomar el vehículo.

Extracción de los dedos de la mano

La extracción de los dedos de la mano se basa en el concepto de polígono convexo, que puede definirse como el polígono recubridor del objeto, resultante de simplificar todos los puntos del contorno a un número determinado de lados. Entonces, si aproximamos a

un número bajo de lados, podemos asegurar que cada vértice se corresponderá con un dedo, teniendo en cuenta que deben eliminarse los que estén por debajo de un determinado umbral. En la figura 3.17 observamos un polígono de X lados.

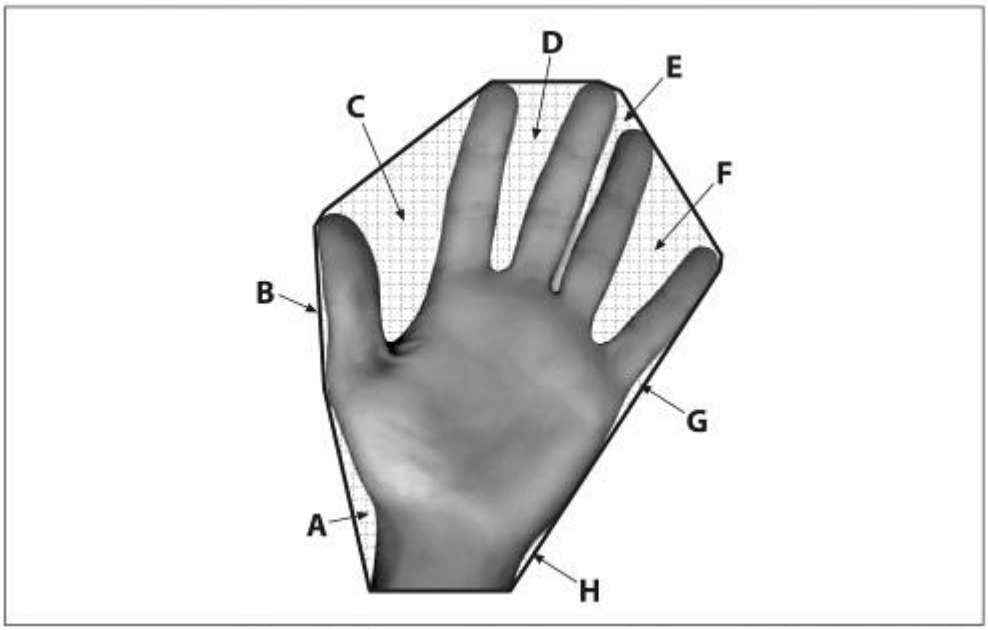


Figura 3.17: Polígono convexo

Como se puede ver en la figura 3.17, el polígono tiene nueve lados y, por tanto, nueve vértices por lo que no todos ellos se corresponderán con un dedo. Se hace necesario pues aplicar un filtro que discrimine aquellos vértices que no son dedos de la mano.

Existen varias técnicas para seleccionar los vértices que interesa guardar, una de ellas es el análisis mediante una red neuronal, método efectivo, pero complicado de poner en marcha, pues requiere de un conjunto de pruebas muy grande y entrenar la red. Otras no relacionadas con la inteligencia artificial se basan en el análisis puro de propiedades como la distancia a un punto o la pertenencia a un grupo concreto de puntos.

En la figura 3.17 los vértices son el resultado de calcular los defectos de convexidad, es decir, las zonas abiertas señaladas y nombradas con letras de la A a la H. La unión de los vértices que delimitan dicha región, se corresponderá con los dedos de la mano. Para descartar aquellos puntos que no interesan, ya que aunque cubren un **defecto de convexidad** que no tienen porqué ser un dedo, se aplica el filtro anteriormente mencionado, el cual se basa en el cálculo de dos propiedades para cada punto del polígono convexo: **la distancia con respecto del centro de la mano** (que es el centro del rectángulo contenedor del contorno, ya que sólo se ven las manos en la segmentación) y **un umbral de altura**, que consiste en descartar todos aquellos puntos que estén por debajo de él. En la ilustración 3.18 se pueden

apreciar, círculos en color gris claro y sólido, que son los dedos de la mano y círculos huecos de color negro, que se corresponden con los puntos interiores de la mano. El centro de la mano está marcado con una cruz azul y verde. El dedo de la izquierda no está marcado porque al estar recogido, no cuenta para el sistema como un dedo porque no está a la distancia mínima del centro (de esta forma evitamos que los nudillos se consideren como un dedo).

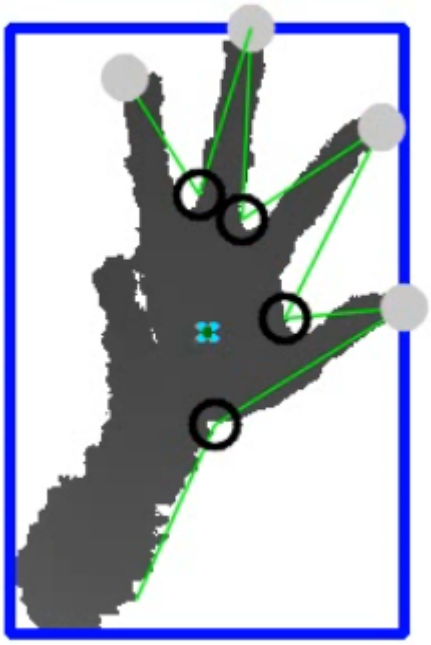


Figura 3.18: Extracción de los dedos

Los puntos interiores de la mano podían calcularse de dos formas diferentes, una utilizando la técnica del polígono cóncavo, que no es sino la inversa del convexo, por lo que se obtendrían tanto los dedos como sus uniones en una misma pasada (la técnica es más compleja) y otra, de nuevo, mediante el cálculo de la distancia de los puntos más cercanos al centro.

Así pues, todo está basado en un cálculo de distancias con respecto al centro de la mano:

- Los dedos de la mano serán los puntos más alejados (descartando los que estén por debajo del centro del contorno.)
- Los puntos de anclaje de los dedos o lo que es lo mismo, la palma de la mano, serán los puntos más cercanos al centro del contorno.

Control del giro

El ángulo de giro determinará la mayor o menor medida en que el vehículo girará conforme movemos el brazo. Se considerará un sistema formado por un volante, fijo en la imagen y la mano derecha del usuario, que girará trazando arcos sobre este (figura 3.19).

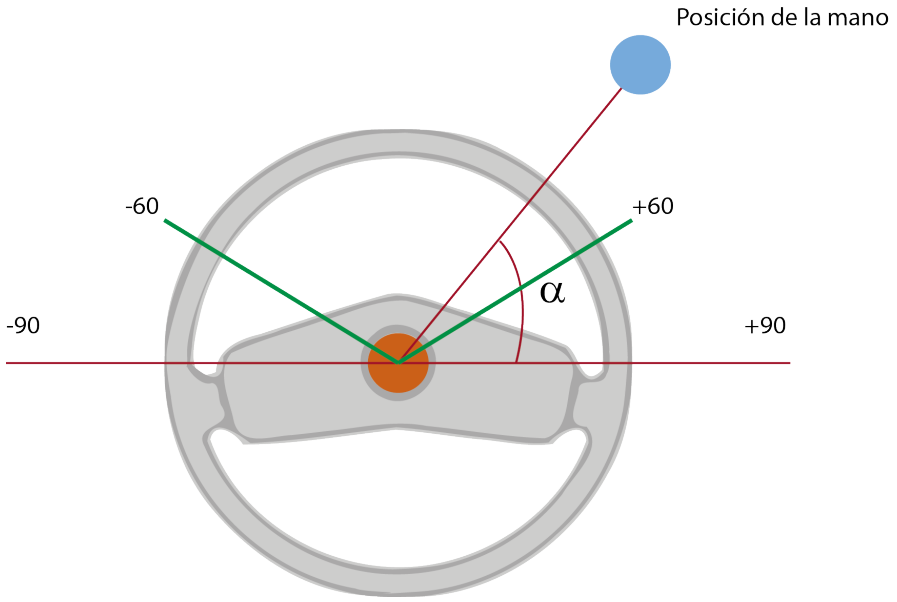


Figura 3.19: Sistema de control del giro

En la figura 3.19, el punto azul indica la posición de la mano con respecto al volante y el naranja el centro del mismo. El ángulo que formen el vector que une el punto azul y naranja con el eje horizontal OX será el que se envíe al vehículo. Esta forma tan sencilla de resolver el problema del giro es posible ya que el controlador de *Lego Mindstorms* tiene memoria de los ángulos girados para cada motor y es capaz de situarse en un ángulo determinado con respecto al punto de partida, luego si en un instante de tiempo el ángulo mide 45° y en el instante siguiente se envía 50°, el vehículo no girará 50° grados más desde dónde estábamos, sino que calcula el punto de partida y determina el giro que debe realizar para alcanzar esa posición, es decir, 5° grados más.

Existen limitaciones físicas físicas a este modelo, pues no es posible en la vida real realizar un giro de 90°, ya que la mecánica impone límites. En este caso se ha establecido como intervalo de giro [-60, 60], rango dónde se puede desplazar la mano libremente sin provocarle daños al vehículo (forzar el mecanismo, engranajes que no engranan, etc.). En la figura 3.19 se represente mediante dos segmentos verdes unidos en el centro del volante.

3.5.5. Técnicas de seguimiento de objetos

Seguir un objeto consiste en primer lugar, en detectarlo y en segundo lugar, en predecir cuál será su próxima posición cuando no sepamos o no podamos reconocerlo en la imagen. Existen numerosas técnicas de seguimiento o “tracking” de objetos, pero pueden englobarse en dos categorías principales, las basadas en modelos y las basadas en características. No es objeto de este proyecto estudiar en profundidad estas técnicas, sino dar una noción general y elegir aquella que sea más simple en coste computacional, valorando muy positivamente que

tenga ya un respaldo de código (por ejemplo, que venga implementado con alguna librería).

- **Basado en modelos** Este tipo de técnicas se basan en el conocimiento del modelo del objeto. Puede ser un modelo CAD o una proyección del objeto. Esta técnica es más robusta que la basada en características.

- **Basado en características** Se basa en las características extraídas de la imagen y no en la búsqueda de un modelo conocido. La idea fundamental en la que se basa el "tracking" de características es *¿por qué hacer tracking del objeto entero cuando se puede obtener el mismo resultado haciendo tracking solo de las características?*. Este planteamiento suele ser computacionalmente más eficiente que el basado en modelo, pero es menos robusto.

En el siguiente esquema, podemos ver los diferentes tipos de técnicas de seguimiento que existen:

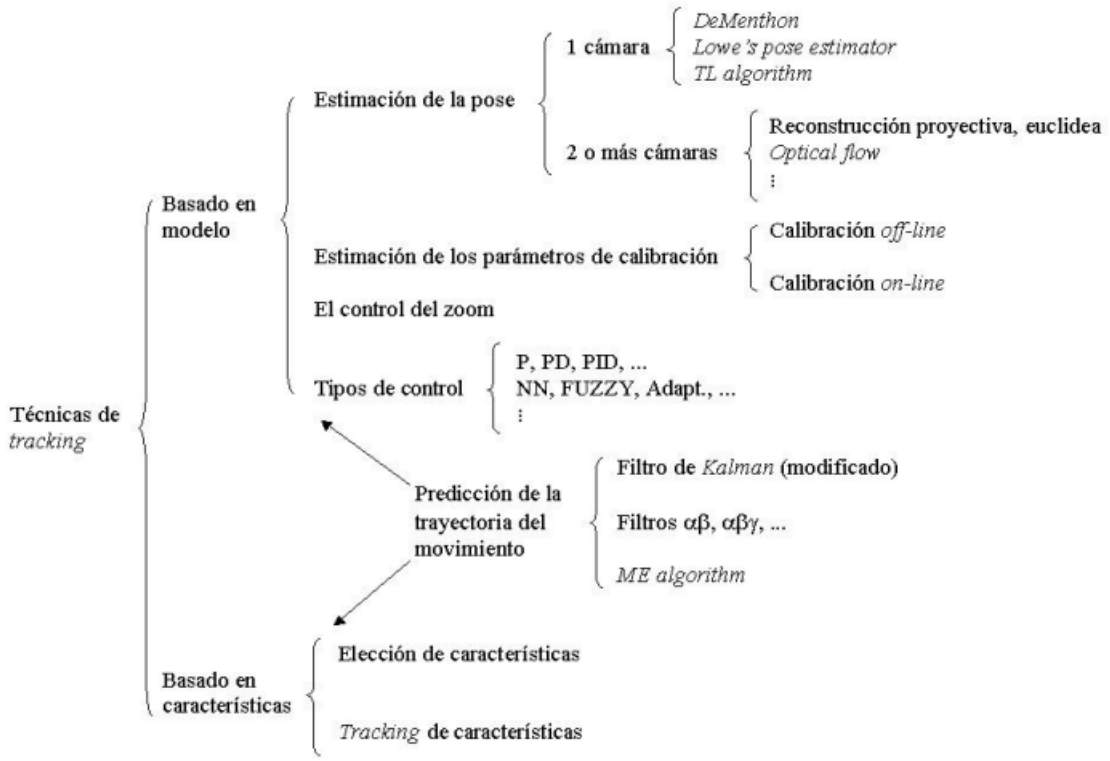


Figura 3.20: Técnicas de tracking

En este proyecto se realizará un tracking de la mano izquierda basado en características, pues interesa reconocer objetos que sean manos y se pueden utilizar características como el área del contorno, el centro del mismo, la profundidad, etc. En cuanto a la predicción de la

trayectoria, se utilizará un filtro de Kalman [3], ya que es fácil de utilizar y viene implementado con la librería OpenCV. Además, el rendimiento y el coste computacional es muy bueno.

Filtro de Kalman

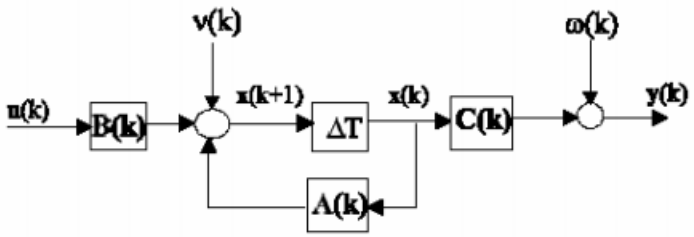
El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960 que sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal, incluso cuando el sistema está sometido a ruido blanco aditivo. El objetivo del filtro de Kalman es estimar los estados de una manera óptima, de manera que se minimice el índice del error cuadrático medio. Aplicándolo al proyecto, se puede decir que sirve para conocer la posición de un objeto cuando no es visible (estado oculto) en la imagen.

En la construcción de un sistema determinístico trabajaríamos de la siguiente forma: En primer lugar, construiríamos un modelo del sistema a partir de las leyes físicas que definen las dinámicas. En segundo lugar, estudiaríamos su estructura y su respuesta. Por último, y si fuera necesario, se diseñarían compensadores para alterar las características del sistema o bien reguladores. Para ello tendríamos además un amplio conjunto de experiencias que han sido aplicadas con anterioridad sobre estos sistemas. En la realidad ocurre que los sistemas determinísticos son aproximaciones de lo que realmente son, principalmente por tres motivos:

- No existe un modelo matemático perfecto de un sistema real.
- Existen perturbaciones que no se pueden modelar de una forma determinística.
- Los sensores no son perfectos.

Por lo tanto, surge la necesidad de plantearse cómo desarrollar modelos de sistemas que tengan en cuenta las incertidumbres, que estimen de forma óptima los datos que interesan de un sistema mal modelado y con datos alterados por el ruido y que controlen de forma óptima sistemas estocásticos.

Para poder explicar el filtro de Kalman correctamente, conviene utilizar la forma matemática. Por tanto, dado que un filtro de Kalman es al fin y al cabo un sistema lineal cualquiera, entonces se puede representar así:



[3]The Kalman Filter: `http://www.cs.unc.edu/~welch/kalman/`

Figura 3.21: Un sistema lineal genérico con retroalimentación.

De manera que su evolución es expresada en espacio de estados por:

$$x(k+1) = Ax(k) + Bu(k) + v(k)$$

$$y(k) = Cx(k) + w(k)$$

Siendo:

- x(k): estado.
- y(k): salida u observación del sistema.
- w(k): proceso estocástico asociado a la medida.
- v(k): proceso estocástico asociado al sistema.
- A, B y C: matrices determinísticas que definen la dinámica del sistema.

Y asumiendo como condiciones las siguientes:

- E[x(0)] = x_0
- E[w(k)] = 0 $\forall$ k
- E[v(k)] = 0 $\forall$ k
- E[x(0),w(k)] = 0 $\forall$ k
- E[x(0),v(k)] = 0 $\forall$ k
- E[v(k),w(j)] = 0 $\forall$ k
- E[w(k),w(k)] = R(k)
- E[w(k),w(k)] = 0 $\forall$ k $\neq$ j
- E[v(k),v(k)] = Q(k)
- E[w(k),w(j)] = 0 $\forall$ k $\neq$ j
- E[x(0),x(k)] = P_0 $\forall$ k

Las matrices de covarianza Q(k) y R(k) son diagonales y por tanto simétricas. En el sistema real se puede obsrvar el valor de y(k) de manera directa con los sensores adecuados. Esta medida incorporará una serie de incertidumbres asociadas: la incertidumbre del sensor y la del sistema. Por otro lado solo podremos acceder al valor y(k). En caso de necesitar la evolución completa del estado x(k) y/o de precisar el valor de la observación ajena a las variaciones provocadas a la incertidumbre, tendremos que estimar de alguna manera indirecta sus valores.

El filtro de Kalman propone un método para obtener un estimador óptimo del estado. Si suponemos que x̂(k) es la estimación en el instante k del estado. El filtro de Kalman buscará obtener ese valor de estimación de manera que se minimice el error cuadrático medio. Definiendo el error como la diferencia entre el valor real del estado y la estimación:

$$e(k) = x(k) - \hat{x}(k).$$

El objetivo por tanto será minimizar:

$$P(n) = E[e(n).e^T(n)]$$

A la matriz P(n) se la conoce como matriz de covarianza del error.

El esquema general del funcionamiento del filtro es el siguiente y básicamente se basa en la predicción de un estado, la correción del mismo y su estimación posterior.

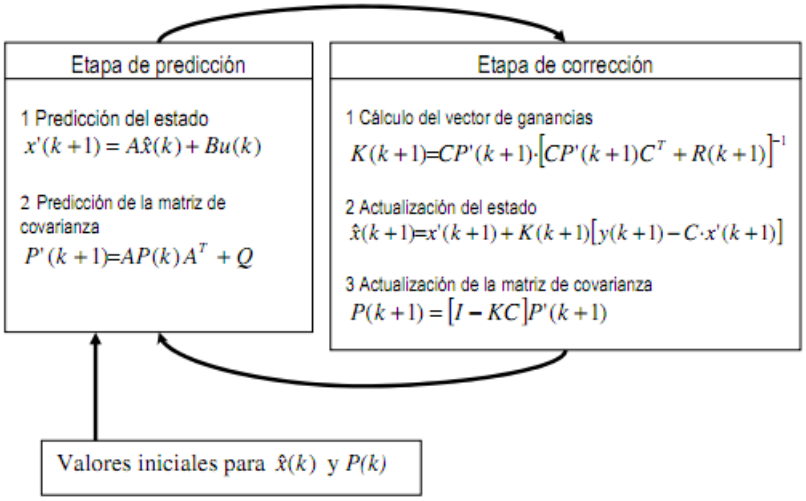


Figura 3.22: Etapas del filtro de Kalman.

Se omite la explicación de cada una de las etapas, ya que es mucho contenido y requiere un análisis en profundidad. No obstante, se puede consultar en el siguiente en la web una explicación concisa y clara del filtro completo[4].

Para finalizar este apartado, expondremos una explicación matemática del modelo dinámico utilizado. En concreto, se ha empleado un sistema formado por un cuerpo moviéndose en el espacio y sometido a tres componentes: posición, velocidad y aceleración.

Así pues, utilizaremos como modelo las ecuaciones del movimiento, donde "v_i" es la velocidad inicial del cuerpo y "s_i" la posición inicial y el estado actual dado un instante de tiempo lo de terminan la velocidad (v), la posición (s), la aceleración (a) y el intervalo de tiempo entre el estado inicial y el actual (Δt):

$$v = v_i + a\Delta t$$

[4]Filtro de Kalman: `http://www.inelmatic.com/web/files/downloads/filtrodekalman.pdf`

$$s = s_i + v_i \Delta t + \frac{1}{2} a (\Delta t)^2$$

$$s = s_i + \frac{1}{2}(v + v_i)\Delta t$$

$$v^2 = v_i^2 + 2a(s - s_i)$$

Sólo quedaría ya definir los parámetros de incialización del filtro, teniendo en cuenta lo anteriormente comentado:

- **Estado del sistema**: posición + velocidad + aceleración, es decir, el modelo dinámico de movimiento que sigue la mano.
- **Dimensión del vector de estado**: 6 $\rightarrow$ se consideran dos subcomponentes por cada componente del estado (s_x, s_y, v_x, v_y, a_x, a_y).
- **Vector de medición**: Lo que estamos midiendo, es decir, un punto en el espacio bidimensional, ya que al fin y al cabo, la forma de representar un objeto en el espacio es mediante un punto.
- **Dimensión del vector de medición**: 2 (posición x y posición y del punto medido)
- **Matriz de transición de estados**:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0{,}5 & 0 \\ 0 & 1 & 0 & 1 & 0{,}5 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0{,}5 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

- **Matriz de medición asociada al vector de medición**:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0{,}5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0{,}5 \end{bmatrix} \tag{3.2}$$

Así pues, el filtro quedaría adaptado a nuestro problema y ya en el capítulo de resultados y pruebas expondremos los resultados de forma gráfica, que es la mejor forma de verlo.

3.5.6. OpenCV

Todas las técnicas descritas anteriormente deberán programarse posteriormente, pero afortunamdamente existen bibliotecas de visión con mucha tradición y peso en el mercado, avaladas por su buen funcionamiento y optimización de recursos como son *CImg* y *OpenCV*. *OpenCv* [5] es, probablemente, la má popular. Es multiplataforma, y está escrita en C, C++,

[5] Más información: `http://sourceforge.net/projects/opencvlibrary/`

Phyton y Java entre otros. Incluye una gran cantidad de ejemplos y recursos online, así como libros y revistas que facilitan la utilización de manera sencilla y eficiente. Contiene todo lo que ofrecen las bibliotecas *CImg* y además las completa con algoritmos de reconocimiento, seguimiento, extracción de imágenes de dispositivos externos, calibración, así como detección de movimiento y búsqueda de patrones. Esta opción ha sido la elegida dadas las circunstancias y requerimientos de este trabajo.

3.5.7. Sombra proyectada sobre objetos

Ya por último, falta comentar un dato importante acerca de las imágenes obtenidas con el sensor láser. Cuando hablamos de **"sombra"**, nos referimos a aquellas zonas que son "invisibles" al *Kinect* por quedar detrás del objeto, normalmente. En el siguiente esquema (fig. 3.23), se observa este fenómeno, que debe tenerse en cuenta a la hora de analizar la imagen, pues debe eliminarse.

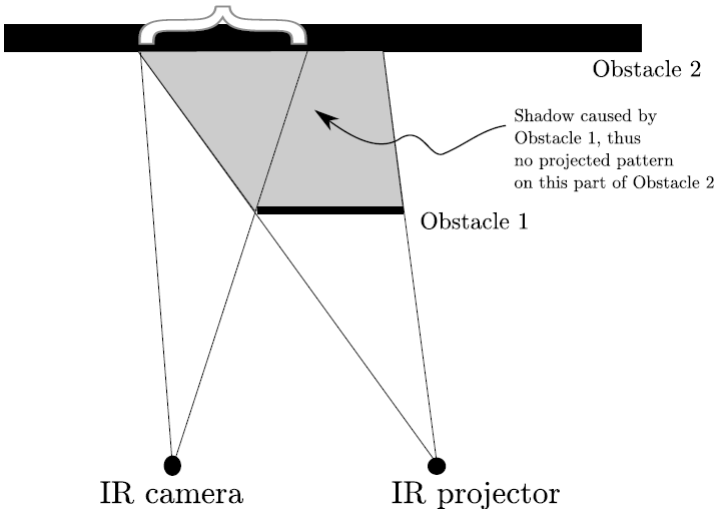


Figura 3.23: Sombra proyectada sobre objetos

El proceso de eliminación es sencillo, pues las zonas de sombra tienen un valor 0 en la matriz de profundidad, así que una simple segmentación modificada para evitar que tome como valor máximo el 0 bastará.

3.6. Análisis final: recopilación de estrategias

En las secciones anteriores se ha hablado de las diferentes estrategias a seguir para cada uno de los tres grandes bloques en los que se divide el proyecto, que son control gestual, comunicación y análisis de imagen (apartados 3.3, 3.4 y 3.5). En un intento por sintetizar toda la información para crear una idea general de funcionamiento. El proceso global consta

de dos fases: inicialización y control del vehículo.

Para finalizar, se incluirán los casos de uso con los que se puede topar el usuario al utilizar la aplicación.

3.6.1. Fase 1: Inicialización

En esta primera fase, el programa cliente intentará establecer una conexión *Bluetooth* con el dispositivo *NXT*, el cual estará esperando conexiones nuevas. Si el coche no está encendido y su respectivo programa cargado, el usuario será capaz de reconectar a través de la interfaz, cuando el coche esté listo.

Después de este paso y antes de poder controlar el vehículo, es necesario que el usuario se presitúe delante del *Kinect* a una distancia recomendada de entre 85 y 100 cm. Inmediatamente después, el programa entra en el modo de control y el usuario ya puede empezar a controlar el vehículo. El siguiente esquema (fig. 3.24) ilustra esta situación, de una forma más genérica:

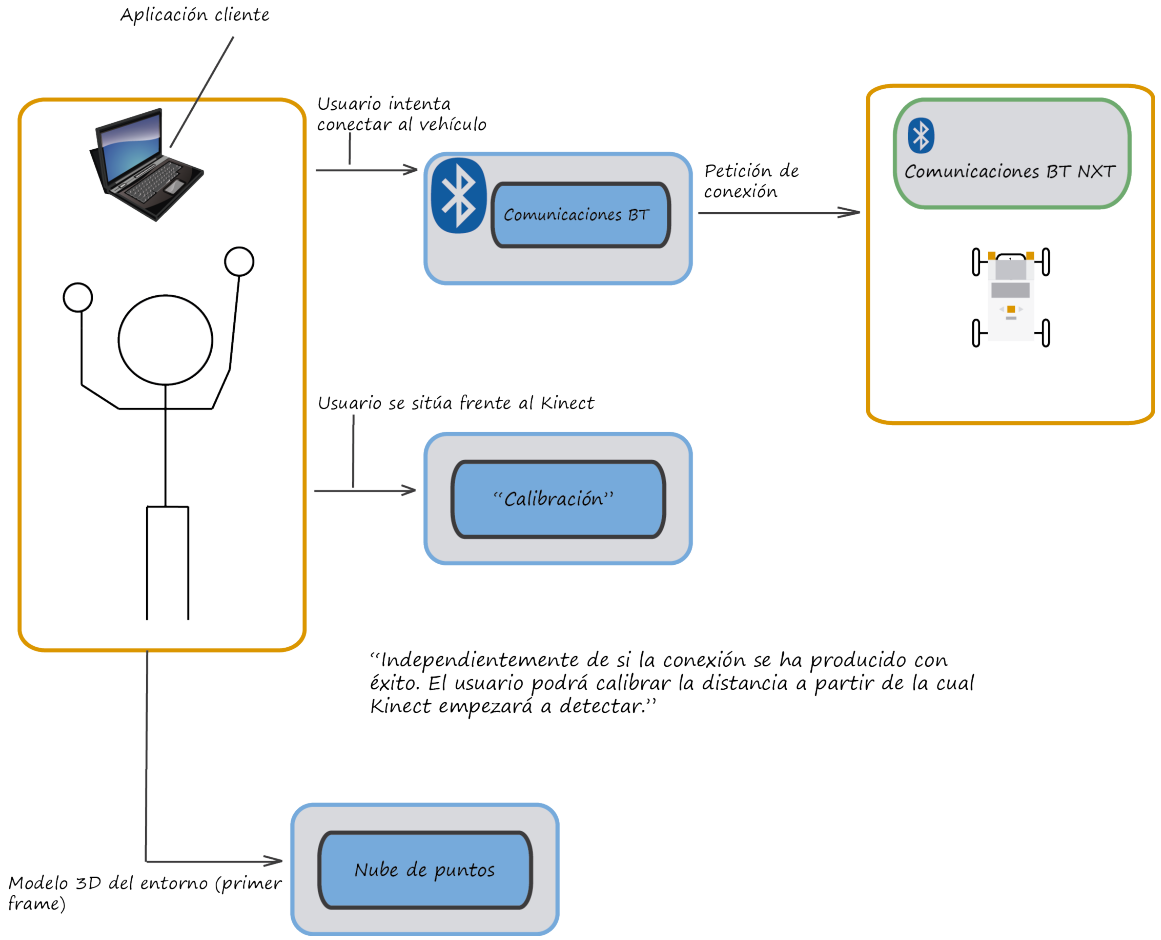


Figura 3.24: Vista general de funcionamiento (1)

3.6.2. Fase 2: Control del vehículo

El siguiente gráfico ilustra el proceso por el cual el usuario hace un gesto, *Kinect* lo captura y tras el análisis de la imagen se envía una órden al vehículo.

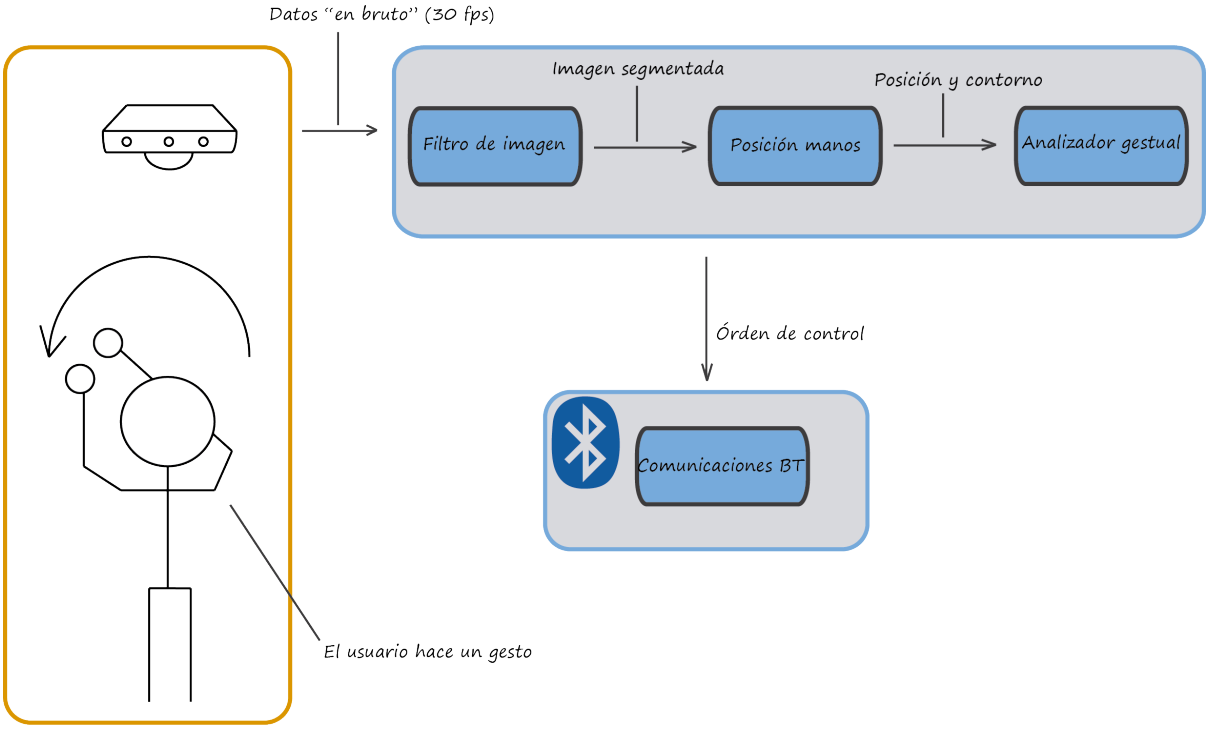


Figura 3.25: Vista general de funcionamiento (2)

La imagen capturada pasará por tres fases de análisis visual:

- **Filtrado:** La matriz de datos originales, es normalizada y segmentada.
- **Posición y seguimiento de las manos:** Se determina la posición y los contornos de las manos y se aplica el filtro de Kalman.
- **Análisis gestual:** A partir de los contornos y la posición de las manos, se determina el número de dedos de la mano izquierda (la marcha del vehículo) y el ángulo de giro del volante.

Finalmente, se transmitirán vía *Bluetooth* al vehículo los datos obtenidos (velocidad + giro).

3.6.3. Casos de uso

Todas estas técnicas descritas en este capítulo no tienen sentido si el usuario no es consciente de lo que está ocurriendo, es por eso que necesita de una interfaz gráfica de usuario, sobre la que irá viendo el resultado del análisis de las imágenes y el estado del bluetooth y de la calibración.

Los casos de uso serán limitados, pues toda la aplicación se basa en el control descrito en el capítulo de análisis. Así pues, en la figura 3.26, podemos ver las acciones que puede llevar a cabo el usuario y, posteriormente, una descripción textual de las mismas.

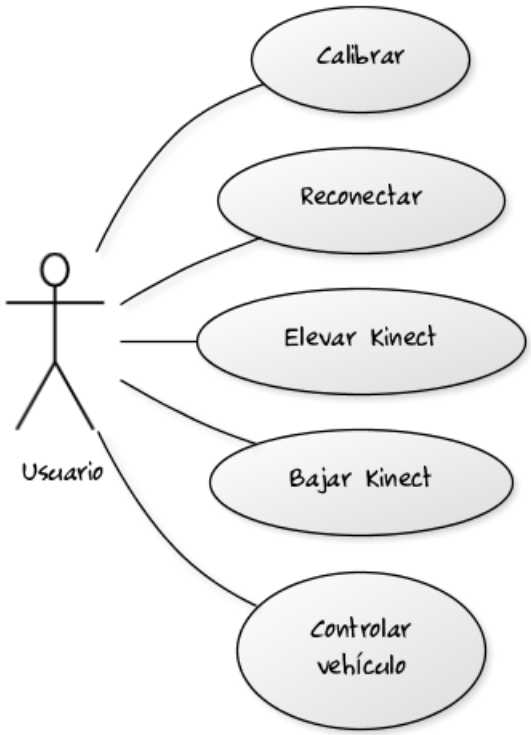


Figura 3.26: Casos de uso.

De forma textual, podemos describir el objetivo de cada caso de uso, los actores implicados (sólo uno en este caso), el escenario en que se desarrolla y las precondiciones y postcondiciones necesarias.

CU-01	Calibrar distancia
Objetivo	El usuario debe poder calibrar la profundidad de la segmentación
Actores	Usuario
Precondiciones	El programa está funcionando y no se ha realizado la calibración aún
Postcondiciones	Ninguna
Escenario principal	1. El usuario pulsa el botón de calibrar y tiene 10 segundos para posicionarse a la distancia deseada

Cuadro 3.1: CU: Calibrar distancia

CU-02	Reconectar bluetooth
Objetivo	Si la conexión se pierde o no puede realizarse, el usuario tiene que poder reintentarlo
Actores	Usuario
Precondiciones	El programa está funcionando y el intento de conexión anterior falló
Postcondiciones	El botón de reconexión desaparecerá.
Escenario principal	1. El usuario pulsa el botón de reconectar y espera un mensaje de confirmación

Cuadro 3.2: CU: Reconectar bluetooth

CU-03	Elevar y bajar el ángulo de visión
Objetivo	Poder ajustar el ángulo vertical de visión del Kinect
Actores	Usuario
Precondiciones	El programa está funcionando
Postcondiciones	Ninguna
Escenario principal	1. El usuario pulsa sobre los botones “flecha” para subir o bajar el ángulo de visión en **un grado**

Cuadro 3.3: CU: Cambiar ángulo vertical de visión

CU-04	**Reconectar bluetooth**
Objetivo	El usuario debe poder controlar el vehículo con su propio cuerpo
Actores	Usuario
Precondiciones	El programa está funcionando, la calibración se ha realizado y se ha establecido una conexión bluetooth
Postcondiciones	Ninguna.
Escenario principal	1. El usuario pulsa el botón de reconectar y espera un mensaje de confirmación

Cuadro 3.4: CU: Controlar el vehículo

Capítulo 4

Diseño del sistema

El diseño del sistema es el paso siguiente al análisis y consiste en describir mediante diagramas u otras ténicas cómo se ha llevado a cabo la implementación de las estrategias decididas en la fase de análisis. Este capítulo engloba tanto a la parte *hardware* como a la *software*.

4.1. Diseño software

El diseño *software* se centra en la parte no física del proyecto, es decir, en la lógica que gobierna el *hardware* y como tal, tiene una importancia fundamental. En base a los casos de uso descritos en el capítulo anterior, se detallará las funcionalidades y la estructura de la interfaz de usuario. Para finalizar, se explicará la arquitectura interna del software, tanto para el vehículo como para la aplicación cliente (la que usa el usuario en su ordenador), acompañados de sus respectivos diagramas de clases.

4.1.1. Interfaz gráfica de usuario (GUI)

La interfaz gráfica de usuario permite a este comunicarse con el vehículo y configurar un par de opciones como son la calibración y el ángulo de inclinación del *Kinect*. Además, permite, en tiempo real, que el usuario pueda ver en la pantalla el resultado de aplicar las técnicas de visión comentadas en el capítulo anterior, esto es, podrá saber en todo momento cuántos dedos y qué ángulo está enviando la aplicación al vehículo.

En la siguiente figura (fig. B.39), se muestra la pantalla de inicio que el usuario verá nada más arrancar el programa. Se han recuadrado y numerado las regiones de interés pues se explicarán a continuación:

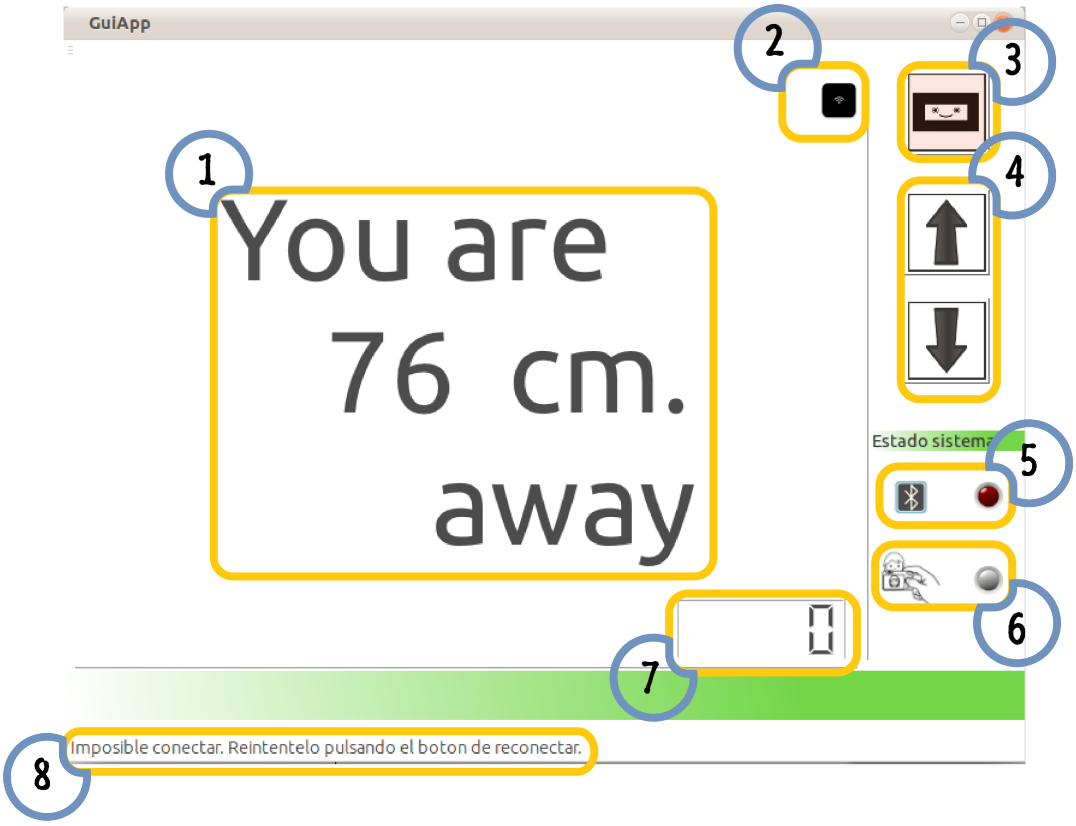


Figura 4.1: Vista principal de la interfaz

Cada región o zona recuadrada tiene una funcionalidad asignada, ya sea pasiva (mostrar información) o activa, desencadenar un evento o acción. Más detalladamente, clasificaremos los elementos presentes en activos y pasivos, basándonos en lo anteriormente dicho (nos referiremos a cada recuadro por "elemento + número", donde número es el número asociado que le acompaña en la imagen):

- **Elementos activos**
  - Elemento 2: Botón que permite al usuario reestablecer la conexión de *Bluetooth* con el vehículo. Aparece cuando fracasa un intento de conexión con el vehículo, por ejemplo, al inicio, pues intenta conectarse automáticamente.
  - Elemento 3: Botón de calibración. Tras pulsarlo, el usuario dispone de un tiempo para situarse a la distancia deseada del *Kinect*, ya que éste realizará la segmentación en ese nivel de profundidad.
  - Elemento 4: Botones de ajuste de la altura. Permiten ajustar el campo de visión vertical del aparato hacia arriba o hacia abajo (1 grado cada vez).
- **Elementos pasivos**

- Elemento 1: Pantalla principal, inicialmente muestra la distancia en metros del usuario al *Kinect* y posteriormente los dedos de la mano y el volante.
- Elemento 5: Indica el estado de la conexión con una “lámpara” de color rojo si está desconectado y verde si la conexión está establecida. Al inicio tendrá un color gris, indicador de que no se ha producido actividad aún.
- Elemento 6: Mismo tipo de indicador, pero para indicar si se ha calibrado la distancia con respecto a la fuente.
- Elemento 7: Indicador de los dedos de la mano. Muestra cuántos dedos está reconociendo el sistema para un instante dado.
- Elemento 8: Barra de información. Muestra información acerca de errores (por ejemplo, error al conectar).

Una vez realizada la calibración, la cual no implica necesariamente la conexión se entra en el modo de análisis de imagen y el usuario verá en patalla el resultado del procesado de sus manos, mediante un punto por cada dedo extendido y la mano izquierda enmarcada en un cuadro azul y la derecha en uno rojo (la imagen está al espejo). La siguiente figura (fig. 4.2) muestra la pantalla de ejecución, no habiendo más cambios que el de que la zona central, que pasa a mostrar imágenes en tiempo real y el de un pequeño recuadro en la esquina inferior izquierda que mostrará la imagen a color, a modo de referencia, pues no tiene utilidad alguna más que estética.

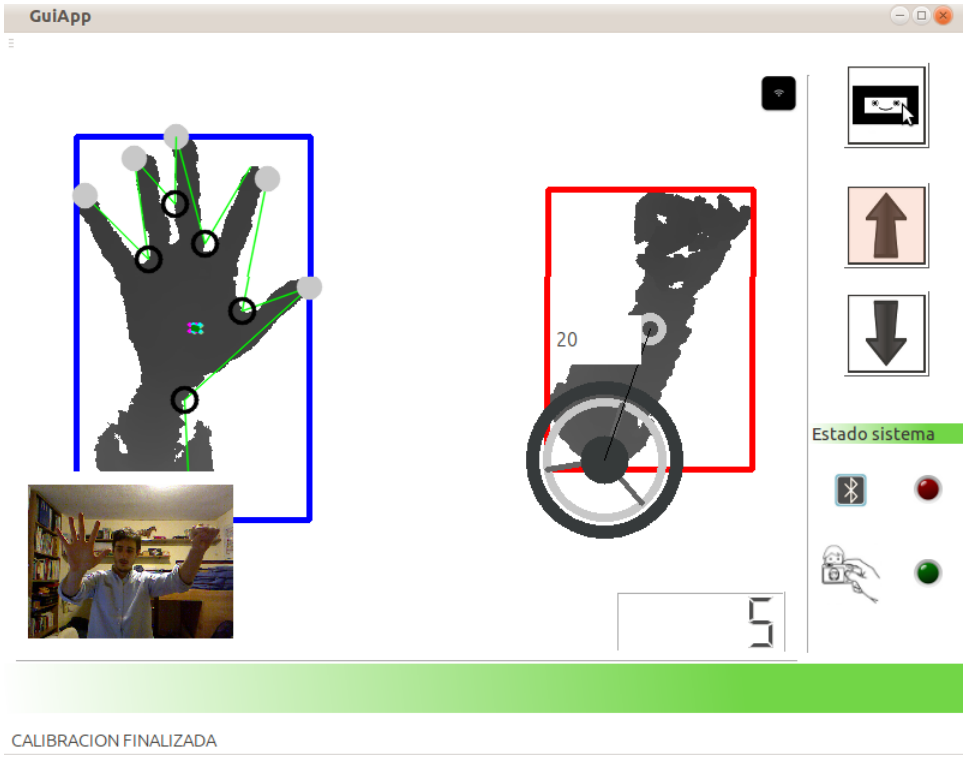


Figura 4.2: Vista principal de la interfaz (modo ejecución)

4.1.2. Aplicación cliente

La aplicación cliente consiste en el software que irá instalado en el ordenador o equipo que esté conectado al *Kinect*. Está desarrollada en C++ y, por simplicidad y claridad, se mostrará primero el diagrama completo y posteriormente se irá explicando por regiones (ampliadas, para una mejor visibilidad).

Diagrama de clases

En este apartado se muestra el diagrama de clases de la aplicación completo. La siguiente figura muestra las relaciones entre las clases, sus atributos y sus métodos. Se ha dividido y numerado el esquema en cuatro regiones (color rojo), para poder referenciarlas fácilmente.

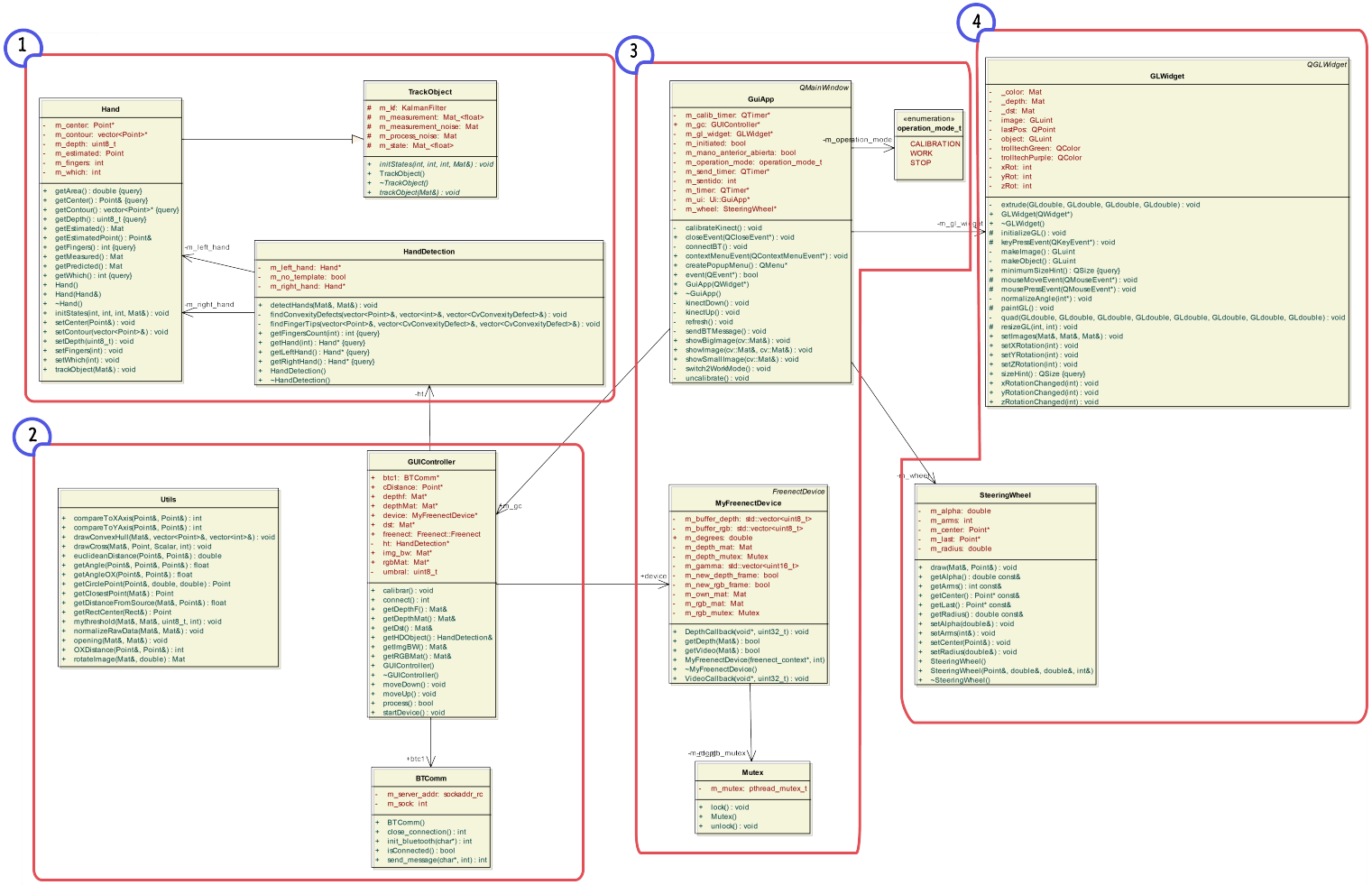


Figura 4.3: Diagrama de clases aplicación cliente (C++)

Diagrama de clases: primera parte

Se corresponde con el primer recuadro (marcado con un 1) del diagrama de la figura 4.3. En él se incluyen las clases relacionadas con el análisis de las manos, es decir, *Hand*, *HandDetection* y *TrackObject*.

TrackObject es una clase abstracta, de la cual hereda *Hand* (y cualquier otro objeto que se quiera "trackear") y que obliga a todas las clases hijas a implementar los métodos de inicialización (initStates) y seguimiento (trackObject). A continuación, se detallan más concisamente los aspectos de esta clase:

- **initStates**: La inicialización consiste en la construcción de una matriz de transición del modelo dinámico (T), un matriz de covarianza que representa el ruido del proceso (Q), la matriz de covarianza para la medición (R) y la matriz de transición de la medición. Se debe pasar por parámetro la dimensión del vector de estado y del vector medición.

- **trackObject**: Especifica la forma mediante la cual se realiza el seguimiento o "trackeo" del objeto. Para este caso, se produce una inicialización del filtro de Kalman en el constructor y este método simplemente actualiza la posición actual y corrige la predicción realizada por el filtro.

El objeto que se pretende seguir, es de tipo *Hand* y representa una mano (un contorno al fin y al cabo). Simplemente abstrae el contorno en un objeto que registra información acerca del centro de contorno, los puntos que lo conforman, la profundidad a la que se encuentra, qué tipo de mano es (izquierda o derecha) y el siguiente punto estimado.

Para finalizar, la clase *HandDetection*, es la que trabaja con objetos de las clases anteriormente y se ocupa del análisis de los contornos, determinando en cada instante de tiempo, qué contorno se corresponde con qué mano y extrayendo los posibles dedos para la mano izquierda. En el capítulo de análisis puede encontrarse una descripción detallada de cómo se ha resuelto este problema y el código está documentado y accesible junto a esta memoria.

La siguiente figura, muestra estas tres clases ampliadas, para una mejor lectura (fig. 4.4):

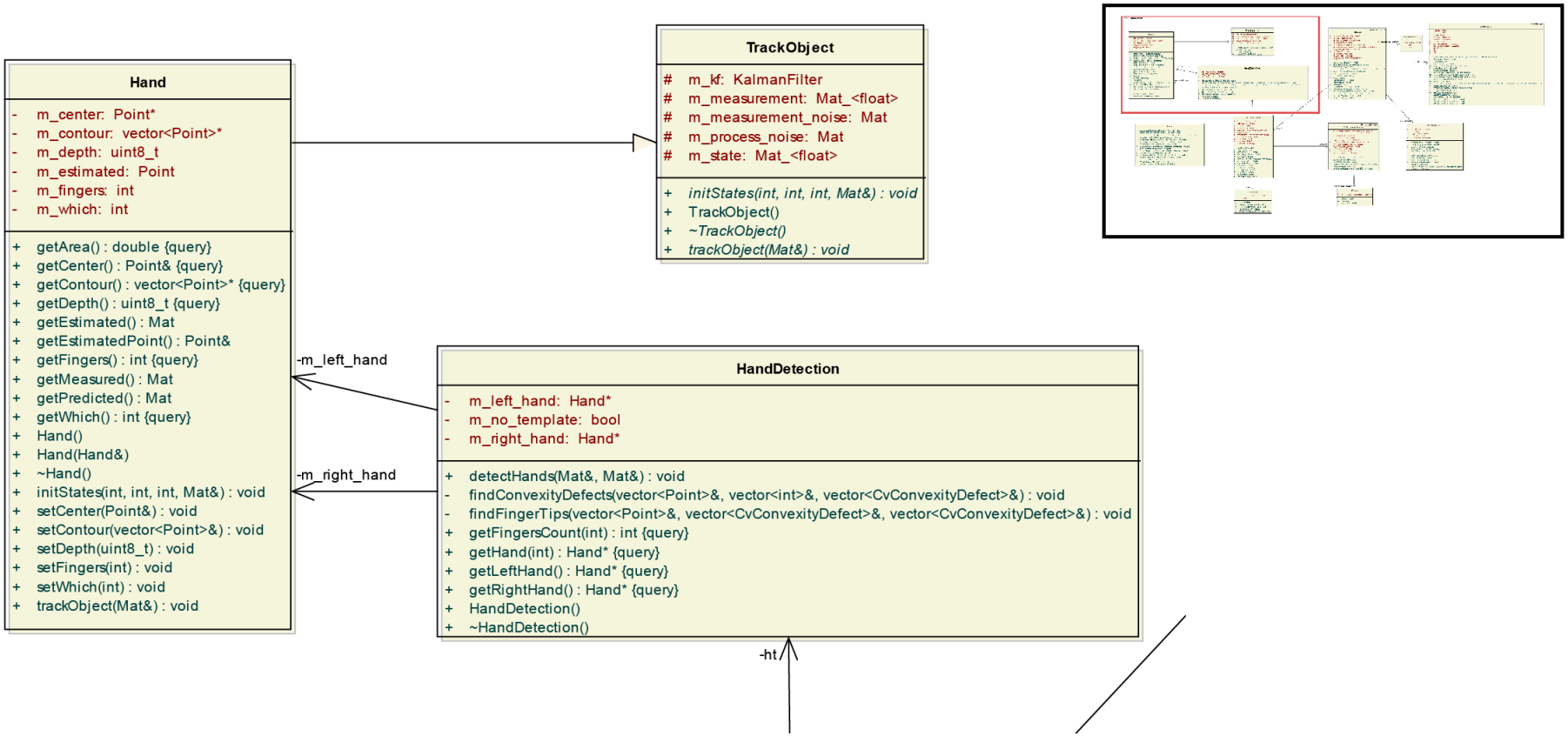


Figura 4.4: Diagrama de clases: parte 1

Diagrama de clases: segunda parte

Se corresponde con el segundo recuadro (marcado con un 2) del diagrama de la figura 4.3. En él se incluyen las clases *Utils*, *GuiController* y *BTComm*.

La clase *Utils*, proporciona funciones comúnmente utilizadas en el análisis de imagen principalmente; son todas estáticas y de acceso directo. La clase *BTComm* recoge toda la funcionalidad de las comunicaciones (lo que en el análisis denominábamos módulo de comunicaciones). Permite establecer una conexión y enviar mensajes vía *Bluetooth*. *BTComm* utiliza las biblioteca *bluetooth.h* , *rfcomm.h* y *socket.h* para implementar dicha funcionalidad.

Por último, *GuiController* es una de las clases más importantes puesto que hace las veces de controlador en un modelo MVC (modelo-vista-controlador). Aunque la arquitectura no se corresponde con este tipo de modelo, la clase *GuiController* se ocupa de aceptar las peticiones de la interfaz y de redirigirlas para que las trate el objeto correspondiente.

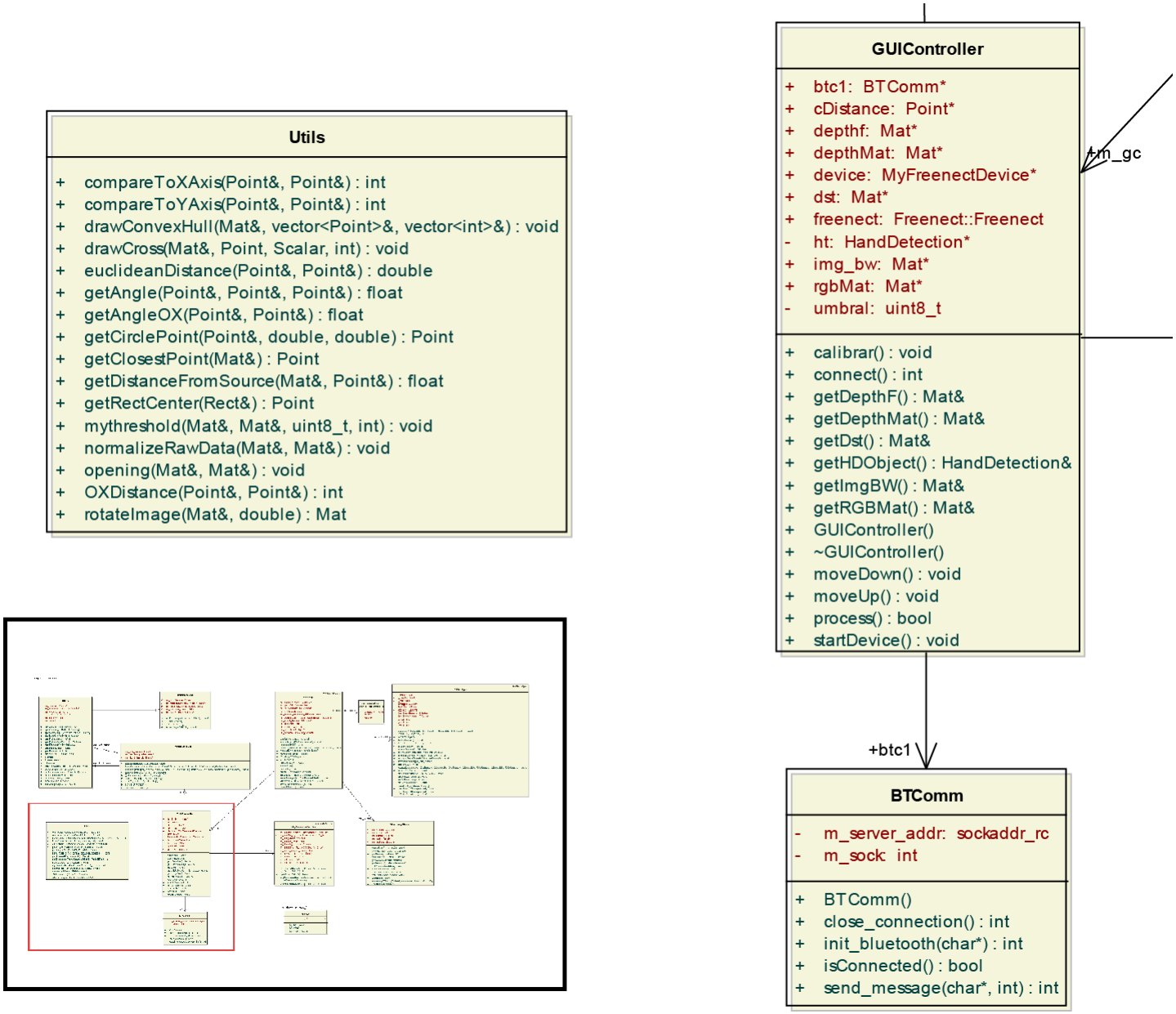


Figura 4.5: Diagrama de clases: parte 2

Diagrama de clases: tercera parte

Se corresponde con el tercer recuadro (marcado con un 3) del diagrama de la figura 4.3. En él se incluyen las clases *GuiApp*, *MyFreenectDevice* y *Mutex*.

La clase *GuiApp* representa toda la lógica de la aplicación. Gracias al uso de temporizadores controla el flujo y el orden en que se van sucediendo los eventos. Inicialmente mostrará en la vista principal (fig. B.39) los datos acerca de la distancia del usario a la fuente y el estado de las conexiones y de la calibración. Controlará todos los eventos asociados a los botones (calibrar distancia, subir o bajar Kinect) y delegará la funcionalidad sobre *GuiController*. Además, se ocupa de inicializar todo el aspecto visual de la interfaz gráfica. El enumerado *operation_mode_t* es utilizado por la aplicación para reflejar los tres estados posibles de la aplicación, que son:

- **CALIBRATION**: Especifica que la aplicación se encuentra en la fase previa a la que el usuario pulsa el botón de calibración.
- **WORK**: Indica que el usuario ha realizado la calibración correctamente y ya el análisis de imagen ya está disponible.
- **STOP**: Estado de no actividad, es decir, aplicación está parada.

La clase *MyFreenectDevice* (cabecera *mykinect.hpp*), es una abstracción del dispositivo *Kinect* y permite el acceso a todos los recursos del dispositivo, es decir, la cámara de profundidad, la cámara a color, los micrófonos y el audio. Esta clase utiliza el adaptador o "wrapper" para C++ de la biblioteca *libfreenect*, que como ya mencionamos en el estado del arte, es el controlador desarrollado por el proyecto *OpenKinect*. El adaptador está implementado en la cabecera *libfreenect.hpp* y encapsula toda la funcionalidad de esta biblioteca, para que pueda ser utilizada en clases como *MyFreenectDevice*.

La última clase de esta grupo, *Mutex* es una pequeña abstracción de la estructura (lenguaje C puro) *pthread_mutex_t m_mutex*, y representa un mutex (semáforo) cualquiera. Proporciona funcionalidad para bloquear el acceso a un recurso y viceversa (lock y unlock) y es necesaria para el acceso a los datos del Kinect en tiempo real.

La figura 4.6, refleja lo anteriormente explicado, ampliando la zona del diagrama correspondiente.

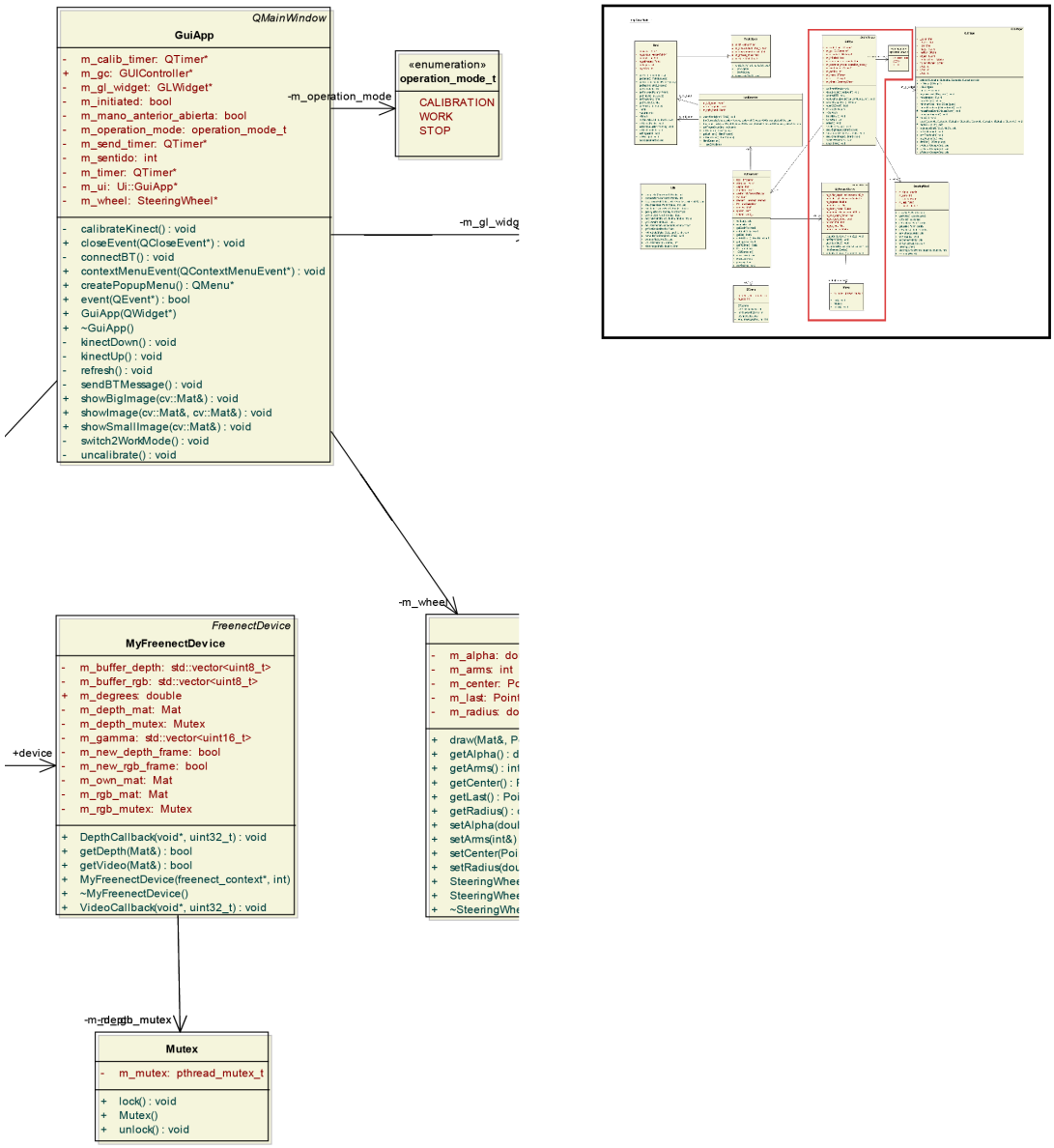


Figura 4.6: Diagrama de clases: parte 3

Diagrama de clases: cuarta parte

La cuarta parte se corresponde con el recuadro marcado con un 4 y completa la explicación del diagarama de clases de la figura 4.3. En él se incluyen las clases *GLWidget* y *SteeringWheel*, ambas está dedicadas a aspectos gráficos de la aplicación.

GLWidget es el resultado de la investigación realizada para mostrar una imagen en 3D del entorno gracias a los datos obtenidos del *Kinect*. Utiliza la biblioteca *OpenGL* para gene-

rar, en un widget aparte, lo que se denomina “nube de puntos” y que se detallará en primer apéndice de esta memoria. Esta clase no aporta funcionalidad alguna al proyecto y procesa la imagen en tres dimensiones justo después de arrancar la aplicación, con lo que el usuario podrá ver el resultado antes de comenzar a calibrar el dispositivo.

SteeringWheel, dibuja el volante que aparece en la pantalla principal (figura 3.19). En su constructor se puede indicar el número de brazos que tendrá el volante, opción cuya única finalidad es estética (son tres brazos o radios del volante al centro del mismo, por defecto). El volante girará a la vez que la mano del usuario y su centro servirá para calcular el ángulo de giro con respecto de esta.

La siguiente figura, muestra estas tres clases ampliadas, para una mejor lectura (fig. 4.7):

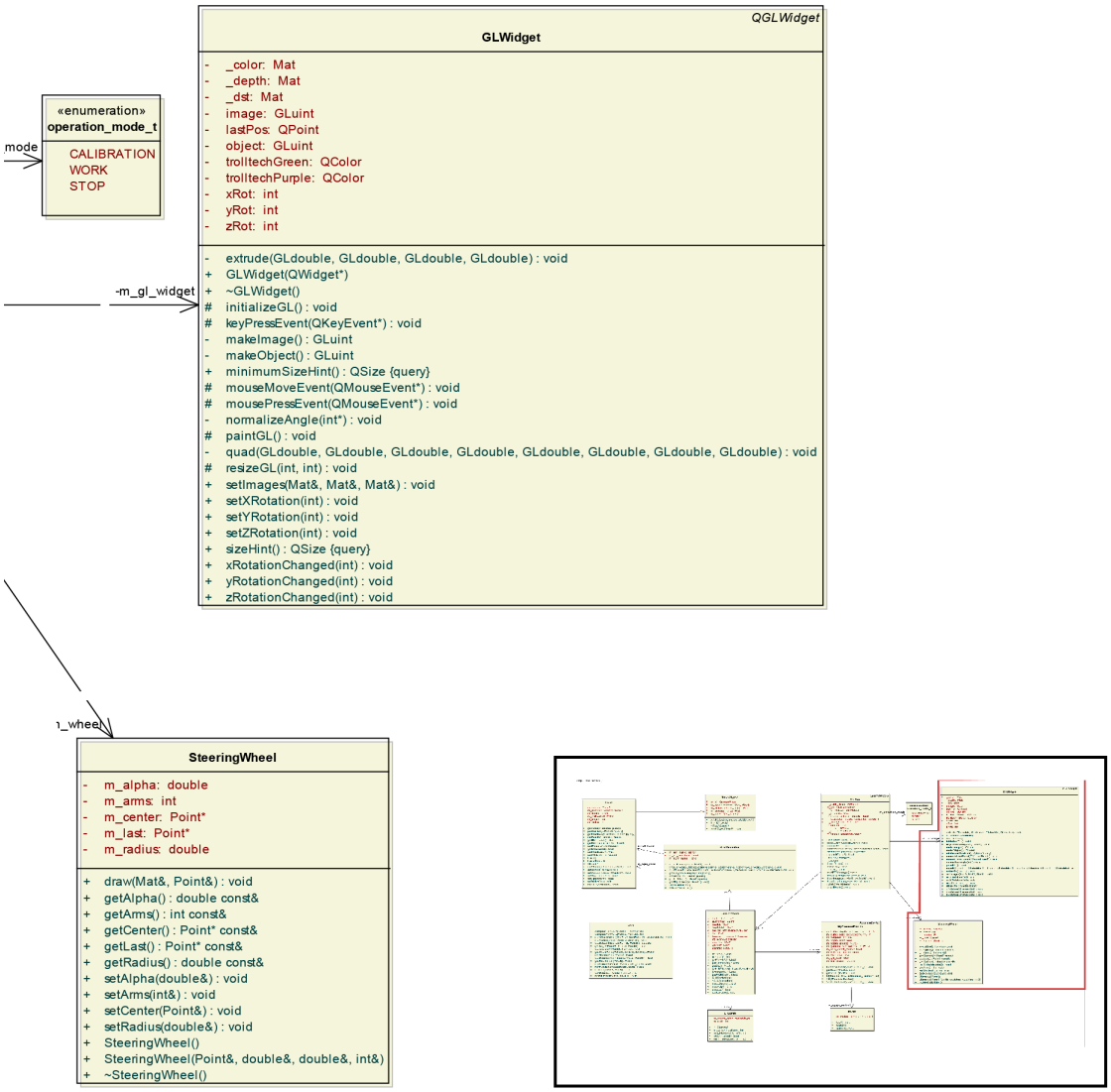


Figura 4.7: Diagrama de clases: parte 4

4.1.3. Aplicación vehículo

La aplicación instalada en el “brick” controlador del vehículo está implementada en Java y destaca por su simplicidad, pues sólo tiene que esperar por una conexión entrante y recibir los datos que le lleguen a continuación. Posteriormente, analizará o parseará estos mensajes entrantes y enviará una orden al vehículo.

Diagrama de clases

En la figura 4.8 aparecen dos clases, una principal, *BTReceive* que carga con todo el proceso de ejecución y otra, *DataThread* que escucha y procesa mensajes de datos entrantes.

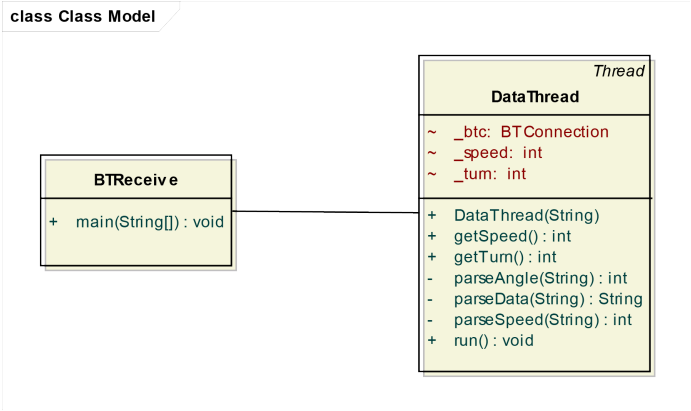


Figura 4.8: Diagrama de clases aplicación vehículo (Java)

El motivo de haber creado una clase aparte que gestione un hilo de escucha es evitar los bloqueos. Aunque las funciones de lectura vía *socket* no son bloqueantes, se producía un cuello de botella en la recepción de datos, pues la lectura de los datos retrasaba el envió de las órdenes al vehículo, además de algunos errores en la recepción. Trasladando la lectura de datos a un hilo aparte, permite que el hilo principal de ejecución envíe solo órdenes al vehículo, las que estén disponibles en ese momento, tras consultar al demonio de escucha.

Además, un detalle importante a tener en cuenta, es que la aplicación envíaba muchos más mensajes de los que el receptor podía leer en el mismo intervalo de tiempo, por lo que se implementó el envío cada 300ms.

La siguiente figura (fig. 4.9), muestra una secuencia de ejecución del programa. Después de establecer la conexión, *BTReceive* lanza un demonio paralelo que leerá periódicamente del *buffer* de entrada de datos del *socket* y parseará los datos recibidos, obteniendo de esta forma, la velocidad y el giro enviados por el equipo cliente. Por su parte, *BTReceive* estará en un bucle infinito enviando órdenes constantes a los servomotores del vehículo y, por cada iteración, pedirá al demonio que le devuelva la última velocidad y giro que leyó del cliente.

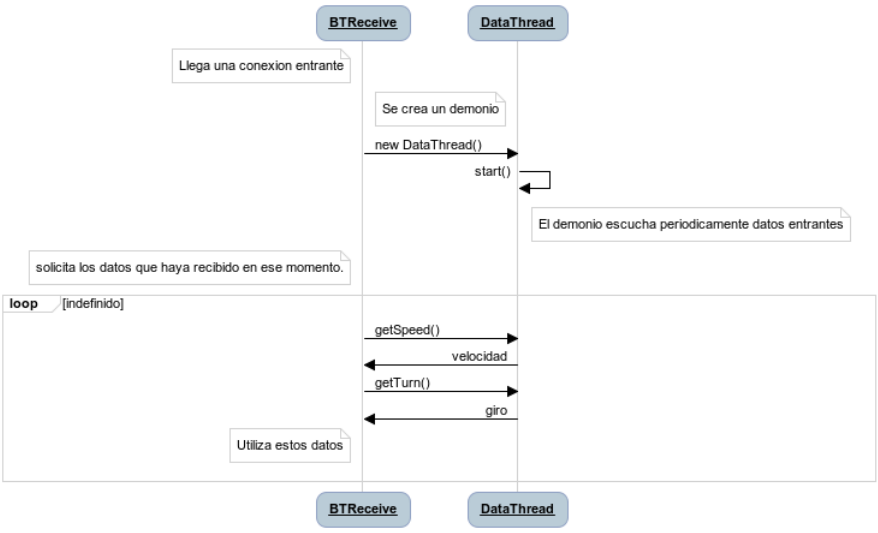


Figura 4.9: Secuencia de ejecución del programa

4.2. Diseño hardware

En base a las restricciones impuestas al vehículo anterior, se buscó un modelo de coche que se ajustara a ellas. El modelo se extrajo de una web[1] dedicada especialmente al diseño de robots con el set de construcciones de *Lego*, pero con ciertas modificaciones, pues el modelo original era más sofisticado en el sentido de que incluía un sensor de color acoplado en el chasis y mirando al suelo, elemento inútil en el proyecto.

4.2.1. Diseño del vehículo

El diseño del vehículo está perfectamente detallado en el apéndice B de este documento, pero podemos avanzar su aspecto y sus características principales.

A continuación, mostraremos algunas vistas del vehículo, en las cuales se aprecia perfectamente la mecánica del coche. Explicaremos aquellas que lo requieran.

Vista frontal

En ella podemos apreciar con claridad la disposición de las ruedas delanteras y del sistema de transmisión de giro, que veremos en la figura 4.14 ampliado.

Figura 4.10: Vista frontal

Vista lateral

Muestra una vista lateral del vehículo. Se aprecia el chasis y cómo se han recogido los cables detrás de lo que podría llamarse “parabrisas” del vehículo.

[1] NXTPrograms: `http://www.nxtprograms.com/NXT2/race_car/index.html`

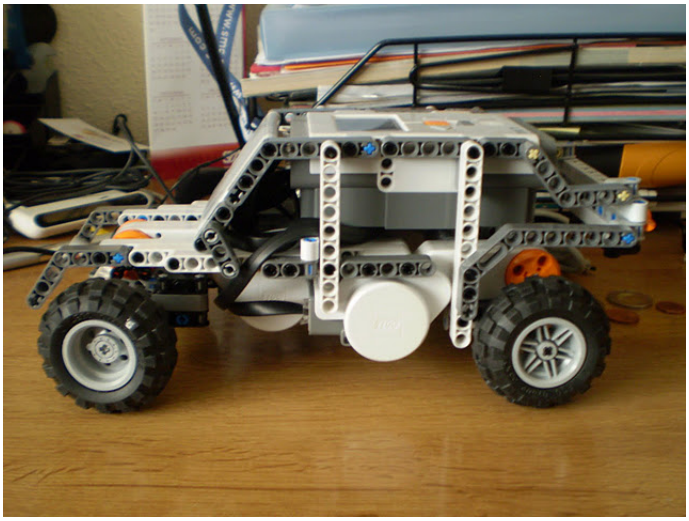


Figura 4.11: Vista lateral

Vista aérea

En esta vista (fig. 4.12) se aprecia el controlador, que contiene el ordenador con el firmware LeJOS, que gobernará el vehículo. Se intuyen los cables entrando en los puertos del dispositivo.

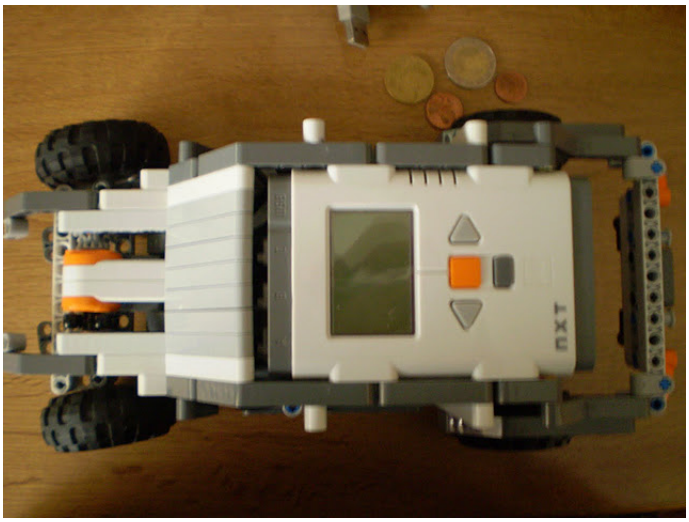


Figura 4.12: Vista aérea

Vista posterior

La vista posterior (fig. 4.13) muestra cómo realiza la transmisión de la marcha a las ruedas. El sistema consiste en un par de engranajes por cada rueda trasera. Ambas se moverán a la misma velocidad.

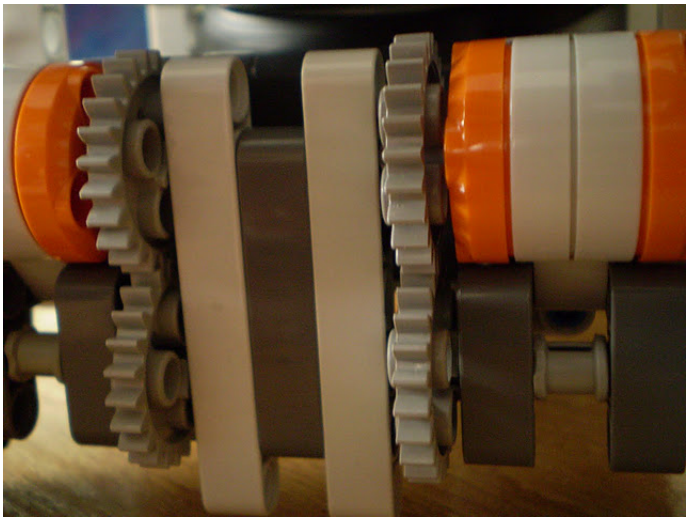


Figura 4.13: Vista posterior

Vista transmisión delantera

La transmisión delantera consiste en dos engranajes en bisel perpendiculares el uno del otro, lo que permite transformar un movimiento circular en el plano vertical en otro en el plano horizontal, facilitando el giro de las ruedas.

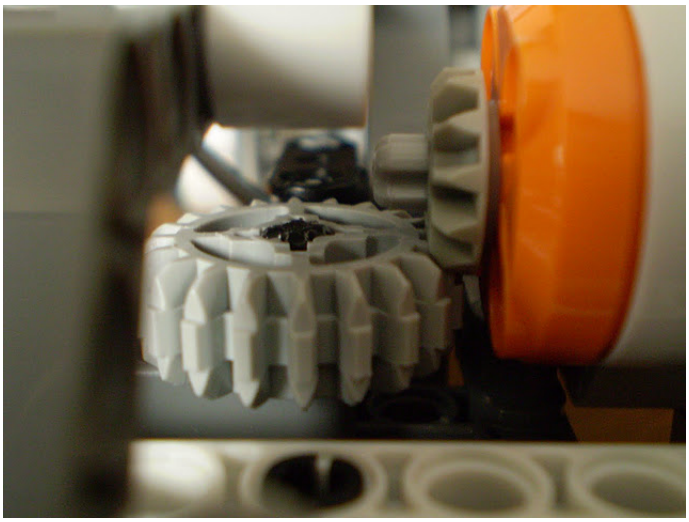


Figura 4.14: Vista transmisión delantera

Capítulo 5

Resultados y conclusiones

En los capítulos anteriores, se hacía un resumen del estado actual de la tecnología y, posteriormente, un análisis profundo de la solución, barajando las posibles alternativas a cada parte del problema y escogiendo una de ellas. El diseño del software y del hardware constituye la etapa siguiente, detallando con diagramas la implementación de las decisiones tomadas en el análisis. Este capítulo se centrará en mostrar los resultados obtenidos de ejecutar el software; se mostrarán imágenes de la ejecución del programa y se acompañarán de una breve explicación.

De todas formas, el resultado del trabajo realizado se puede ver mejor en un vídeo que se incluye junto a esta memoria, en el cual el autor aparece manipulando la aplicación y el vehículo.

5.1. Resultados

En esta sección se expondrán los resultados obtenidos para el cálculo de los dedos de la mano, el ángulo de giro y el filtro de Kalman, ya que esta es la parte visible en el programa cliente. Como ya hemos dicho, una demostración del movimiento del vehículo en tiempo real está disponible adjunto a esta memoria y supone el resultado más concluyente.

Los resultados obtenidos en el análisis visual obtienen mejores resultados dentro de un rango de distancia, entre 85 y 105 centímetros con respecto del cuerpo (brazos extendidos) al *Kinect*, ya que a distancias más cercanas, al extender el brazo puede quedar demasiado cerca y no aparecer la mano al completo en la imagen y a distancias más lejanas las diferencias en los contornos son más pequeñas y pueden ser omitidas por el programa (contornos más pequeños, por lo que no se consideran como manos).

5.1.1. Extracción de dedos

La extracción de los dedos de la mano ha sido muy satisfactoria, pues reconoce perfectamente cuántos dedos está mostrando el usuario. Las siguientes figuras ilustran diferentes posiciones para indicar una misma marcha al vehículo:

- **Un dedo:**

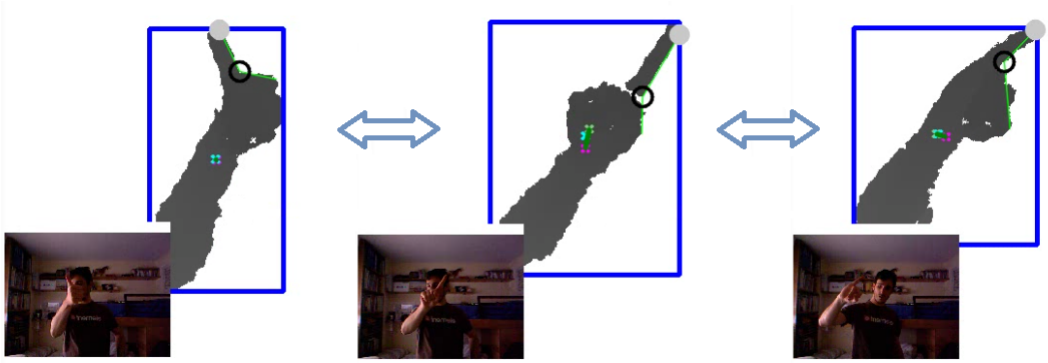


Figura 5.1: Primera marcha

- **Dos dedos:**

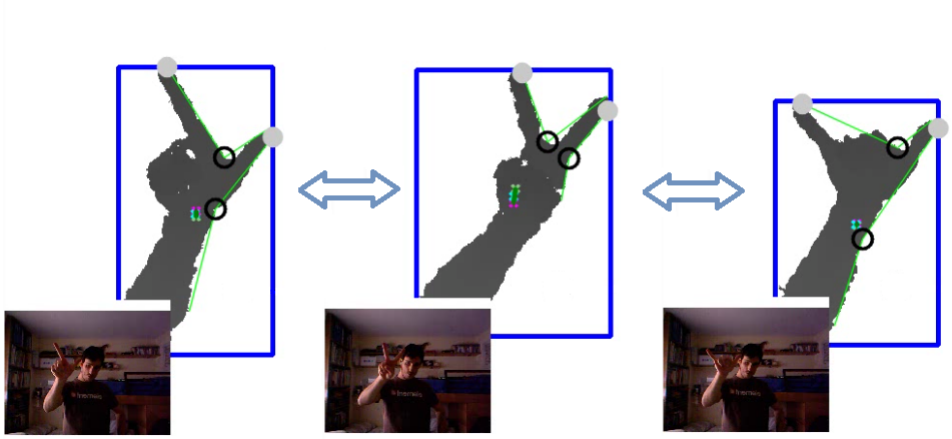


Figura 5.2: Segunda marcha

- **Tres dedos:**

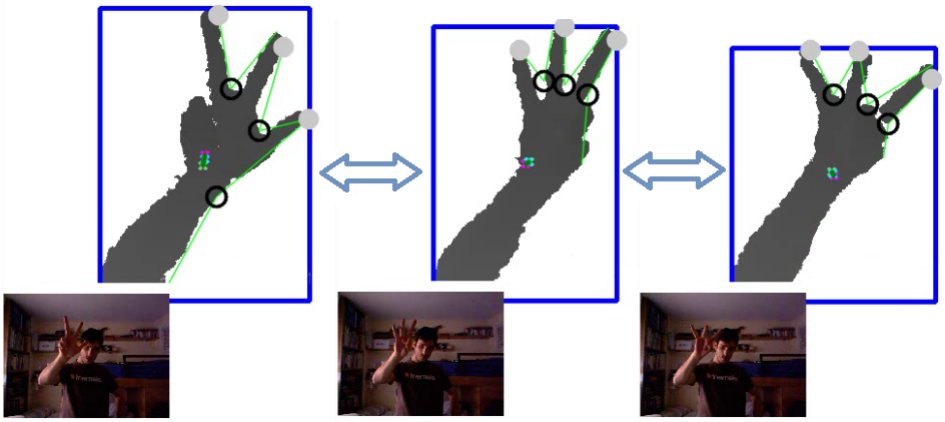


Figura 5.3: Tercera marcha

- **Cuatro dedos:**

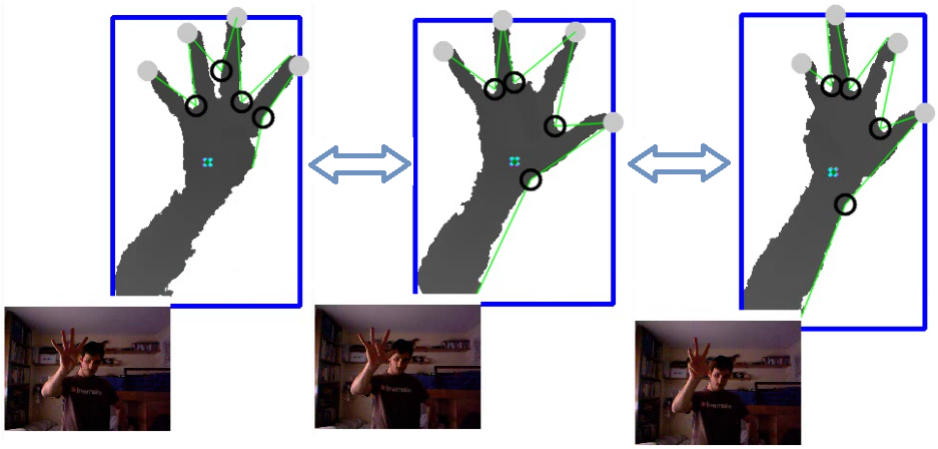


Figura 5.4: Cuarta marcha

- **Valor máximo (5) y mínimo (0):**

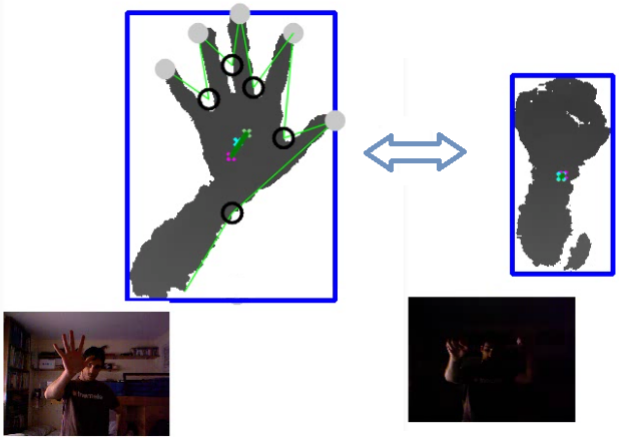


Figura 5.5: Quinta marcha y ninguna marcha puesta

Como se puede ver, la posición y el orden de los dedos de la mano no es fija y el usuario puede indicar la marcha como más le plazca, siempre y cuando indique el número que realmente quiere que el sistema interprete.

5.1.2. Sentido del vehículo

Este punto fue sencillo de implementar, pues se considera que se cambia de sentido de la marcha cuando el usuario abre por completo la mano derecha. En la figura 5.6 se muestra la situación en la que la mano está abierta (se reconocen cinco dedos abiertos) y el indicador de la marcha pasa a ser negativo (pasa de ir hacia delante a ir hacia detrás).

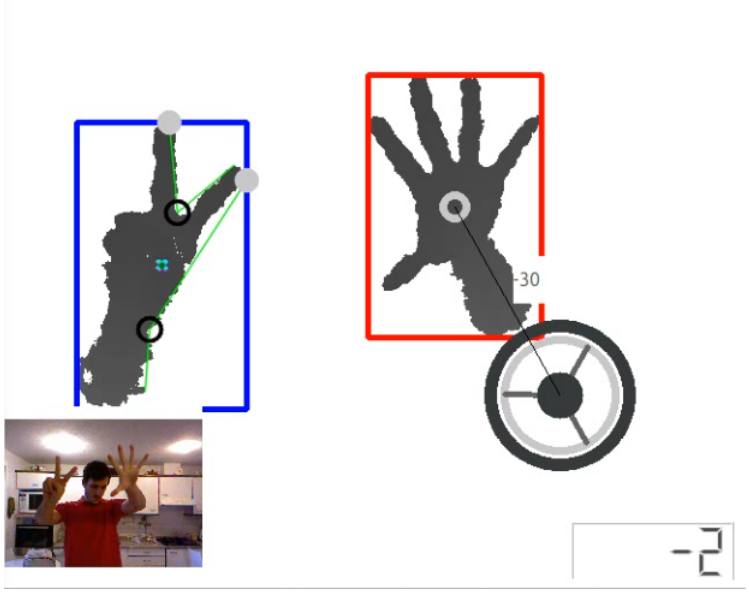


Figura 5.6: Inversión del sentido de la marcha

5.1.3. Seguimiento de la mano

Dado que la mano izquierda es objeto de *"tracking"*, se dibujan en la imagen los puntos "estimado" (cruz de color azul claro), "predicho"(cruz de color amarillo) y "real"(cruz de color rosada), donde el punto predicho es el punto estimado con respecto al *frame* anterior y el estimado como tal es el resultado de corregir la predicción realizada en base al punto real medido. La línea verde une los tres puntos formando una cadena.

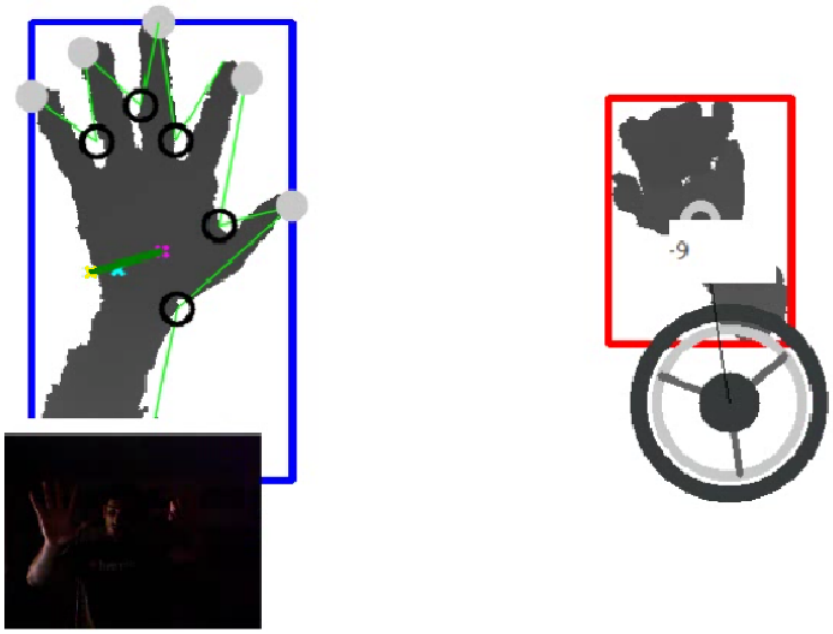


Figura 5.7: Seguimiento mano izquierda: código de colores

El punto real se corresponde con el centro del contorno y sirve para corregir la predicción realizada por el filtro. En este caso se observa como el movimiento de la mano es hacia la derecha y la predicción del filtro queda un poco más a la izquierda del punto real que indica donde está la mano, pero es corregida por el filtro y obtiene una nueva posición entre los otros dos puntos, la estimada, en color azul claro.

5.1.4. Ángulo de giro

El resultado de girar el brazo derecho sobre el volante trazando arcos es un ángulo de giro con respecto al eje horizontal, considerando ángulo 0 cuando la mano y el centro del volante forman una recta perpendicular a dicho eje, es decir 90 grados.

La siguiente ilustración muestra la vista del programa y al usuario, con lo que se puede comprobar cómo coinciden los gestos y el vehículo se encuentra en fase de giro. Es difícil apreciar perfectamente el giro del vehículo pues un vídeo lo ilustra mucho mejor (incluído en la memoria).

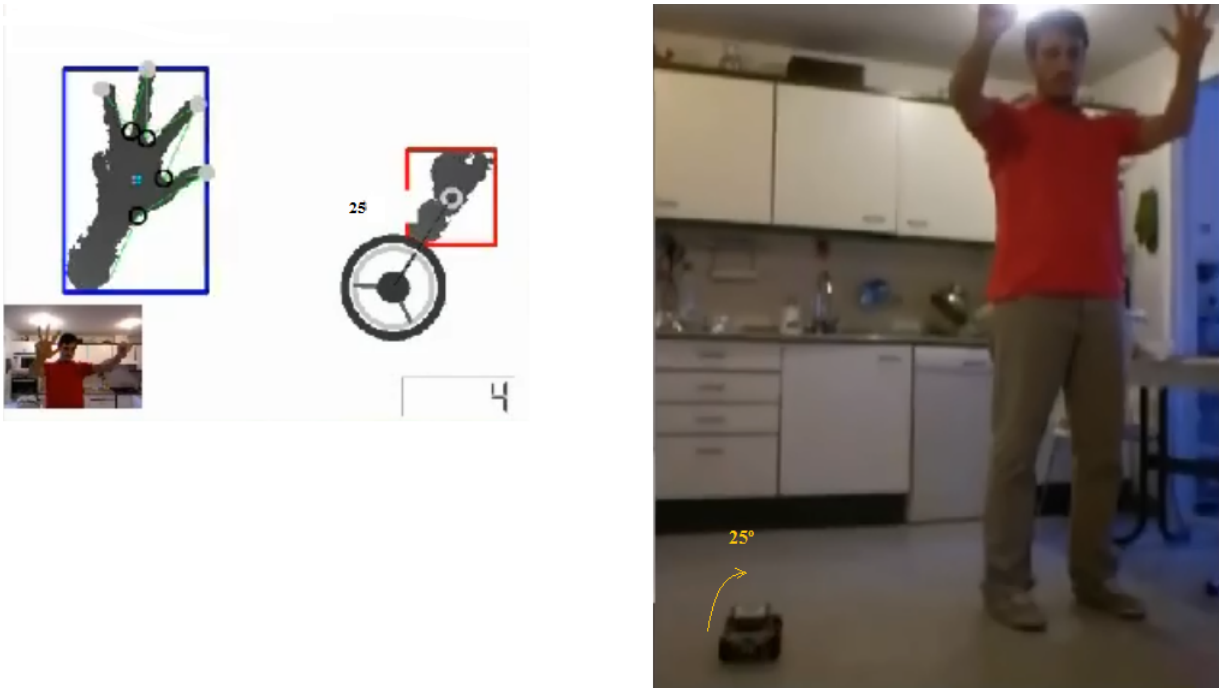


Figura 5.8: Ejemplo de giro

5.2. Conclusiones

Llegados al final de esta memoria, procede hacer balance del sistema implementado y de la experiencia adquirida a lo largo del tiempo que ha durado el proyecto.

En lo que a la parte técnica se refiere, los objetivos han sido cumplidos, pues se ha realizado un estudio completo de las posibilidades que ofrece *Kinect*, se ha buscado un caso práctico para aplicar lo que esta tecnología nos ofrece y se ha implementado con éxito. El resultado es un vehículo controlable a voluntad por cualquier usuario que disponga de un *Kinect* y un vehículo *Lego* de características similares.

Considero que la librería o biblioteca *Open Source libfreenect* es muy limitada en cuanto a funcionalidad, pues sólo aporta acceso a los componentes del dispositivo *Kinect* y que para futuros trabajos sería interesante utilizar una más avanzada como es *OpenNI+NITE* que incorpora algoritmos avanzados de *tracking* y esqueletización, muy útiles a la hora de determinar las partes del cuerpo.

Muchas han sido las dificultades encontradas a lo largo del camino, pues nunca había tratado con temas relacionados con la visión por computador y aprender todas las técnicas me tomó bastante tiempo y estudio (el filtro de Kalman en especial). En la parte de la comunicación vehículo-equipo, ha sido fundamental el conocimiento adquirido durante la carrera,

sobre todo durante la asignatura de Redes II, aunque hubo que complementarlo con el estudio de la tecnología *Bluetooth*.

Mirando ya hacia la parte personal, ha sido un proyecto muy enriquecedor, pues he descubierto campos que tenía un poco desterrados por creer no ser de mi agrado, como es de las redes. El estudio de las tecnologías y sobre todo ver que tu aplicación produce resultados (¡se mueve!) ha sido muy satisfactorio.

Para finalizar, este proyecto ha hecho que mi interés por seguir trabajando con el *Kinect* aumente, sobre todo me gustaría adentrarme en el mundo del modelado 3D, por lo tanto, puedo afirmar que ha sido una magnífica inversión.

Si tuviera que ampliar este proyecto, trataría de utilizar técnicas de inteligencia artificial para la clasificación de los objetos en las imágenes, como una red neuronal.

Por tanto, como **líneas futuras de investigación** planteo un sistema de reconstrucción facial basado en los datos aportados por nuestro escáner 3D, es decir, *Kinect* y por qué no, la realización de algún videojuego para ordenador, los cuales empieza ya a aparecer[1].

[1]Zombie Holdout: `http://kinect.dashhacks.com/zombie-holdout`

Bibliografía

[1] Andrews, G.R. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 1999.

[2] R. González. *Digital Image Processing*. Prentice Hall.

[3] *OpenKinect Project documentation*. Disponible [Internet]: `http://openkinect.org/wiki/Documentation` [15-10-2011]

[4] Tou, J.T. y González, R.C. *Pattern Recognition Principles, 2nd edition*. Addison-Wesley, 1977.

[5] A. K. Jain, R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall 1988.

[6] Gary Bradski, Adrian Kaehler. *Learning OpenCV: computer vision with the OpenCV library*. O'Reilly.

[7] *OpenCV 2.1 Cheat Sheet*. Disponible [Internet]: `http://opencv.willowgarage.com/wiki/Welcome?action=AttachFile&do=get&target=opencv_cheatsheet.pdf` [28-04-2011]

[8] Álvaro Cassinelli, Stéphane Perrin,Masatoshi Ishikawa. *Smart Laser-Scanner for 3D Human-Machine Interface*. Disponible [Internet]: `http://www.k2.t.u-tokyo.ac.jp/fusion/LaserActiveTracking/I-02-cassinelli.pdf`

[9] Cristina Manresa, Javier Varona, Ramón Mas and Francisco J. Perales. *Real–Time Hand Tracking and Gesture Recognition for Human-Computer Interaction*. Disponible [Internet]: `http://dmi.uib.es/~ugiv/papers/ELCVIAManresa.pdf` [2-08-2011]

[10] Tou, J.T. y González, R.C. *Pattern Recognition Principles, 2nd edition*. Addison-Wesley, 1977.

[11] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, Ivan Poupyrev. *3D User Interfaces: Theory and Practice*. Addison-Wesley Professional, 2004

[12] *Rafael Molina Soriano. Bases Del Filtro de Kalman*. Disponible [Internet]: `http://decsai.ugr.es/vip/doctorado/pvd/T7bn.pdf` [10-09-2011]

[13] *Kalman Filter Made Easy*. Disponible [Internet]: `http://www.ocf.berkeley.edu/~tmtong/kalman.php` [25-09-2011]

[14] W. Richard Stevens. *UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI*. Prentice Hall, 1998

[15] Bryan Hall. *Beej's Guide to Network Programming*. Jorgensen Publishing, 2009

[16] Albert Huang. *An Introduction to Bluetooth Programming*. Disponible [Internet]: `http://people.csail.mit.edu/albert/bluez-intro/` [17-08-2011]

Apéndice A

Modelado tridimensional del entorno

Según se avanzó en la investigación con Kinect y conforme se estudiaban otros proyectos, la mayoría de estos aprovechaban, como es lógico, la capacidad de representar las imágenes como escalas de grises, pues al fin y al cabo, es lo mismo que asignar un valor de profundidad, esto es, obtener una imagen en tres dimensiones.

Muchos de los usos que se le pueden dar están relacionados con el dise˜no 3D, ya que estas imágenes de puntos con profundidad pueden ser exportados a formatos de archivo de programas dedicados al dise˜no gráfico.
Esto es una enorme ventaja, pues el modelar en 3D es una tarea ardua y siempre es una ventaja que alguien modele por ti. Para este proyecto, y sin que tenga relación con el objetivo del mismo, se ha implementado un prototipo de lo que se denomina "nube depuntos", como un complemento extra que en un futuro pueda ser explotado en algún sentido.

A.1. Nube de puntos

De forma breve, se puede definir como un conjunto de vértices en un sistema de coordenadas tridimensional, esto es cada vértice está definido por tres coordenadas (X, Y, Z) y, por lo general, suele ser una representación de la superficie externa de un objeto. A cada punto tridimensional, también se le conoce por el nombre de vóxel. En la foto (figura **??**), podemos ver un ejemplo de una nube de puntos cualquiera de una cocina (el grosor de cada vóxel puede variar según lo establezca el usuario, dando un aspecto más tosco o más fino).

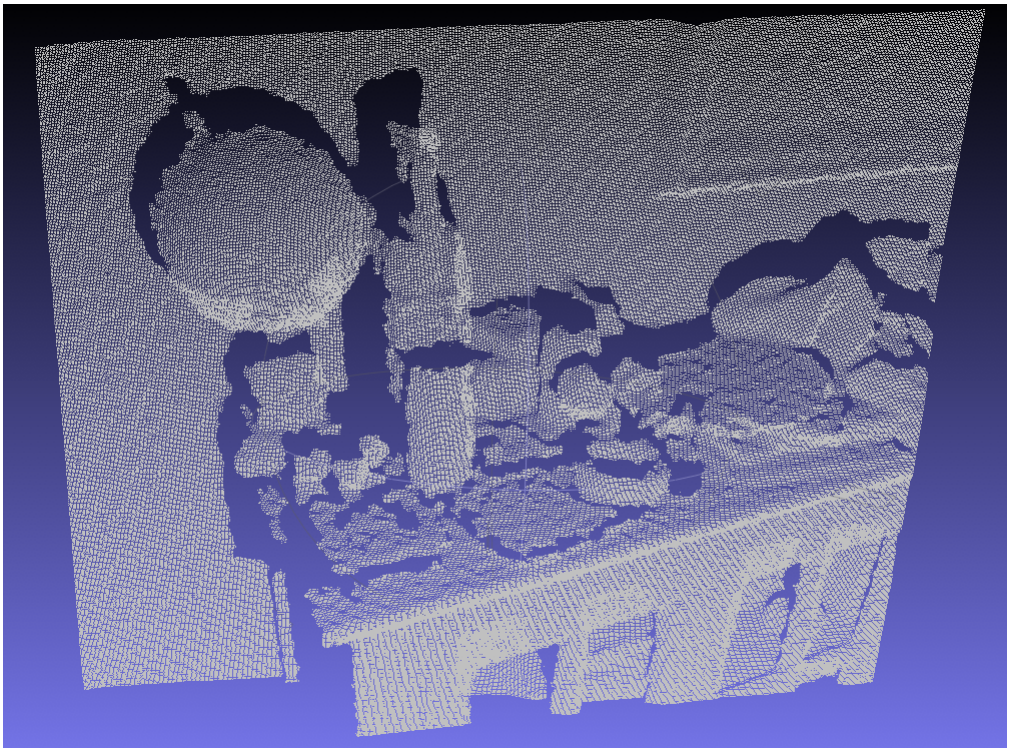


Figura A.1: Nube de puntos de una cocina con grano grueso

A.2. Resultados obtenidos

Para visualizar la nube[1], se ha utilizado la biblioteca *OpenGL*, famosa por ser muy utilizada en el mundo del videojuego y permitir crear entornos tridimensionales.

Estos son los resultados obtenidos:

[1] Vídeo completo: `http://www.youtube.com/watch?v=yA7rFC4SOlw`

Figura A.2: Vista frontal

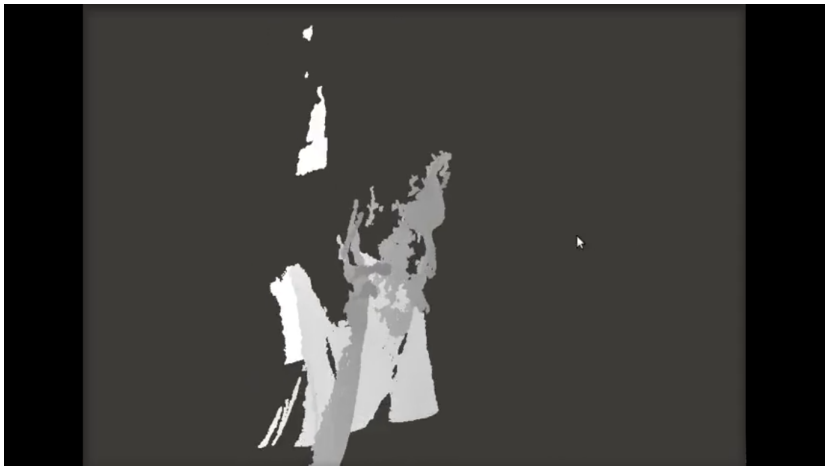


Figura A.3: Lateral 2

Así pues, vemos cómo es posible crear modelos 3D a partir de los datos de profundidad recogidos. La otra parte, la de modelar, se vuelve una tarea muy complicada pues habría que exportar estos puntos al formato de fichero de algún programa de diseño, con la consiguiente complejidad.

Apéndice B

Características y montaje del RACE CAR de Lego® Mindstorms® NXT 1.0

B.1. Introducción

Para el desarrollo del proyecto se eligió como elemento telecontrolable el Race Car de Lego® dada su versatilidad de construcción, que ha facilitado su personalización, y fundamentalmente por poseer un interface de control inalámbrico basado en Bluetooth que nos ha permitido su utilización como elemento demostrador del verdadero objetivo de nuestro proyecto, que no era otro que el de crear una aplicación de telecontrol mediante el dispositivo Kinect®.

El Race Car está diseñado para que su conducción sea como la de un coche real, con control de dirección en las ruedas delanteras y de tracción con regulación de velocidad en las ruedas traseras, mediante engranajes.

Debido a la complejidad mecánica de algunas de las piezas empleadas en el diseño de la dirección de este coche, especialmente los engranajes negros cónicos de 20 dientes (reacción de engranajes), se hace necesario antes de iniciar la marcha el ajuste de la dirección de las ruedas delanteras, asegurándose de que apuntan directamente hacia adelante.

Figura B.1: Vista general

B.2. Instrucciones de montaje

En el presente apartado exponemos las principales fases de montaje del vehículo mediante imágenes en las que se muestran las piezas utilizadas en cada fase y el resultado final del montaje en cada una de las fases.

Las instrucciones de montaje completas están disponibles en el sitio oficial de *NXTPrograms*[1].

Como puede observarse a lo largo de las diferentes fases del montaje algunas piezas del diseño del Race Car son puramente "decorativas" y se incorporan únicamente para darle una apariencia más real al coche. Así pues manteniendo las piezas esenciales se podría personalizar el aspecto final del vehículo sin más que utilizar otras piezas del amplio abanico que ofrece el sistema LEGO.

[1]Instrucciones completas: `http://nxtprograms.com/NXT2/race_car/steps.html`

Fase 1

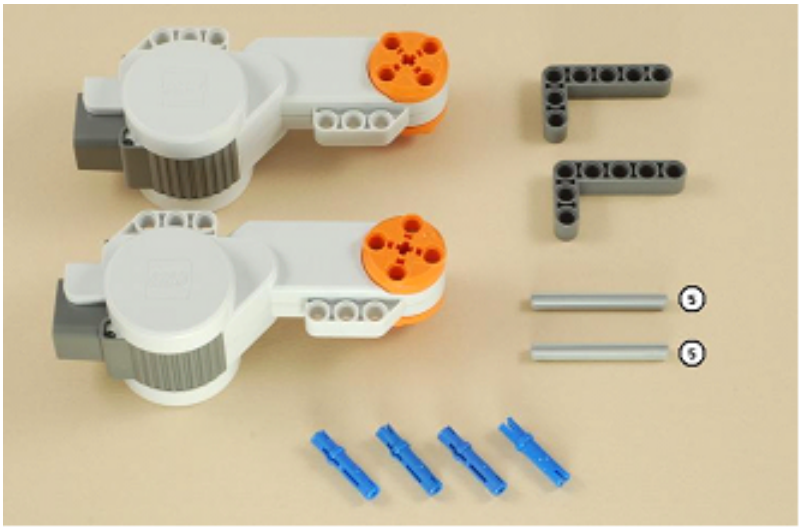


Figura B.2: Paso 1

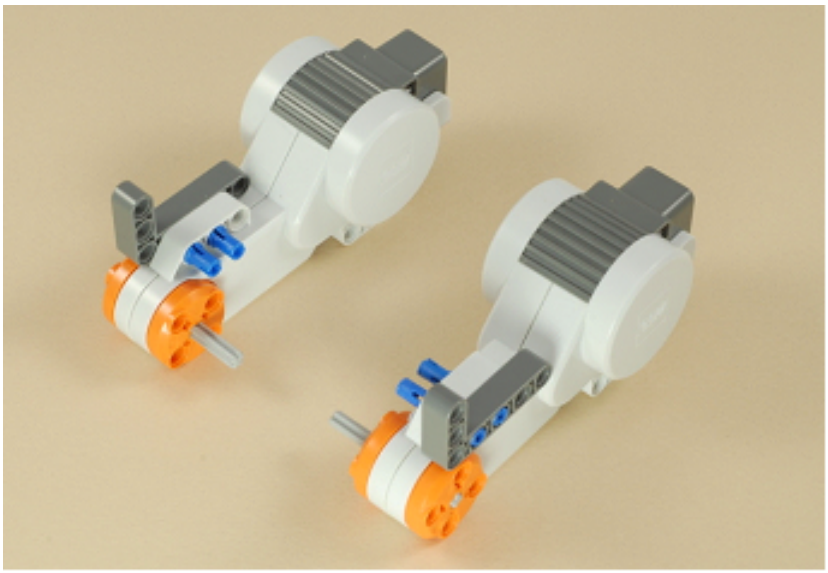


Figura B.3: Paso 2

Fase 2

Figura B.4: Paso 3

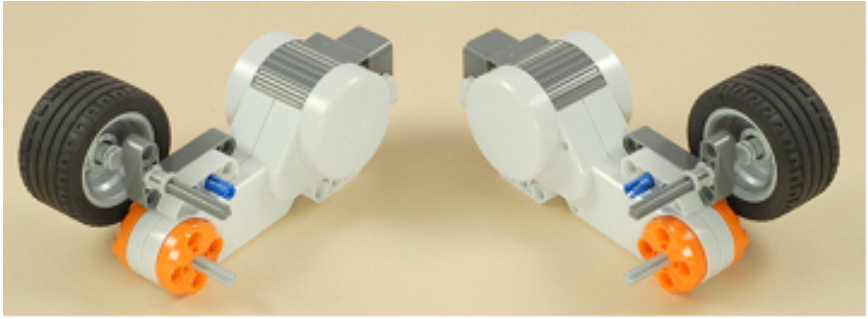


Figura B.5: Paso 4

Fase 3

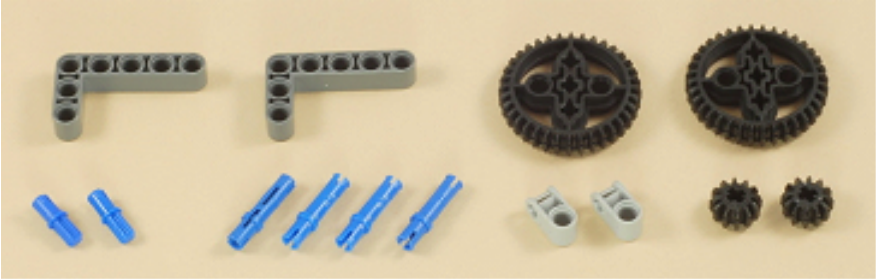


Figura B.6: Paso 5

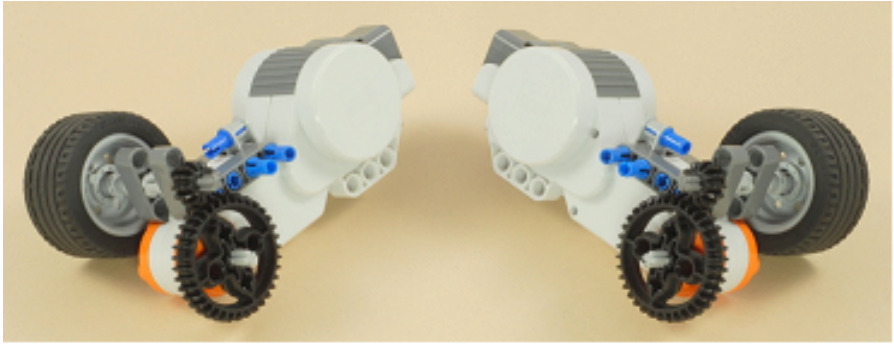


Figura B.7: Paso 6

Fase 4

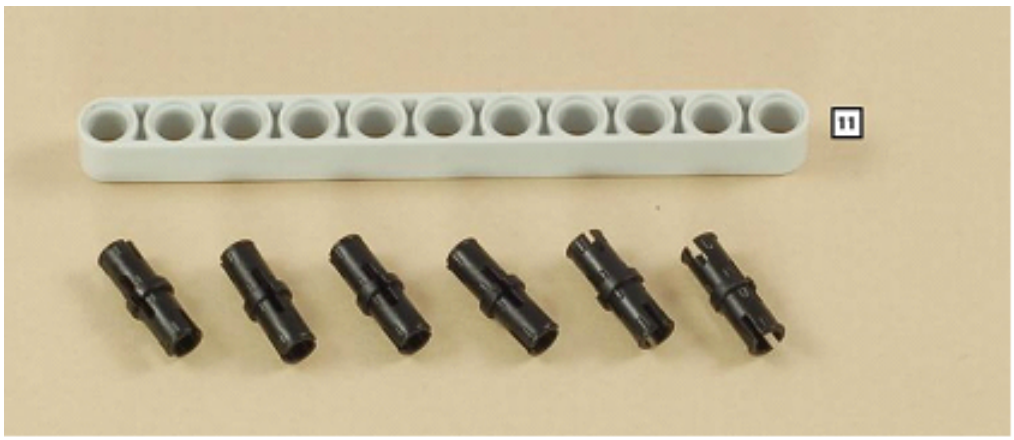


Figura B.8: Paso 7

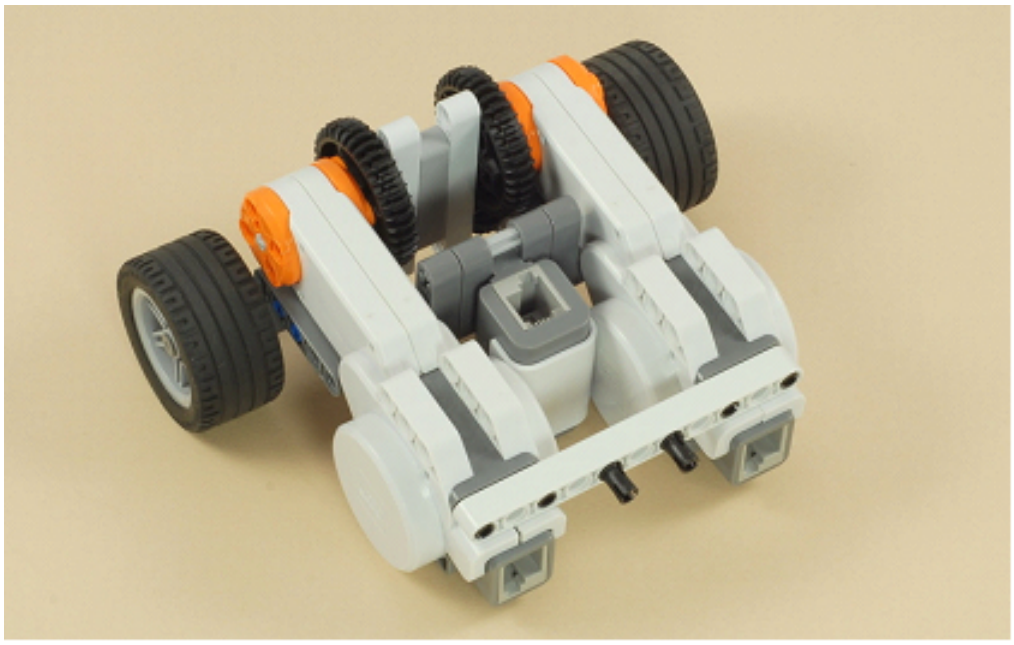


Figura B.9: Paso 8

Fase 5

Figura B.10: Paso 9

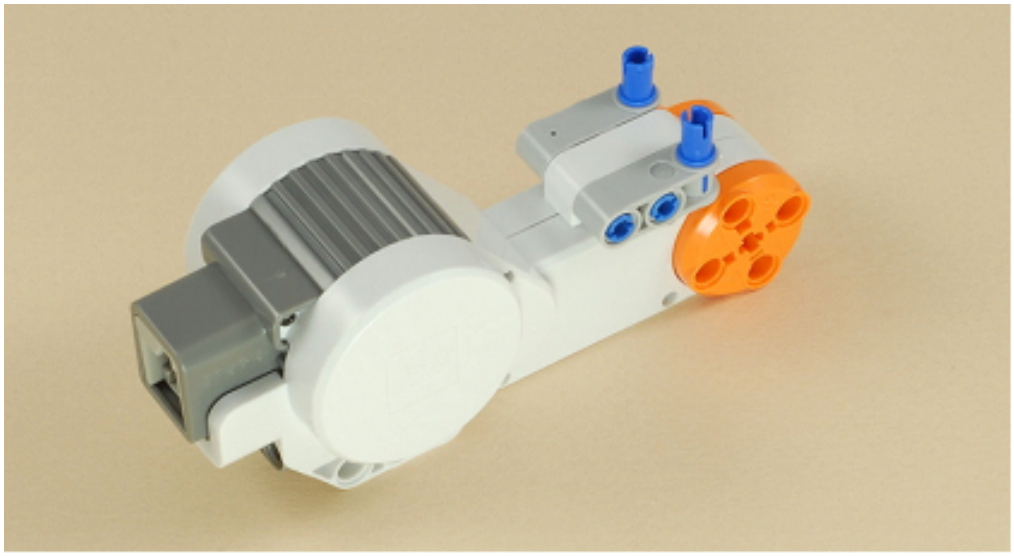


Figura B.11: Paso 10

Fase 6

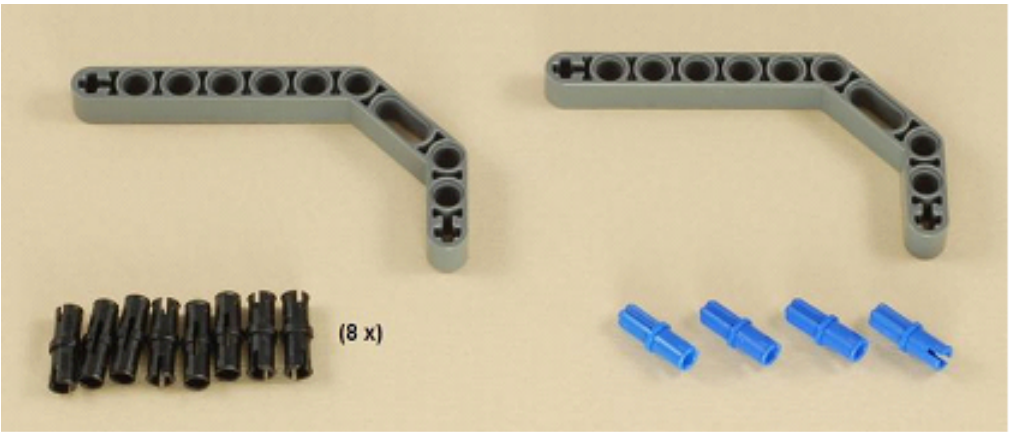


Figura B.12: Paso 11

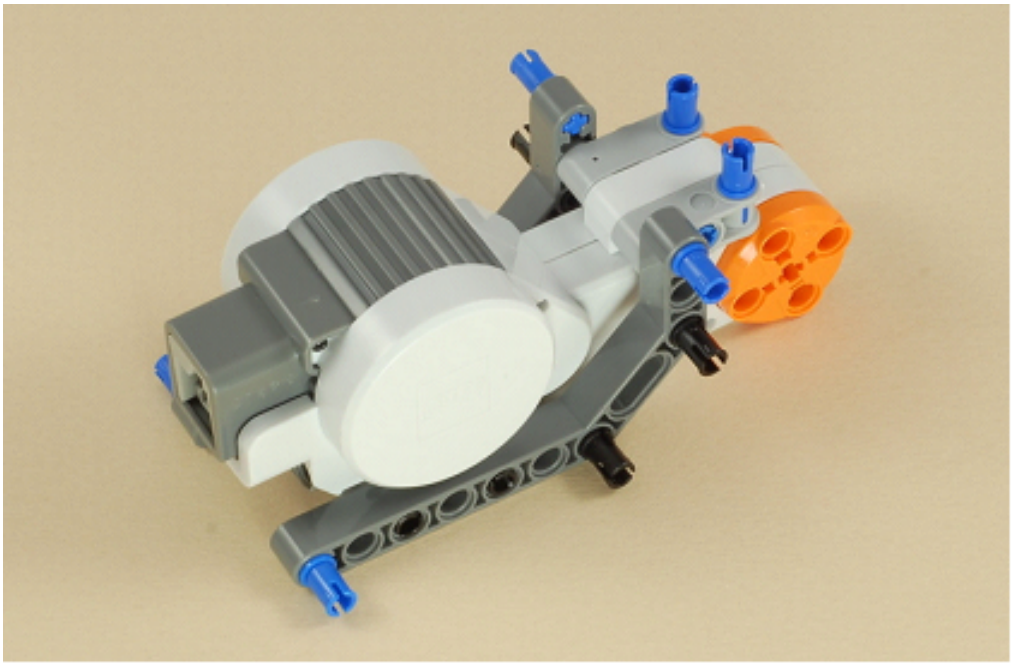


Figura B.13: Paso 12

Fase 7

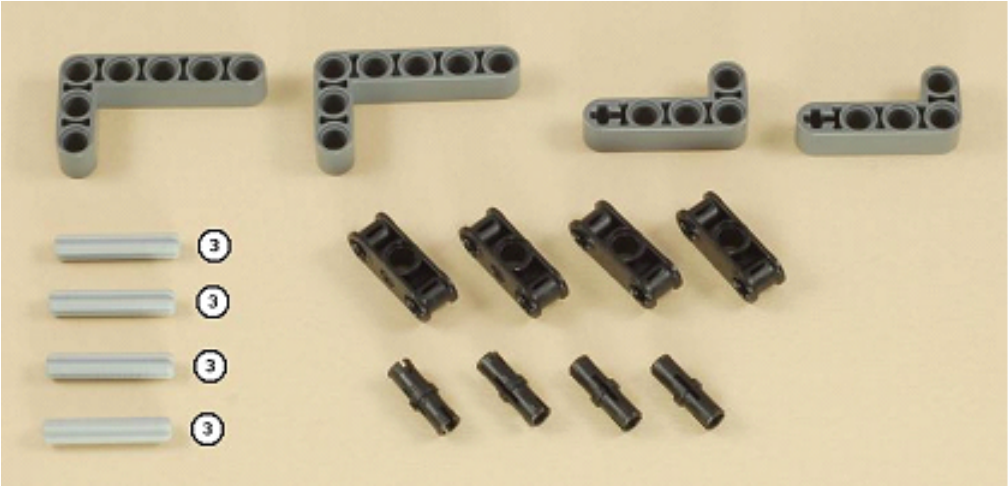


Figura B.14: Paso 13

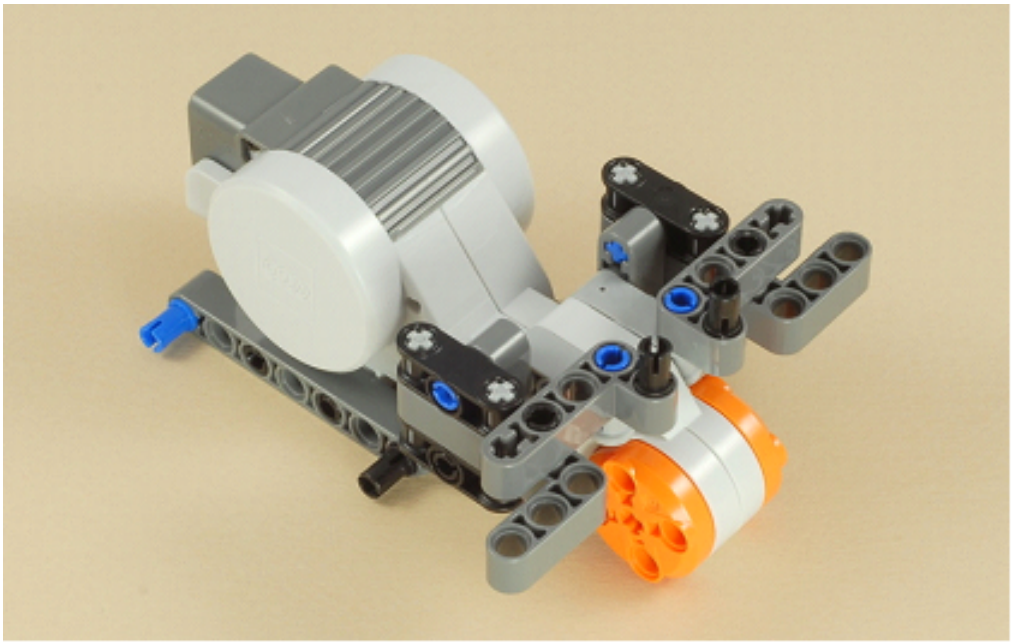


Figura B.15: Paso 14

Fase 8

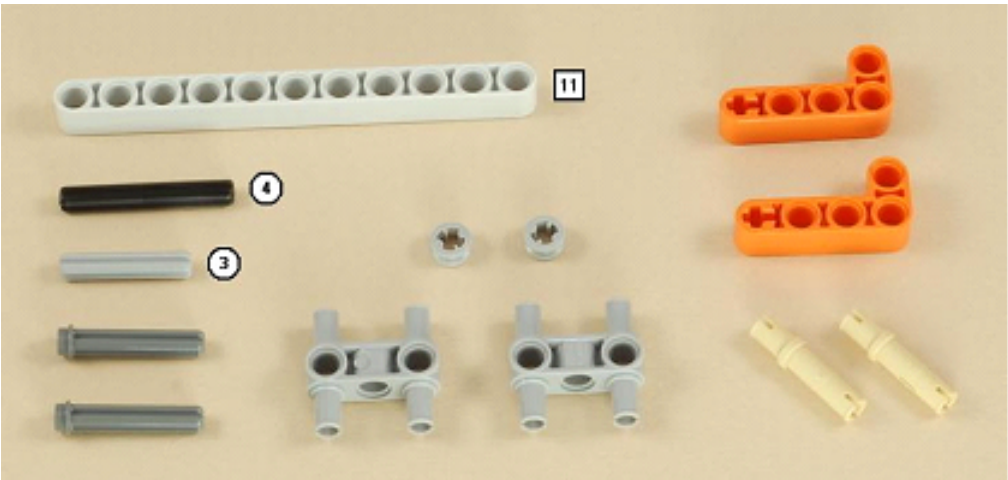


Figura B.16: Paso 15

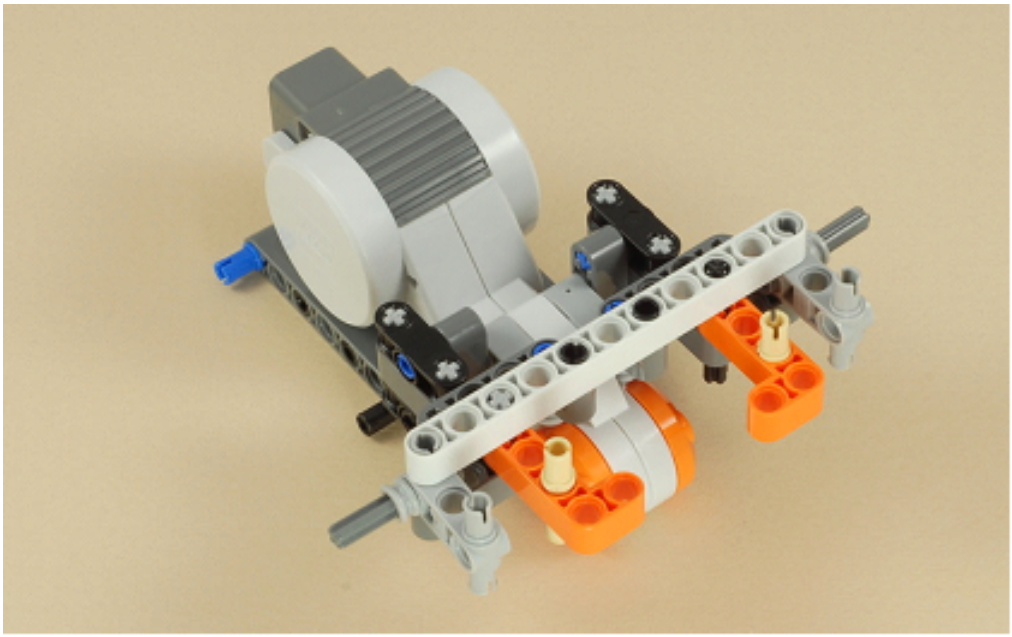


Figura B.17: Paso 16

Fase 9

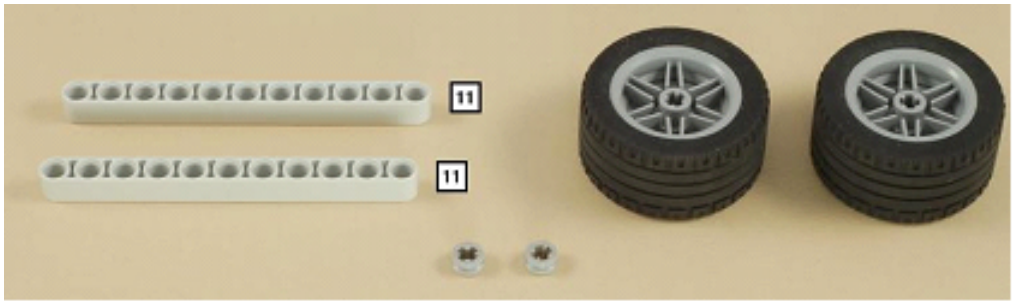


Figura B.18: Paso 17

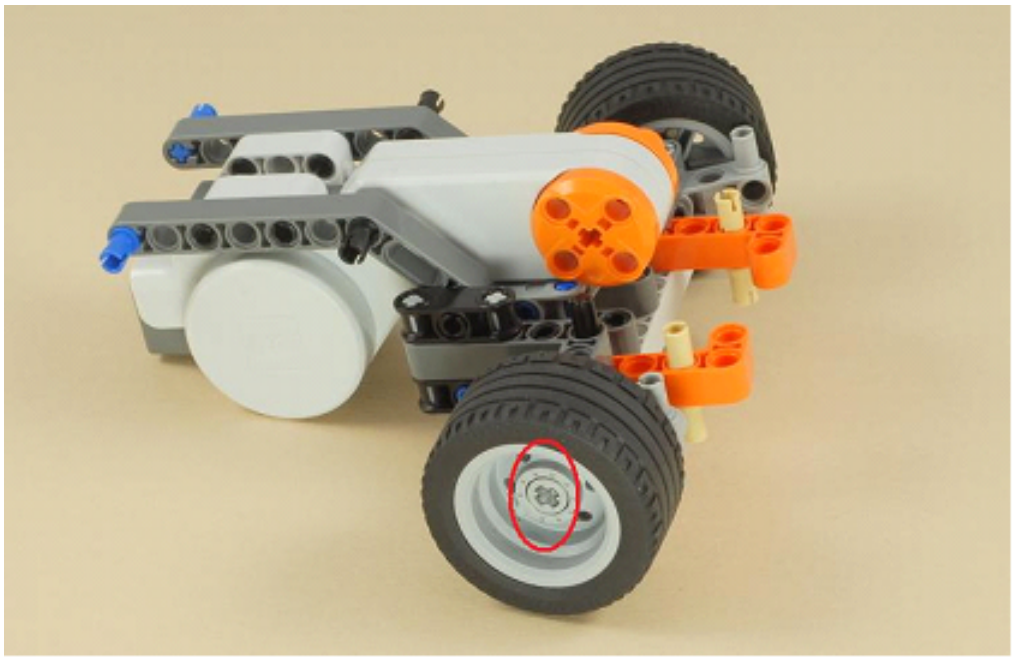


Figura B.19: Paso 18

*

Fase 10

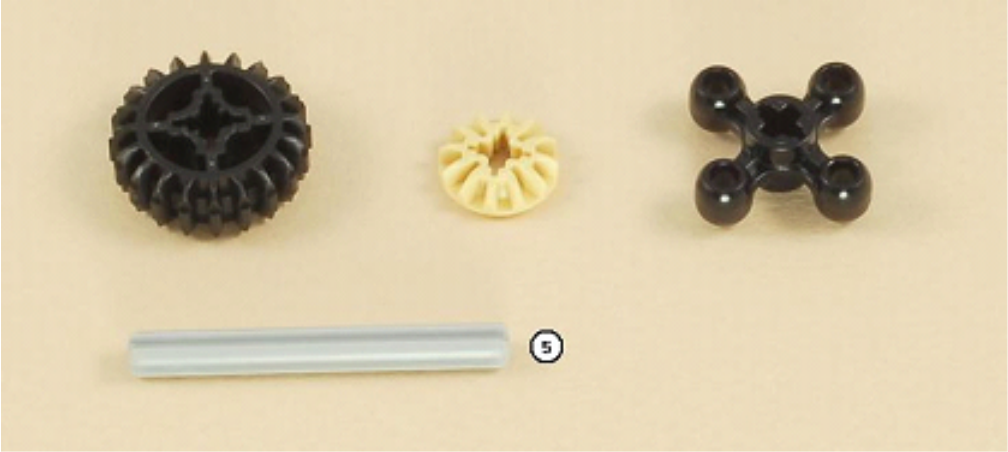


Figura B.20: Paso 19

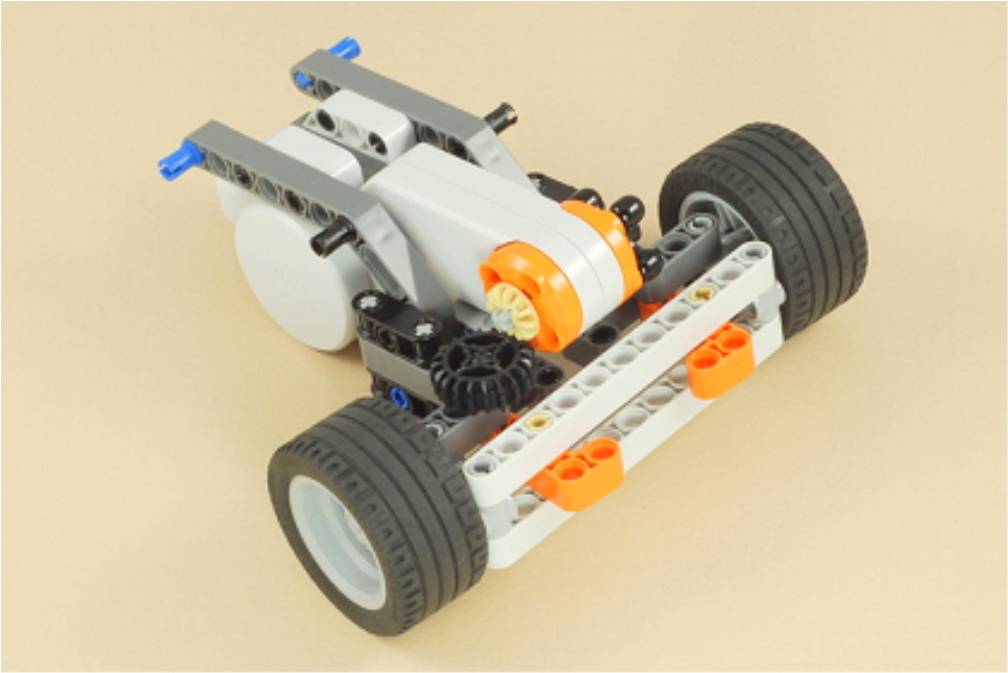


Figura B.21: Paso 20

Fase 11

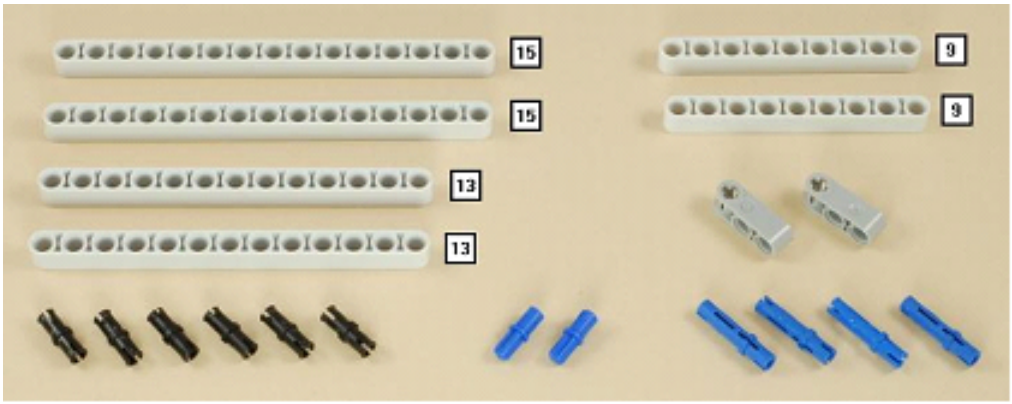


Figura B.22: Paso 21

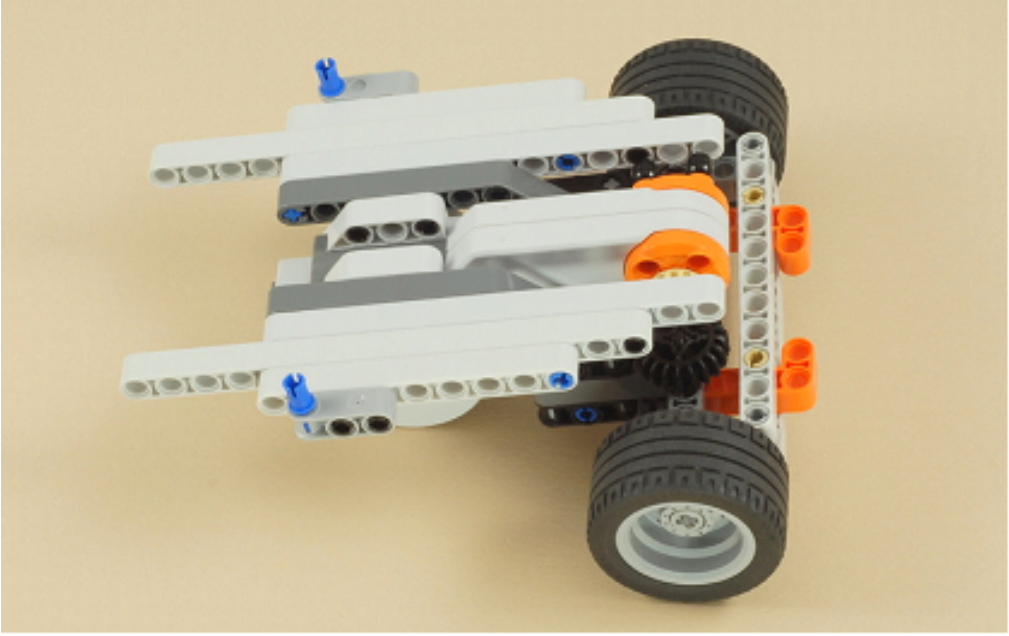


Figura B.23: Paso 22

Fase 12

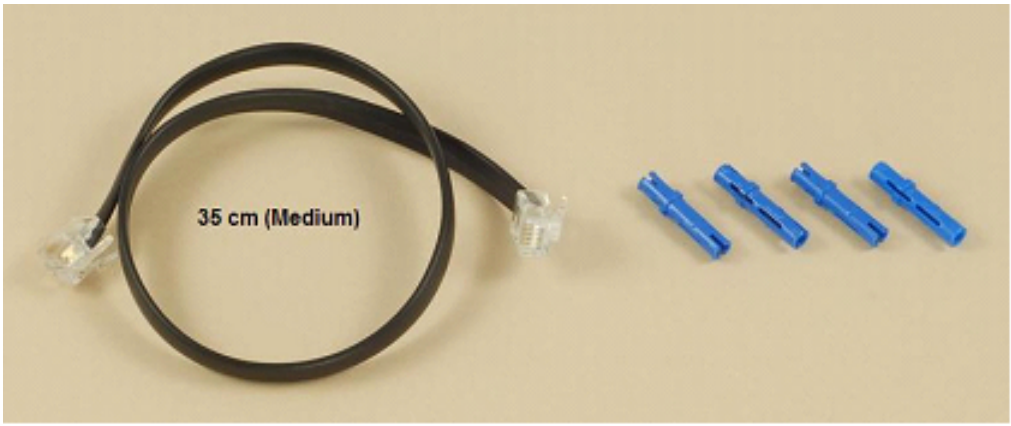


Figura B.24: Paso 23

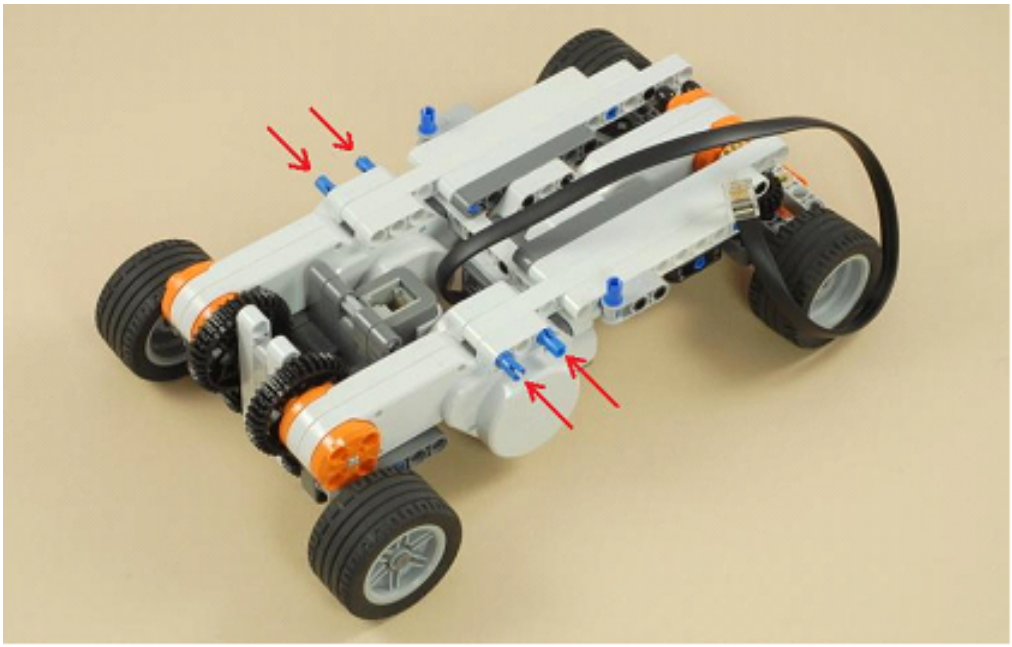


Figura B.25: Paso 24

Fase 13

Figura B.26: Paso 25

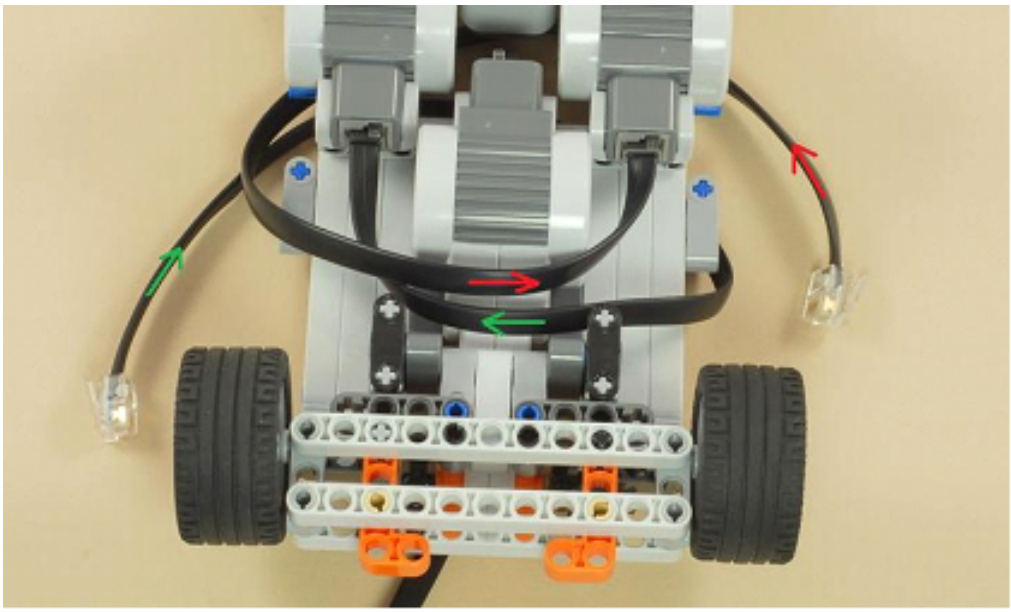


Figura B.27: Paso 26

Fase 14

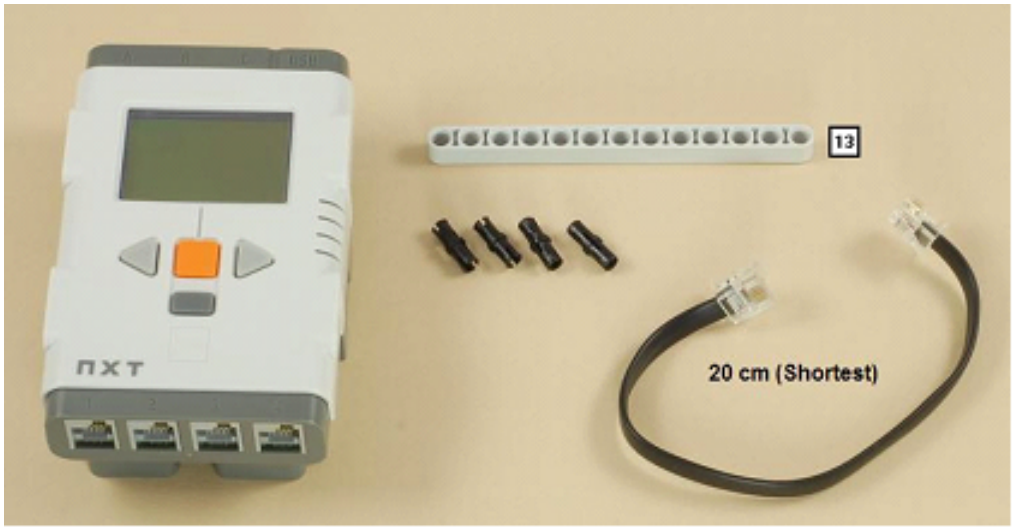


Figura B.28: Paso 27

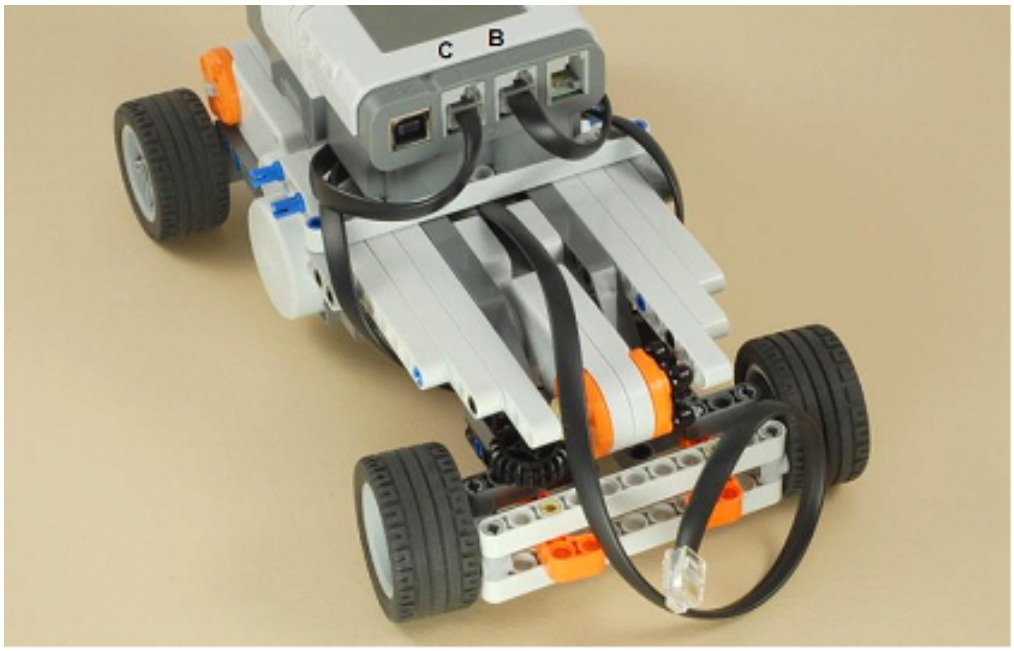


Figura B.29: Paso 28

Fase 15

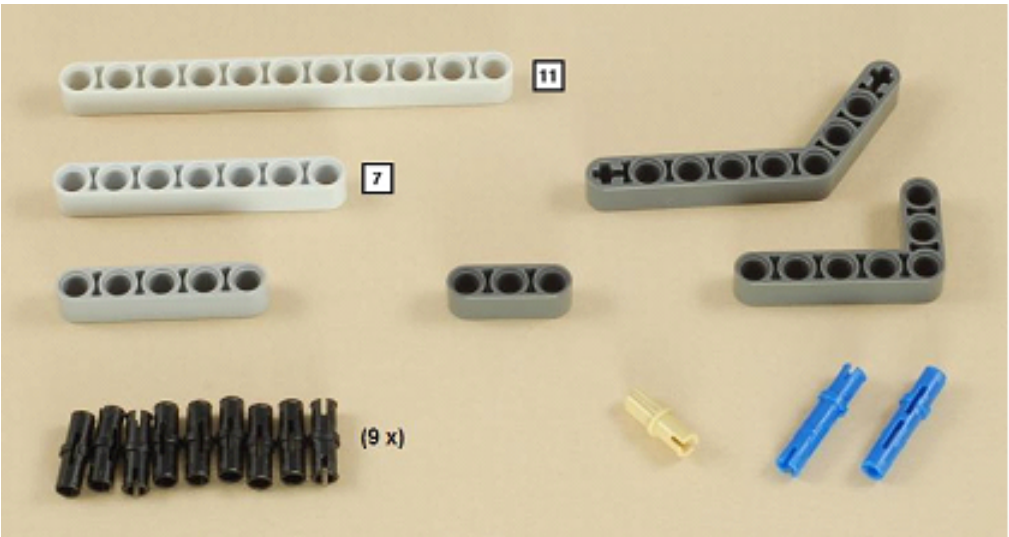


Figura B.30: Paso 29

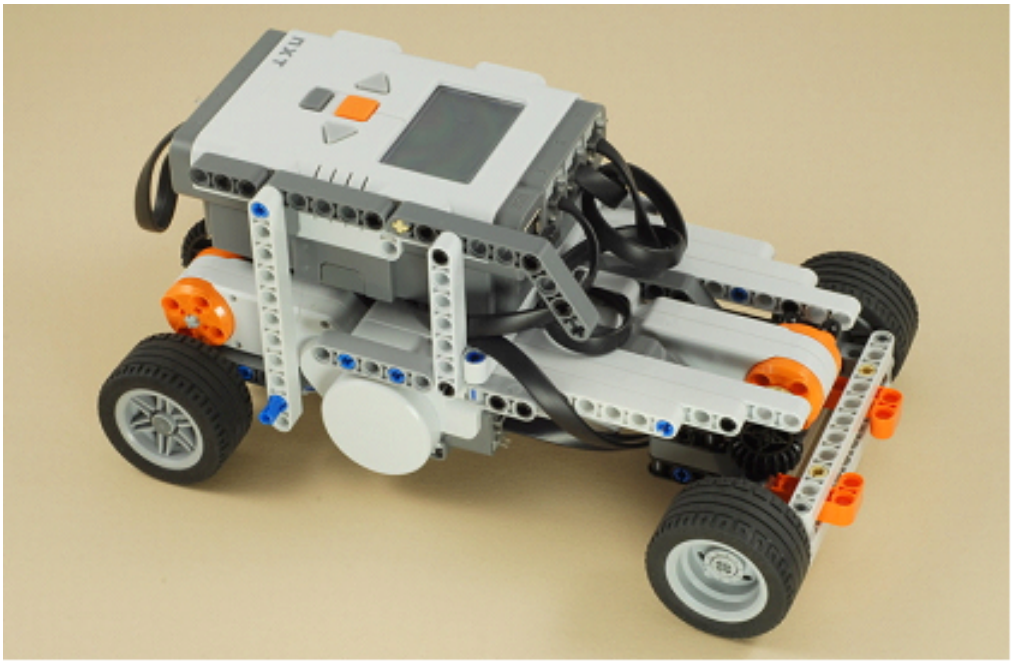


Figura B.31: Paso 30

Fase 16

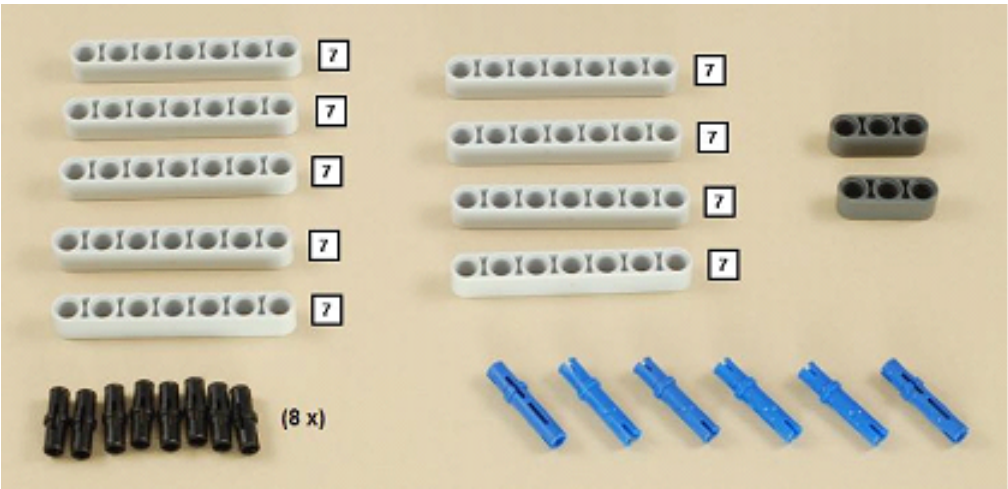


Figura B.32: Paso 31

Figura B.33: Paso 32

Fase 17

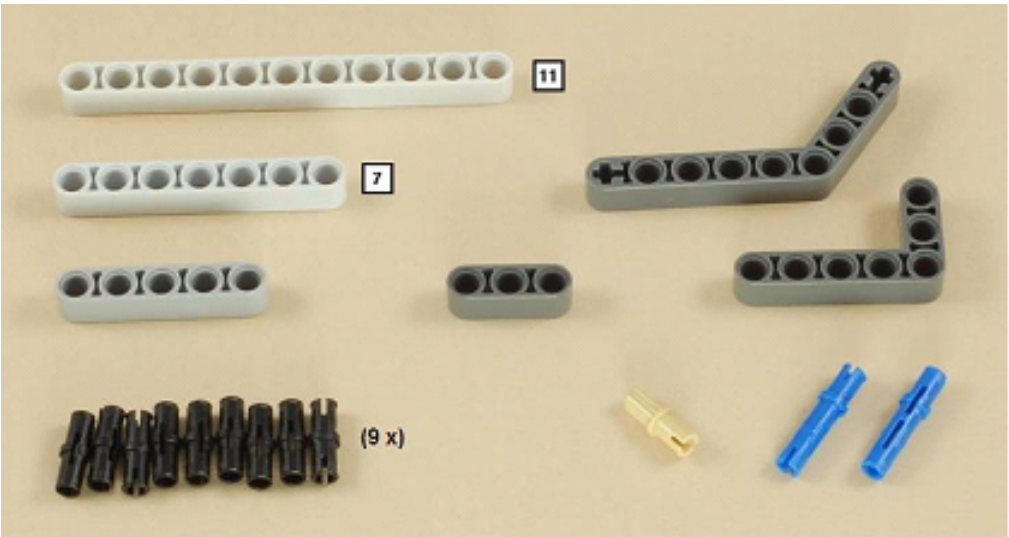


Figura B.34: Paso 33

Figura B.35: Paso 34

Fase 18

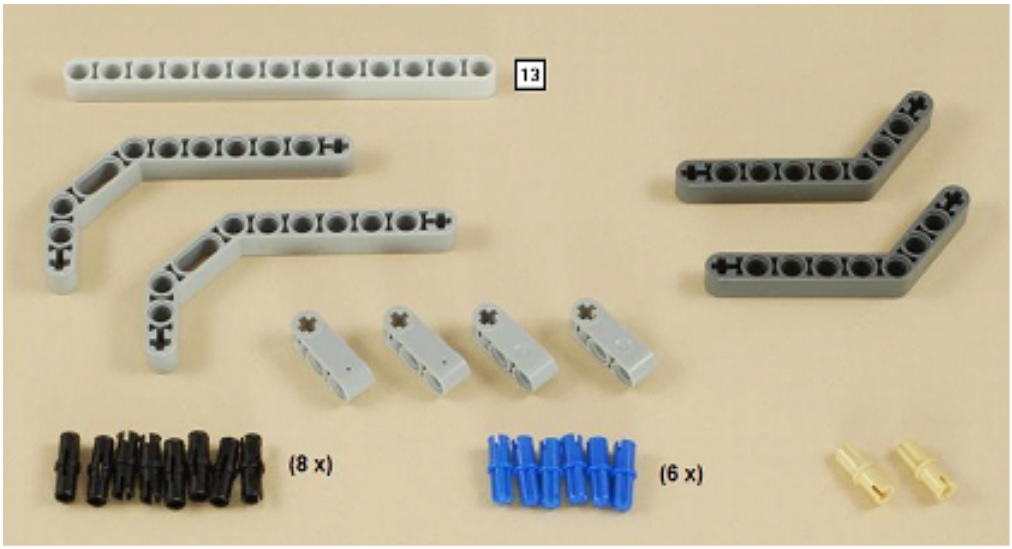


Figura B.36: Paso 35

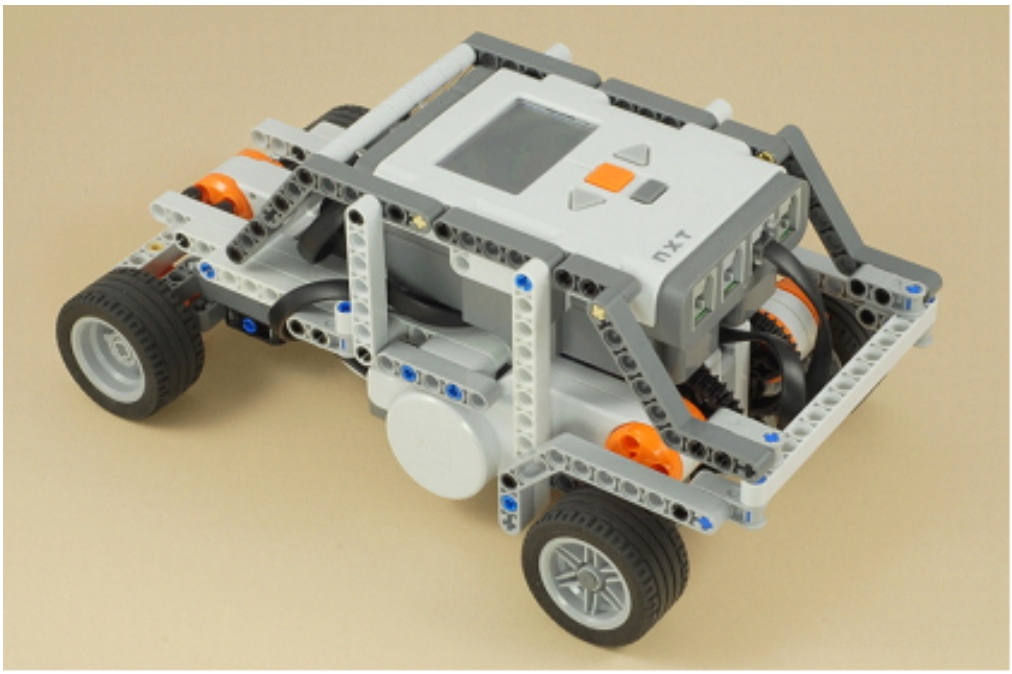


Figura B.37: Paso 36

Fase 19

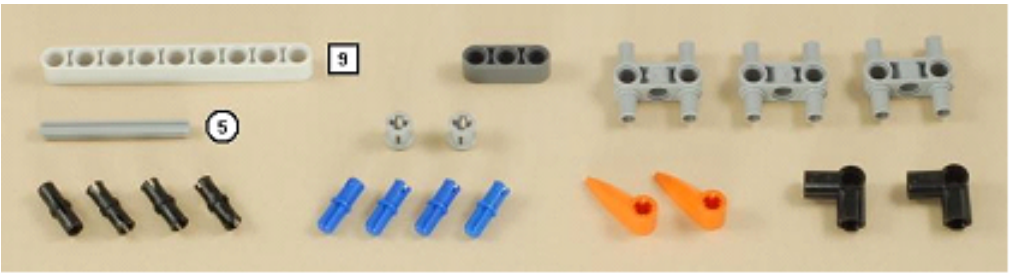


Figura B.38: Paso 37

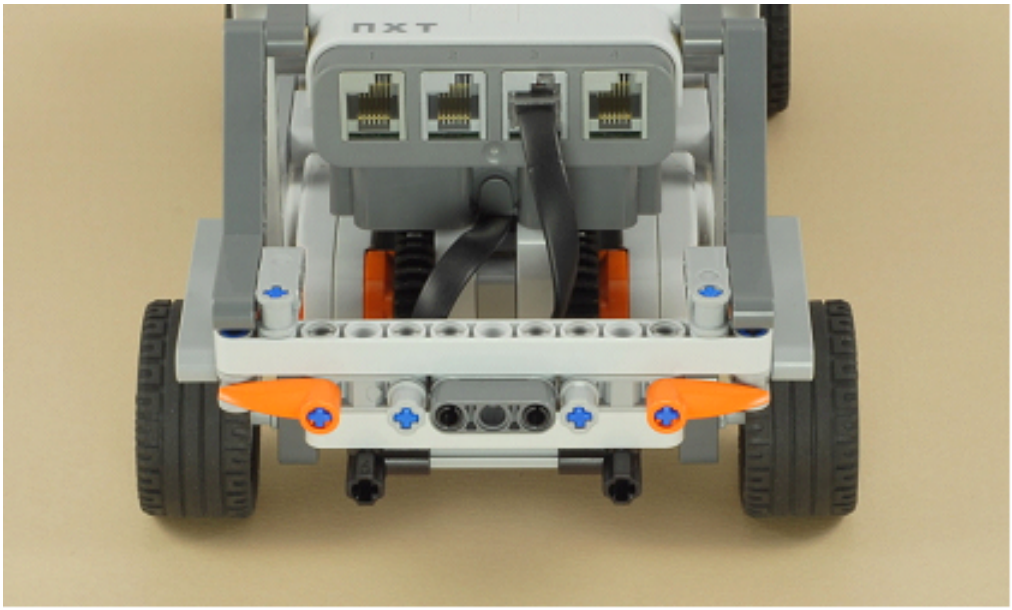


Figura B.39: Paso 38

Apéndice C

Manual de usuario

Este capítulo recoge una guía completa de utilización del software implementado. Junto con esta memoria se adjunta el código fuente, que deberá ser compilado para poder ser utilizado.

C.1. Equipo cliente

C.1.1. Instalación y ejecución

Para poder ejecutar el código será necesario estar bajo el sistema operativo *Linux* (probado en *Ubuntu 10.04*) y tener instalado *Qt*, el kit de desarrollo de aplicaciones de *Nokia*. Es posible, también, crear un ejecutable[1] para arrancar directamente desde consola. Si procedemos a través del *SDK* de Nokia, basatará con abrir el fichero principal del proyecto y pulsar el botón de "play" para que comience la aplicación.

El usuario debe tener instaladas en el sistema las siguientes librerías:

- **libfreenect** driver del *Kinect*, descargable desde el sitio web de OpenKinect[2]
- **GLU** "OpenGL Utility library". Librería OpenSource para OpenGL.
- **GL** Librería OpenGL Oficial.
- **GLUT** OpenGL Utility Toolkit, un conjunto independiente de herramientas para para escribir programas OpenGL.
- **ICE** ZeroC's Internet Communications Engine.
- **X11** X Window System versión 11.

[1]Tutorial: `http://doc.trolltech.com/4.6/qt-embedded-deployment.html`
[2]OpenKinect: `https://github.com/OpenKinect/libfreenect`

- **libxext** Sublibrería de de X11.
- **libxmu** librería para programar en el sistema de ventanas X.
- **libxi** librería cliente para ficheros de desarrollo XInput.
- **lpthread** librería que implementa el estándar POSIX para la manipulación de “hilos”.
- **libm** Para el desarrollo y utilización de bibliotecas de matemáticas.
- **libcv** Librería OpenCV.
- **libcvaux** Extensiones de OpenCV.
- **librt** Librería de propósito general (listas, tablas hash, sockets, etc.)
- **libcxcore** Para compilar aplicaciones basadas en OpenCV.
- **libhighgui** Para compilar aplicaciones que utilizan el API para construir interfaces gráficas de OpenCV.
- **libjpeg** Librería para codificar y descodificar ficheros JPEG y otras utilidades JPEG.
- **libml** Machine Learning Library. Implementa algoritmos de aprendizaje automático.
- **libbluetooth** Archivos de desarrollo para usar la librería *Bluetooth* para *Linux*, *BlueZ*.

En las siguientes secciones se explicarán las distintas opciones del programa. La aplicación consta de una única pantalla principal, cuyo centro está ocupado por un área grande, que alojará la vista principal de las manos y el volante. En la esquina inferior izquierda, el usuario podrá ver una vista de la cámara a color, a modo de referencia.

Vista inicial

Inicialmente, la zona central estará ocupada por la información acerca de la distancia a la que se encuentra el usuario. El botón de arriba a la derecha, con el icono de la carita, permite calibrar la distancia a partir de la cual filtrará *Kinect*. Tras pulsarlo, el usuario dispone de 10 segundos para situarse frente al dispositivo, con los brazos extendidos a la distancia que él desee, pero se recomienda que sea entre 85 y 105 cm y que el *Kinect* esté a una altura de entre 1.2 y 1.5 metros.

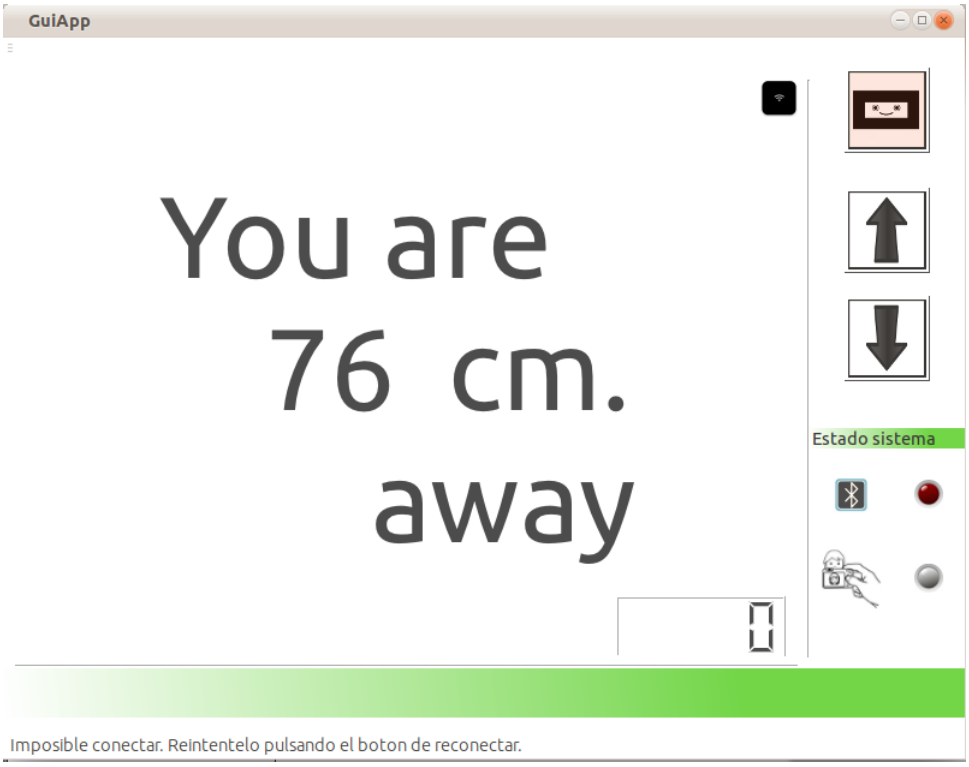


Figura C.1: Vista inicial

Podemos ver otros elementos que compartirá también con la siguiente vista:

- **Botón de reconexión**: La aplicación intentará siempre conectarse automáticamente al vehículo nada más iniciarse, pero en caso de que el coche esté apagado o hayamos olvidado iniciar su programa, aparecerá un botón negro pequeño en la esquina superior derecha que permitirá reconectar.

- **Barra de información**: Informará al usuario del éxito de las operaciones realizadas y de errores puntuales, como por ejemplo, si se ha producido un error en la conexión. Durante la manipulación del vehículo mostrará los datos enviados.

- **Botones "flecha"**: Las flechas permiten al usuario reposicionar el ángulo de inclinación vertical del *Kinect*. Cada vez que lo pulse se desplazará en un grado.

- **Indicadores "LED"**: Se representan como dos pequeñas lámparas LED y están situadas abajo a la derecha. Informan al usuario mediante un sencillo código de colores de si el *bluetooth* o la calibración se han realizado correctamente. El rojo indica error, el verde éxito y el gris, no inicializado.

La aplicación intentará siempre conectarse automáticamente al vehículo nada más iniciarse, pero en caso de que el coche esté apagado o hayamos olvidado iniciar su programa, aparecerá un botón negro pequeño que permitirá reconectar.

Vista de ejecución

En la vista de ejecución, el usuario ya ha pulsado el botón de calibrado y el texto del panel central desaparece dando lugar a la imagen con las manos y el volante. Es decir, el usuario al extender sus brazos verá solamente las manos y el sistema determinará sus contornos y el centro de cada uno. Para la mano izquierda, el sistema calculará cuántos dedos están abiertos.

Aparecen dos elementos nuevos:

- **Contador de marchas**: Cuenta el número de dedos abiertos en la mano izquierda y muestra un signo negativo o positivo en función de si la marcha es positiva o negativa, respectivamente.
- **Imagen a color**: Muestra la imagen de la cámara a color para dar una referencia al usuario de su posición y como puro elemento decorativo.

Controles

El control del vehículo es sencillo, ya que sólo depende de las manos. Una vez realizada la calibración y que el usuario esté viendo sus manos segmentadas en la imagen podrá realizar las siguientes acciones:

- **Cambios de marcha**: Podrá imprimir mayor o menor velocidad al vehículo indicando qué marcha desea con los dedos de la mano. El rango oscial de 1 a 5 en el sentido positivo y de -1 a -5 en el negativo. 0 significa que el coche está parado. Por ejemplo, si desea meter la quinta, deberá abrir la mano completa, si desea una segunda, mostrar sólo dos dedos, de forma parecida a la siguiente figura (fig. C.2).

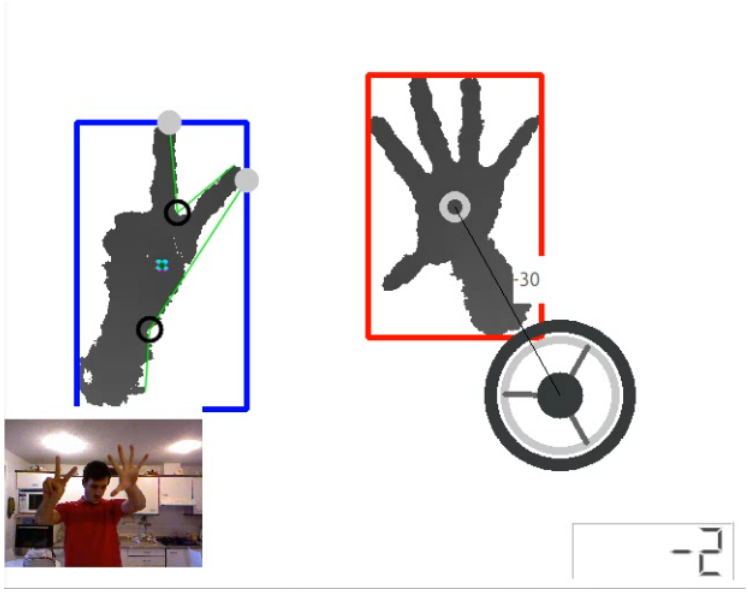


Figura C.2: Indicador de la marcha

- **Sentido de la marcha**: Para cambiar el sentido de la marcha, de marcha adelante o marcha atrás y viceversa, basta con abrir la mano derecha una vez y volver a cerrarla. El gesto es sencillo y no requiere un reflejo especial. En la figura anterior (fig. C.2) se puede ver cómo cambia la marcha y el contador muestra un signo negativo de marcha atrás.

- **Giro**: El giro se realiza con la mano derecha, describiendo un arco sobre el volante dibujado en la pantalla, en la zona central a la derecha. Se considera como ángulo cero cuando el segmento que forma la mano con el centro del volante es perpendicular al eje horizontal OX. El ángulo irá decreciendo hacia la izquierda hasta un tope de -60○ y un límite de +60○ por la derecha. En la figura anterior (fig. C.2) la mano está indicando a la transmisión del vehículo que se sitúe en un ángulo de -30○ con respecto de la vertical.

Aquí finaliza la explicación de la aplicación cliente en el ordenador, que es la que utilizará el usuario. Para finalizar el programa basta con que pulse sobre la“x” típica de “cerrar ventana”.

C.2. Aplicación NXT

La aplicación del vehículo es muy sencilla de ejecutar, pero algo más complicada de instalar y configurar[3].

C.2.1. Instalación

En primer lugar, habrá que asegurarse de tener instaladas las siguientes librerías (partimos de la base de que estamos en un sistema Debian):

- libusb
- libusb-dev
- libbluetooth-dev
- ant
- gcj.

[3]Guía de configuración del BT: `http://nxt.ivorycity.com/index.php?/archives/3-How-to-get-started-with-Linux-Bluetooth-and-the-NXT.html`

A continuación, será necesario descargar el programa *NXJ*[4] en nuestro ordenador. Seguidamente, en una consola de comandos escribiremos:

- Extraemos el contenido: *tar -xvf lejos_NXJ_0_8_5beta.tar.gz*
- Movemos el contenido del archivo: *mv lejos_NXJ_0_8_5 lejos*
- Movemos la carpeta a otra dirección, por ejemplo: *mv lejos /opt/*
- Añadimos las variables de entorno de las nuevas librerías, para ello navegamos hasta /etc./profile.d/ y creamos un script llamado *nxt.sh*, con el siguiente contenido:
  - *export NXJ_HOME="/opt/lejos"*
  - *export PATH=*$PATH$*:NXJ_HOME/bin*
  - *export LD_LIBRARY_PATH=*$LD\_LIBRARY\_PATH$*:NXJ_HOME/bin*
- No olvidemos darle permisos de ejecución: *sudo chmod u+x nxt.sh*

El paso siguiente es compilar la librería:

cd /opt/lejos/build
ant

Creamos un grupo de usuarios llamado "lego", los cuales pueden cargar programas al dispositivo:

sudo addgroup lego

Nos agregamos al grupo "lego":

sudo usermod -a lego tuusuario

Creamos el siguiente archivo.

sudo gedit /etc/udev/rules.d/70-lego.rules

Y escribimos en él lo siguiente:

 # Lego NXT
BUS=="usb", SYSFSidVendor=="03eb", GROUP="lego", MODE="0660"
BUS=="usb", SYSFSidVendor=="0694", GROUP="lego", MODE="0660"

Guardamos el archivo y ya estamos listos para programar nuestro lego. Para compilar un fichero **.java* basta con situarse en el directorio donde está y escribir (ponemos la clase principal y el compilador resuelve automáticamente todas las dependencias):

[4]Descarga de NXJ LeJOS: `http://sourceforge.net/projects/lejos/files/lejos-NXJ/0.8.5beta/lejos\_NXJ\_0\_8\_5beta.tar.gz/download`

nxjc Clase.java

El compilador habrá creado un código intermedio que es el que interpretará la máquina virtual de Java del vehículo (*.*nxj*). El último paso es cargar el programa en el controlador, para ello (con el "brick" encendido:

nxj Clase

Una vez transferido el fichero, para ejecutarlo, bastará con ir a la opción Files→Clase.nxj (Clase.nxj o cualquiera que sea el nombre de la clase principal que da nombre al programa) y pulsamos el botón naranja. Si está marcado como la opción por defecto, pulsaremos directamente sobre la opción "Run Default" del menú principal.

Figura C.3: Menú principal

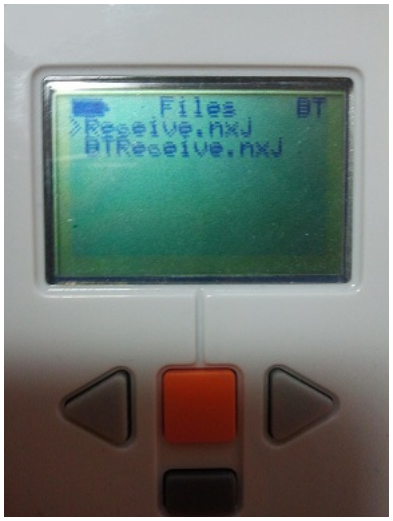


Figura C.4: Menú de aplicaciones instaladas en el controlador

Apéndice D

Gestión económica del proyecto

Este capítulo completa el último de los apéndices propuestos para esta memoria y constituye un aspecto importante a tener en cuenta. En todo proyecto debe considerarse el coste económico del personal utilizado, los materiales y dispositivos y otros gastos posibles. En definitiva, se trata de controlar el gasto final. Para la realización de esta labor, generalmente se utilizan herramientas específicas, pero dadas las características de este proyecto con una estimación es suficiente.

No se ha hecho una planificación como tal, diviendo el proceso en etapas y tareas a cumplir en determinados plazos, pero sí se ha llevado un conteo de los días y las horas dedicadas, dejando libre la distribución del tiempo a la elección del autor. Tampoco se han asignado roles, pues es una única persona quien los concentra en sí misma.

En este apartado se describen los presupuestos asignados al inicio de este proyecto, diferenciando cada uno de los aspectos presupuestaddos.

D.1. Recursos Hardware/Software empleados

Este es el listado tecnológico utilizado en el proyecto:

- **Hardware**
  - Toshiba Satellite con un Intel ® Core™ i5, CPU M430 a 2.27GHz y 4Gbytes de RAM.
  - Un Kinect™.
  - Un set Lego® Mindstorms® NXT 1.0
  - Dispositivo genérico de radio Bluetooth. Fabricante: Cambridge Silicon Radio Ltd.
- **Software**

- Sistema operativo: Ubuntu 10.04 con Gnome 2.30 + Unity. Kernel 2.6.32.
- IDE de desarrollo: *Nokia Qt Creator*. Por su simplicidad y facilidad de uso.
- Lenguajes de programación: C++ para la aplicación cliente en el ordenador, por su potencia y velocidad, indispensables para el procesamiento fluido de las imágenes, seguimiento de objetos y otros cálculos. Java, para el software de la aplicación servidor, instalada en el vehículo.
- SDK de desarrollo: *libfreenect* de *OpenKinect*, por que fue el primero en salir y al no ofrecer tanta funcionalidad extra salvo la básica, permite sumergirnos más en la comprensión de la tecnología *Kinect*.
- Control de versiones: se ha utilizado el sistema de control de versiones *GIT*[1], en concreto a través de la red social *github*[2]. El proyecto puede descargarse de la página creada para tal uso [3]
- Editor de textos para la memoria: se ha utilizado *Latex* y se ha redactado bajo el sistema operativo Windows 7 de *Microsoft*, utilizando el editor de textos *Latex*, *TeXworks*[4]. *Latex* es un sistema de composición de textos mediante una lenguaje de etiquetas y macros, con el consiguiente esfuerzo que esto supone, pues es una herramienta que, aunque produce resultados muy satisfactorios, también genera dolores de cabeza en la misma medida.
- Documentación del código: Doxygen, potente herramienta para documentar código, no importa el lenguaje que se utilice y permite generar documentación en HTML y generar diagramas de clases, de secuencia, de actividad, etc. de forma automática.

D.2. Recursos humanos

Respecto a los recursos destinados a los trabajadores, primero hay que identificar el personal que va a ser necesario para realizar el proyecto. En este caso, sólo una persona, José Manuel Cabrera Canal, el autor de este proyecto, es la encargada de desarrollar todo el proyecto, por lo que asume todos los roles del proyecto: jefe de proyecto, diseñador, analista y programador.

El coste asociado al salario del personal de trabajo, que en esta caso corresponde únicamente al de una persona, se obtiene a partir de una estimación del número de horas que va trabajar en el proyecto. Esta estimación se basa en anotaciones en un cuaderno de bitácora tomadas por el autor, apuntando los días y las horas dedicadas, teniendo en cuenta que cada

[1] git: `git-scm.com`
[2] github: `https://github.com`
[3] GuiPFC, mi PFC: `https://github.com/elelement/GuiPFC`
[4] TeXworks: `www.tug.org/texworks`

día de trabajo supone una media de cinco horas.

Días trabajados	Horas/día	Coste por hora	Coste total en euros
197 días	5h/día	20/hora	19700

Cuadro D.1: Desglose del gasto en RRHH

Además de los gastos en recursos humanos, el hardware utilizado ha tenido también un coste, que se detalla a continuación (los recursos hardware listados anteriormente y que no aparecen en la lista eran capital que ya poseía el autor.):

Concepto	Coste
Kinect	84
“Radio bluetooth” de “Cambridge Silicon Radio Ltd”	220,29
Coste total	**304.29**

Cuadro D.2: Desglose del gasto de hardware