Universidad Carlos III de Madrid

Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

HAPTIC TELEOPERATION OF THE YOUBOT WITH FRICTION COMPENSATION FOR THE BASE

Autor: Miguel Corberán Ruiz

Tutor: María Dolores Blanco Rojas

Leganés, Septiembre de 2012

Abstract

Haptic devices are bringing new possibilities for teleoperation by increasing the level of awareness that the operator can have over the slave. In other words, they create a stronger link between them. Because it is not enough to have a view of the task at hand, it is better to feel what is really happening at the other side.

The main goal of the project is to provide the KUKA youBot with an Omega 6 haptic interface. The operator can feel the movement limitations that the arm's tooltip may be experiencing, resulting in a better driving practice. But with these new capabilities other concerns arise, like the choice of an appropriate control algorithm, the correct coupling of workspaces or the design of a suitable data handling scheme.

However, the current setup has not yet been submitted to a proper system validation and so there is still much work to do in order to increase its overall performance.

Since friction can have a major role in the control scheme of the system, the latter should be provided with friction compensation. To achieve this, a study of the youBot wheels motor block friction has been carried out. These results are then also incorporated in the robot simulation.

Moreover, when identifying this kind of behaviours some important decisions have to be made in order to get the best results from the time invested. Among those are the selection of a friction model, the system identification experiments and the validation of results.

In conclusion, it has been proven that the implementation of a haptic interface for the youBot is not only feasible but that it delivers a greater overall teleoperation experience. Also, although the results of this project are an initial version of the system, the friction compensation for the base motor blocks is already working with acceptable performance.

Keywords: Haptics, friction modelling, youBot, robotics, mechatronics.

Table of Contents

1 Introduction

The present report attempts to document the progress on the work that has been carried out at the Robotics and Mechatronics (RAM) lab of the University of Twente. This project represents the final work of my studies and with it the closure of an important period in my life. It is because of this that I wanted to work on something of my liking from what I could also gain some experience in the robotics field.

At the University of Twente I found a good working environment and the opportunity to show and enjoy my involvement with mechatronics. Since I have mechanical background I wanted to work on a physical setup in which I could test my developments. After some initial research this project was assigned to me and my work begun.

1.1 Motivation

The facilities at the aforementioned lab are well suited and many interesting projects are taking place concurrently. The acquisition of a new development platform, the mobile robot youBot, had provided a solid setup for new lines of study.

In previous projects to this work a simulation model had been made which could be used to control the youBot. Taking advantage of this model and using an already built control scheme, it was decided to work on the development of an interface to make possible the use of a haptic device with the youBot.

Haptic devices uses are rapidly growing but these are still used majorly in classic teleoperation systems, usually in industrial telemanipulators. This project attempts to control a multiple degree of freedom arm mounted on a mobile base with a haptic device.

Until now mobile robots have been controlled with standard remote control transmitters. These remotes limit the manoeuvrability of the controlled apparatus and require a big deal of concentration and skills so as to execute the correct button and levers combinations. On the contrary, haptic devices are known for their ease of use, appealing to the operator’s intuition. The coupling of two pieces of technology like these is not usual and so it could be of interest to see what could the project develop into.

The involving elements require the application of control engineering knowledge on a real mechanical setup through modern software developing tools. This blend of disciplines suffices to tag this project as a robotics project, which is the kind of work that I wanted to do.

1.2 Main goals

The primary goal will be to model and develop a haptic controller for the KUKA youBot. The project will be mainly focused on the control part of the design since a real hardware setup is already available. It should be noticed that the intended result at the human end is to have a feeling of the movement limitations that the robot's end effector may be experiencing rather than an exact sense of reality.

Prior to all, it will be necessary to identify the different parts involved in the teleoperation concept: both master and slave together with the controller itself form the teleoperator block which allows the human operator to interact with the environment from the distance. The robot (slave) and the haptic device (master) models are also available from previous projects and research. However, it will also be needed to improve the robot model in order to get a more realistic dynamic behavior. This includes the improvement of the motors models and also of the friction force at the wheel blocks.

If a fairly good model is obtained, it will also be used for compensating the friction effect on the wheels and therefore, to improve youBot's drivability through a better control scheme.

Basically, the haptic device will be broadcasting the position at which the robot's end effector should move. At the same time it will also receive and reproduce the force that is being exerted on the robot's end tip. Here it should be addressed, for example, the matching between the motion ranges of both master and slave which could limit the overall functionality.

While modeling it is very important to decide whether these parts have an overall active or passive role in the system. This is close related to the system's stability which is a very important aspect of the development.

After this, research of the many available control algorithms will be carried out and together with the necessary modifications a solution will be applied. For this concepts such as bond-graphs or applications like 20sim and the C++ development tool Orocos –together with ROS– will be used.

It will also be discussed which performance parameters should be measured when simulating/testing the proposed solution. Perfect transparency has always been the objective of this type of projects but it should always be taken into account what is the actual application requirements and to design and model according to that.

1.3 Mains used

The multidisciplinary facets of this project translate into the need for utilizing various different tools. Basically, in one hand there is the mechanical setup and in the other the software used to analyse, control and drive it.

For the current project two main hardware platforms will be used. These are the Omega6 haptic device from Force Dimension and the KUKA youBot.

Omega6

The omega6 haptic device has 6 degrees of freedom although only half of them are reciprocal, that is, only the translational ones are actuated. The rotations can be sensed and transmitted but they are not actively driven. Figure 1 shows a picture of the device:

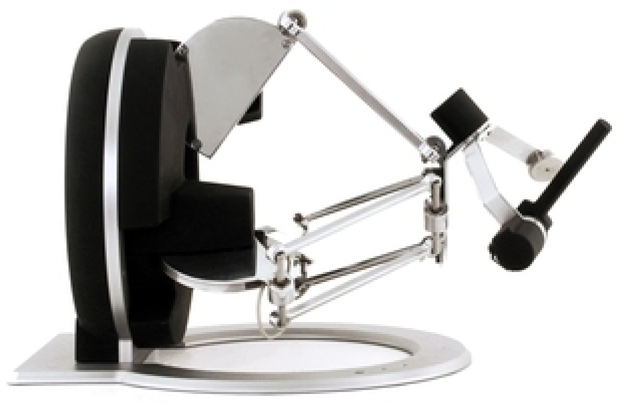


Figure 1: Omega 6 haptic device.

It is constructed with a delta-based parallel kinematics structure that provides good closed loop stiffness and high accuracy. The pen-shaped end-effector allows the user to achieve the wanted orientation quite intuitively. This pen is also constructed in such a way that the translations and rotations are decoupled from each other, opening the device to further uses.

It can be easily calibrated in a single movement and, because of its superior build quality, this can be done just one time at the beginning. This device is also equipped with active gravity compensation which greatly improves the teleoperator feeling and at the same time reduces its fatigue.

It is equipped with two buttons, one for starting the calibration procedure and the other for setting on or off the display of forces. Table 1 shows some of the most important specifications:

workspace	translation rotation	∅ 160 x 110 mm 240 x 140 x 320 deg
forces	translation	12.0 N
resolution	translation rotation	< 0.01 mm 0.09 deg
stiffness	closed-loop	14.5 N/mm
dimensions	height width depth	270 mm 300 mm 350 mm
interface	standard refresh rate	USB 2.0 up to 8 KHz
power	universal	110V - 240V

Table 1: Omega6 specifications.

youBot

The KUKA youBot is a platform that targets the gap between research & development and education. It is a mobile manipulator composed by a wheeled base and a robotic arm mounted on top of it. A good range of mobility is achieved using the four mecanum wheels. The base coordinate frame can be seen in the next picture (figure 2), as well as the wheels numbering.

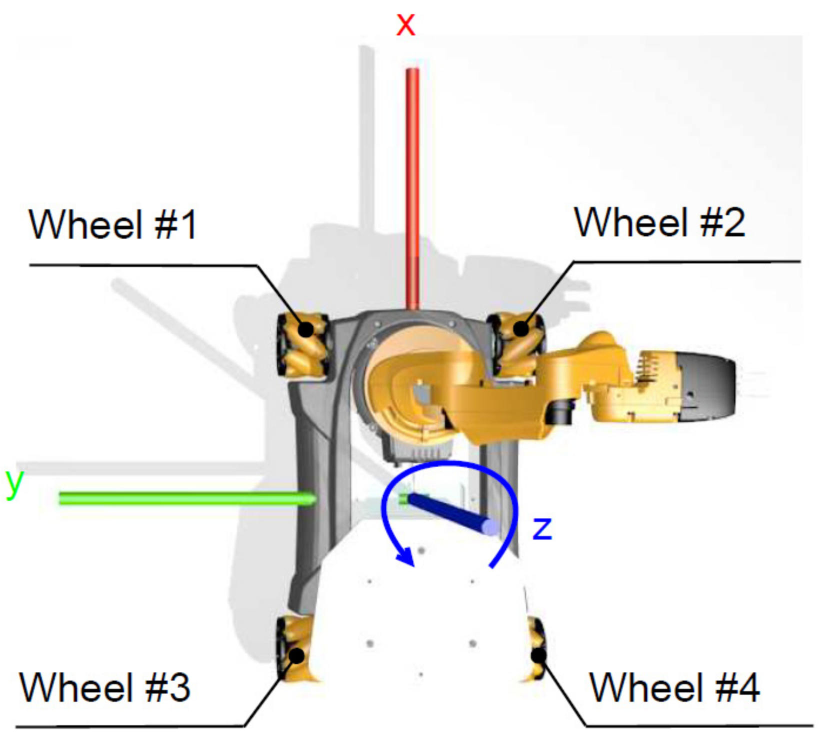


Figure 2: youBot base coordinates frame and wheels numbering.

The coordinate frame origin is located at the centre of odometry. If all the wheels turn in the positive direction the robot chassis will turn as specified by the blue arrow seen above.

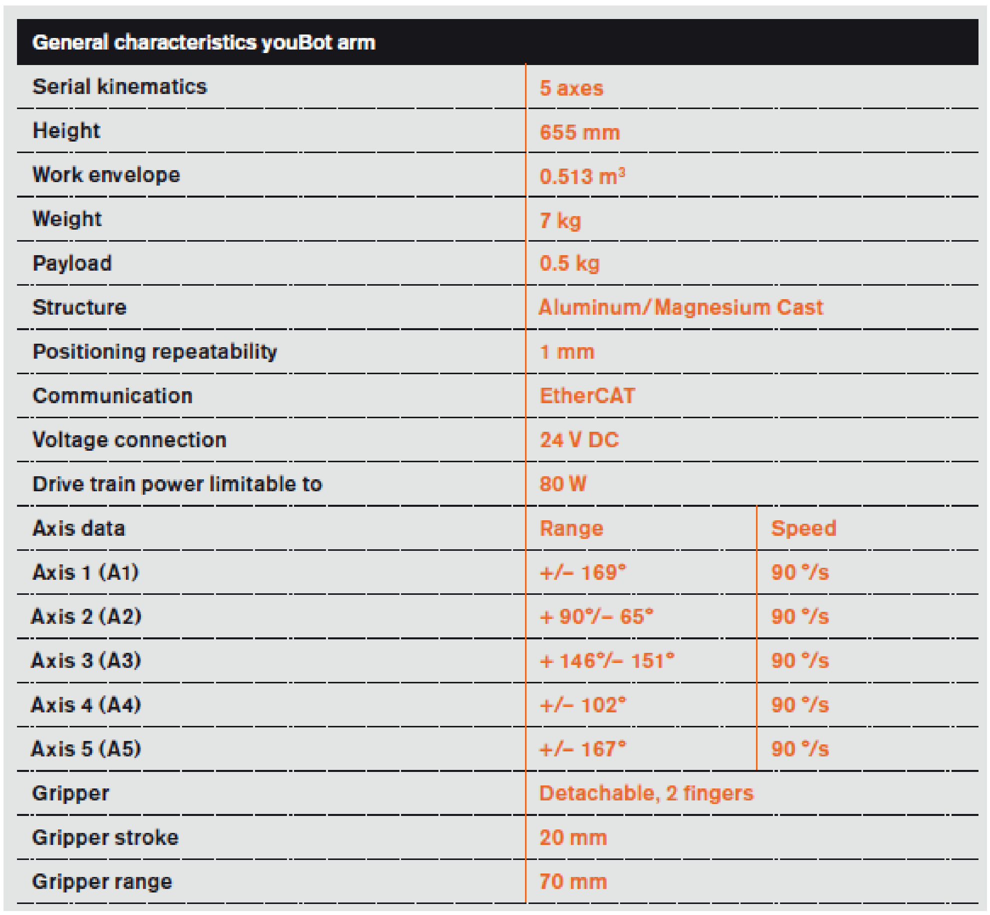
The youBot can be accessed via Ethernet or Ethercat depending on whether the robot is connected to an external computer or to the on board PC. This is valid for the base and the arm separately.

The internal computer is based on a Mini-ITX board with an Intel Atom processor.

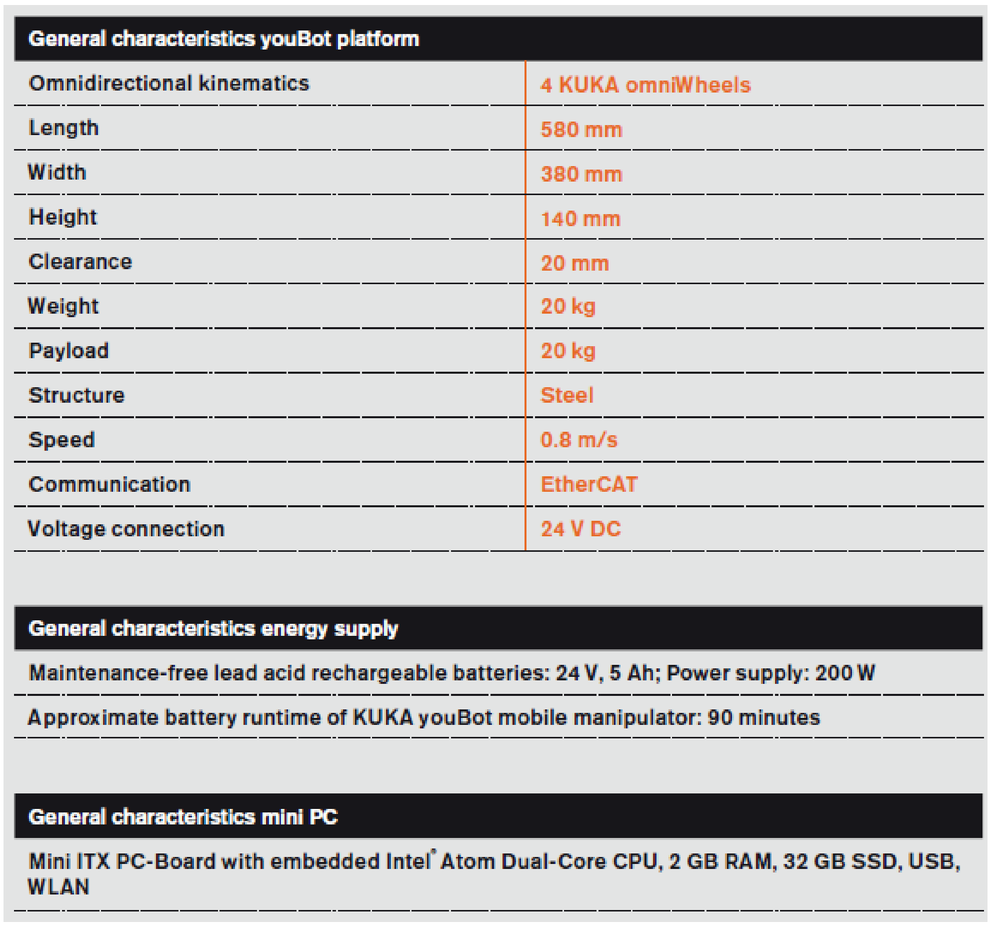
There is as well a 24V power plug that can be used to power the youBot and recharge the battery at the same time. A LCD screen situated at the top indicates the robot's power status as well as the battery capacity. Additionally, I/O devices can be connected through USB ports as well as a monitor using a VGA connection.

The arm is based on a serial chain kinematic structure with five revolute axes. The tooltip is a two-finger gripper that can be removed. As with the base, the motor drivers can be accessed individually providing the ability of targeting each joint separately.

In the next tables can be observed the principal characteristics of the youBot's base, arm, onboard PC and power supply.



General characteristics youBot arm		
Serial kinematics	5 axes	
Height	655 mm	
Work envelope	0.513 m^3	
Weight	7 kg	
Payload	0.5 kg	
Structure	Aluminum/Magnesium Cast	
Positioning repeatability	1 mm	
Communication	EtherCAT	
Voltage connection	24 V DC	
Drive train power limitable to	80 W	
Axis data	Range	Speed
Axis 1 (A1)	+/– 169°	90 °/s
Axis 2 (A2)	+ 90°/– 65°	90 °/s
Axis 3 (A3)	+ 146°/– 151°	90 °/s
Axis 4 (A4)	+/– 102°	90 °/s
Axis 5 (A5)	+/– 167°	90 °/s
Gripper	Detachable, 2 fingers	
Gripper stroke	20 mm	
Gripper range	70 mm	



General characteristics youBot platform	
Omnidirectional kinematics	4 KUKA omniWheels
Length	580 mm
Width	380 mm
Height	140 mm
Clearance	20 mm
Weight	20 kg
Payload	20 kg
Structure	Steel
Speed	0.8 m/s
Communication	EtherCAT
Voltage connection	24 V DC

General characteristics energy supply
Maintenance-free lead acid rechargeable batteries: 24 V, 5 Ah; Power supply: 200 W
Approximate battery runtime of KUKA youBot mobile manipulator: 90 minutes

General characteristics mini PC
Mini ITX PC-Board with embedded Intel® Atom Dual-Core CPU, 2 GB RAM, 32 GB SSD, USB, WLAN

Table 2: youBot specifications.

More specifically, the motors used for the base and the first three joints of the arm (1, 2 and 3) are the 50W brushless Maxon EC45 flat (model number 251601). The fourth joint of the arm uses a 30W brushless Maxon EC45 flat motor (339281) and the fifth joint a 15W brushless Maxon EC32 flat motor (267121). The gear block used in the base motors, which are the ones that have been under study, is a Maxon Spur Gearhead GS45 (301173).

Each joint has its own motor controller containing an ARM Cortex-M3 microcontroller, Hall sensors, an EtherCAT interface and position/velocity/current PID-controllers. It is known that the encoders of all joints have 4000 ticks per revolution and are of the relative type.

Software

The main software developing tools have been 20Sim, Eclipse and Matlab. 20Sim has been used for developing the components that work along the youBot control block. It has also been used for simulations of the complete system as well as for designing the experiments carried out in the present work. This program allows the usage of different development syntaxes side by side, which results in a great versatility. Some of these languages are bond graphs or block diagrams.

Eclipse has been used to create the C++ based components that form the interface between the haptic device and the youBot. Specifically, the kinds of applications that have been built are ROS (Robot Operating System) nodes and Orocos (Open Robot Control Software) components. These are two open source multiplatform frameworks that can be used to create robot applications. Both provide device drivers, libraries, etc. to make easier this task. However, in this project ROS has been used as the global framework because of its powerful simplicity and Orocos has been used for the hard real-time work. There are some packages that allow for them to collaborate in order to bring the best from both of them together.

The program Matlab has been used mostly for the friction modelling part of this work. Data has been imported, pre-processed and then analysed using the various tools it offers. It has delivered as well mains for the identification of the models' parameters.

20Sim and Matlab have been working on Windows 7 stations while Eclipse has been ran in the Linux environment Ubuntu 11.10 (together with ROS and Orocos).

1.4 Project organization

Although the main objective of this work is to develop a haptic interface for the youBot, an important amount of time has been invested in creating a friction model for the wheels motor blocks. This was done for various reasons: to improve the youBot simulation model, to have a friction compensation component that could help on the control of the robot, and because of the interesting results that could be obtained. Moreover, all of these would help to achieve the set goal. It is because of this that the project is divided into two main parts: friction modelling and haptic teleoperator.

The friction modelling part has a brief introduction followed by the most characteristic features of friction phenomena. Next, a short view to the most important models and their evolution is given as well as some advice on model selection, where the initial model is chosen. After this the experimental campaign carried out on the youBot is explained and a model is selected according to the obtained results. This first part ends with the description of two of the applications for the developed model (being the most important the friction compensation for the base motor blocks) and some specific conclusions.

The second part pertains to the former haptic interface developed for the youBot. It starts as well with a short introduction to the subject and after it the most important aspects of haptic control and haptic design are resumed. Following are the initial problem specifications and the haptic interface developments' explanation. Next are also various conclusions on the matter.

Closing the report are the general conclusions of the project and some guidelines for future work based on the experience and progress met so far.

2 Friction modelling

2.1 Introduction

When dealing with robotics applications it is always desired to get the most from the available hardware. In order to achieve this it is important to have a good control scheme. Nowadays the development of a control system is based on the simulation model of the system itself. But if the simulation model is not realistic enough it will become quite hard to achieve it.

Despite that the simulation model of the youBot is already available it has some problems that need to be addressed. One of the biggest issues is the friction modelling of its joints.

The original youBot model assumes a constant friction force. This is far from reality where friction depends on the velocity among other operating conditions. This part of the master thesis targets to develop a friction model which can be used in all the operation regimes.

Friction has a great effect on all mechanical parts, especially in high precision applications, so it needs to be taken into account when dealing with this kind of systems. Friction modelling has always been of great interest for researchers during many years but it was not until the last decades when the friction phenomena could be imitated very closely.

During the latest years various theories have been proposed for friction modelling. Here some of them will be reviewed as an introduction to the subject and after this the implementation, application, and adjustment of the selected model will be explained.

2.2 Tribology

It's not strange that friction has had a lot of people occupied with it. It is a multiregime, highly nonlinear process that depends on many different variables. This has forced the necessity of shifting to a new view when new discoveries were made, resulting in a still evolving theory.

Friction force is composed mainly of two regimes, being these the presliding and the gross sliding (or sliding) regime. Latest research has seen the contact between two surfaces as the interaction of small asperities spread along them which fuse and break depending on the forces acting at each moment. The velocity at which these asperity junctions form and break is what separates one regime from the other [1].

Figure 3 shows the friction force vs. the displacement with forces applied at different rates (arbitrary units used). One can notice the transition from the presliding regime into the gross sliding regime. It is highly related with the breakaway force that is needed to get the object moving. Once this static value is reached the friction force decreases and remains more or less constant.

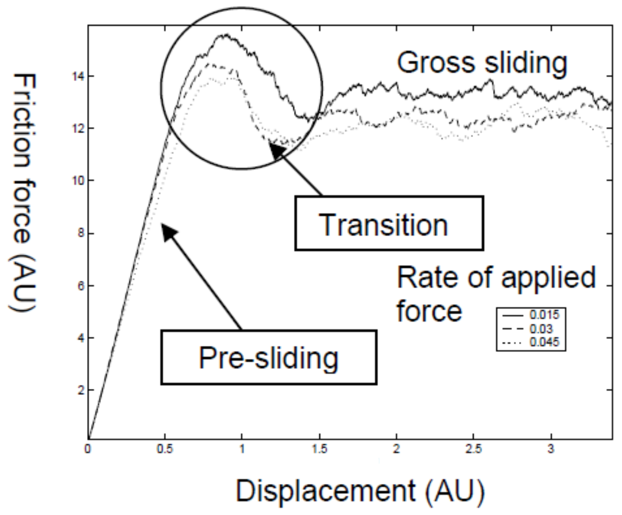


Figure 3: Pre-sliding and gross sliding regimes and the transition between them (from [1]).

In the following the main characteristics of these regimes will be discussed for non-lubricated surfaces. Afterwards, an example of friction behaviour in lubricated contacts will be explained and compared to a dry contact friction case.

2.2.1 Presliding

In the presliding regime the adhesive forces from the asperity contacts are dominant. These asperity junctions deform elasto-plastically before breaking, behaving as nonlinear hysteretic springs. The result from this "bristles" comportment is a hysteresis with nonlocal memory. Figure 4 shows a friction force vs. displacement plot together with the trajectory followed over time (top diagram). The motion –which starts at 0 and finishes at 5– describes a hysteresis in the friction force vs. displacement plot, characteristic of this regime. It can also be noticed how the path that comes from point 3 passes again through point 2 in its way to point 4, describing a local hysteresis.

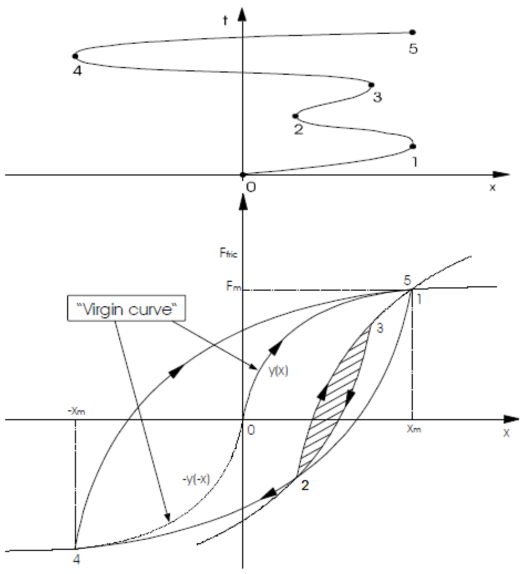


Figure 4: Global and local hysteresis force loops (from [1]).

Experience shows that experiments carried out at different velocities lead to the same results. Therefore, in the presliding regime the friction force is rate independent, that is to say, it depends on the displacement rather than in the velocity.

The theory states that when one asperity gets in contact with another from the other surface they stick together. As the relative displacement increases this junction deforms until it finally breaks and the loose ends release its elastic energy through internal hysteresis losses. Figure 5 shows a spring mechanism (left side) which reproduces the complete cycle, as can be seen in the spring-force vs. spring-extension diagram (right side).

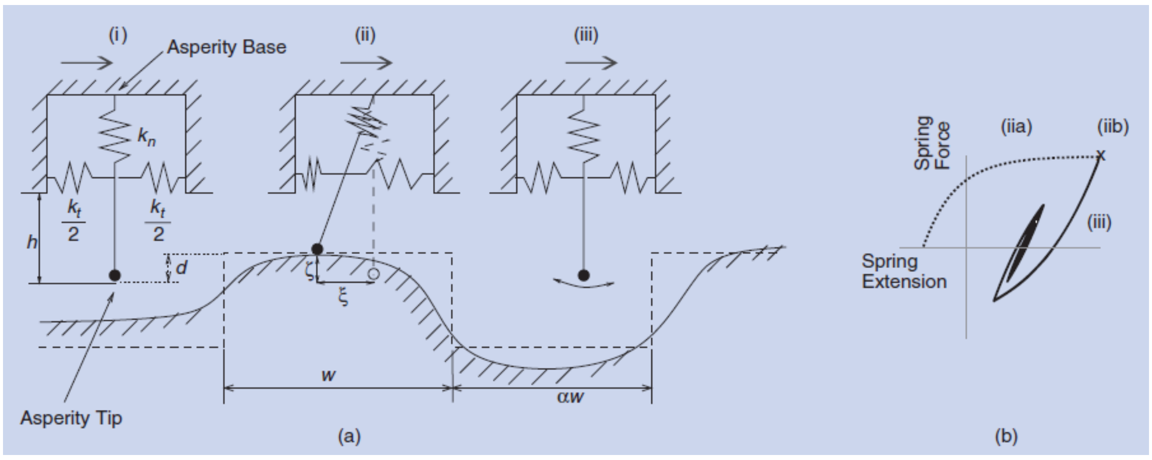


Figure 5: Asperity contact-deformation-breakage cycle (from [2]).

2.2.2 Gross sliding

As the velocity increases the asperity junctions start to break more rapidly and the gross sliding regime starts. The beginning of this regime coincides with the moment in which the break-away force is reached and the two surfaces in contact start sliding. Once this has happened, the friction force will first decrease before increasing again with increasing velocity (figure 6). These are called the velocity weakening and strengthening curves respectively. They together form what is known as the Stribeck curve, with the inflexion point at the Stribeck velocity. In the last part, the friction force is proportional to the velocity; this is the viscous friction region.

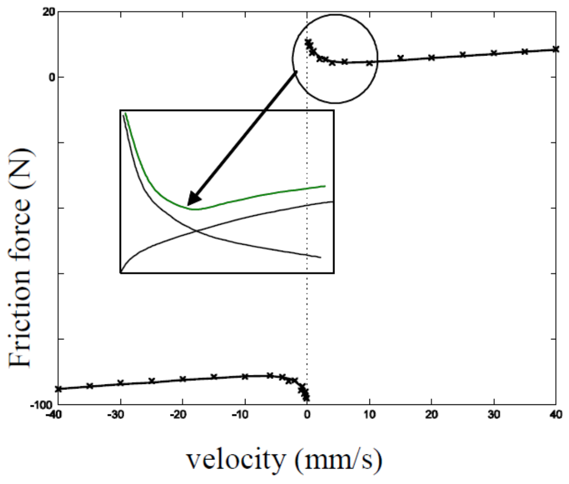


Figure 6: Friction force vs. velocity plot with a detail on the Stribeck curve (from [1]).

Another remarkable effect encountered in the sliding regime is the friction lag or friction memory phenomenon. It can be seen as a lag in the friction force with respect to the sliding velocity. In lubricated friction it is due to the time it takes for the lubricant film to change its thickness when the velocity changes –also known as the *squeeze* effect–. It can be expected the same for the dry sliding, although due to different causes. Mainly, as the force acting between two opposite asperities increases, it leads to normal creep. This causes both surfaces to press each other like in the *squeeze* effect in lubricated friction. In figure 7 can be seen how the different types of frictional lag stick to the Stribeck curve (in red). The higher the acceleration the higher will be the deviation from this curve.

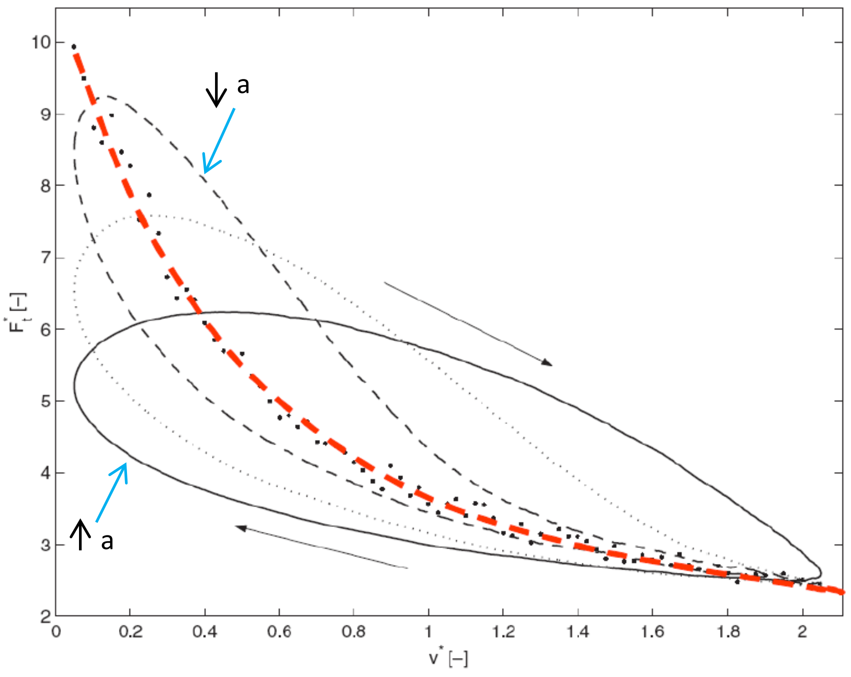


Figure 7: Different frictional lag cases (from [1]).

In lubricated contacts the friction phenomena is similar to the above explained theory. In [3] the effect in lubricated contacts is explained in high extend. Figure 8 shows the evolution of the interaction between two lubricated surfaces with relative motion. Basically it differentiates between four lubrication regimes. Static Friction (I): the asperity junctions deform elastically because the static friction force has not been reached yet. Boundary Lubrication (II): there is some junctions breaking since the two surfaces are starting to slide. No lubricant film is formed. Partial Fluid Lubrication (III): the lubricant fills in major gaps in the contact area. It is drowned in mainly by sliding or rolling. There is still some solid-to-solid contact. Full Fluid Lubrication (IV): the lubrication film completely separates the two materials, supporting the entire load.

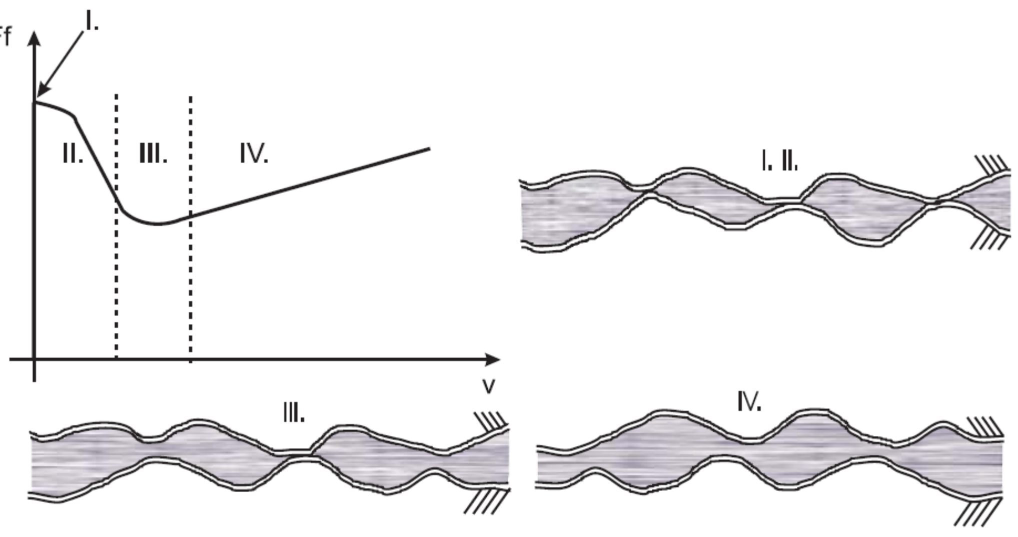


Figure 8: Different friction regimes in a lubricated contact area (from [3]).

From here it can be observed that there is a high resemblance in the friction of a lubricated contact area and that of a not lubricated one. The first two lubricated regimes are similar to the above described presliding regime and the third and fourth correspond to the gross sliding one.

2.3 Most used models

As stated before the study of friction has attracted a considerable attention from researchers in the past years. As it is usual, in the beginning researchers just tried to replicate what they could see at their experiments. In other words, most of the models until quite recently are empirical since they reproduce the noticeable effect of friction from own interpretations and assumptions without caring much about the physics behind it [2].

This practice has been useful so far. But as new and more demanding requirements have been introduced, other kind of models were needed which could imitate closely all the different physical phenomena occurring all along the different regimes found in practise. These are called physics-motivated or analytical models, which start from the microscopic asperity interactions and then develop its way to explain the macroscopically observed friction.

But before explaining some examples from the latter, it is better to get a look to the simpler models. And it is due to this simplicity that they have been used extensively, since usually the goal is to have some sort of balance between model complexity and goal completion.

Most of these models are called static because they are only velocity dependant and do not take into account the dynamics of the problem. In figure 9 one can see a progression of different models which ends with the Kinematic Friction Model (KFM). In a) pure Coulomb friction, in b) viscous friction is added, later in c) static friction is also taken into account and in d) a smooth transition is provided by means the Stribeck curve.

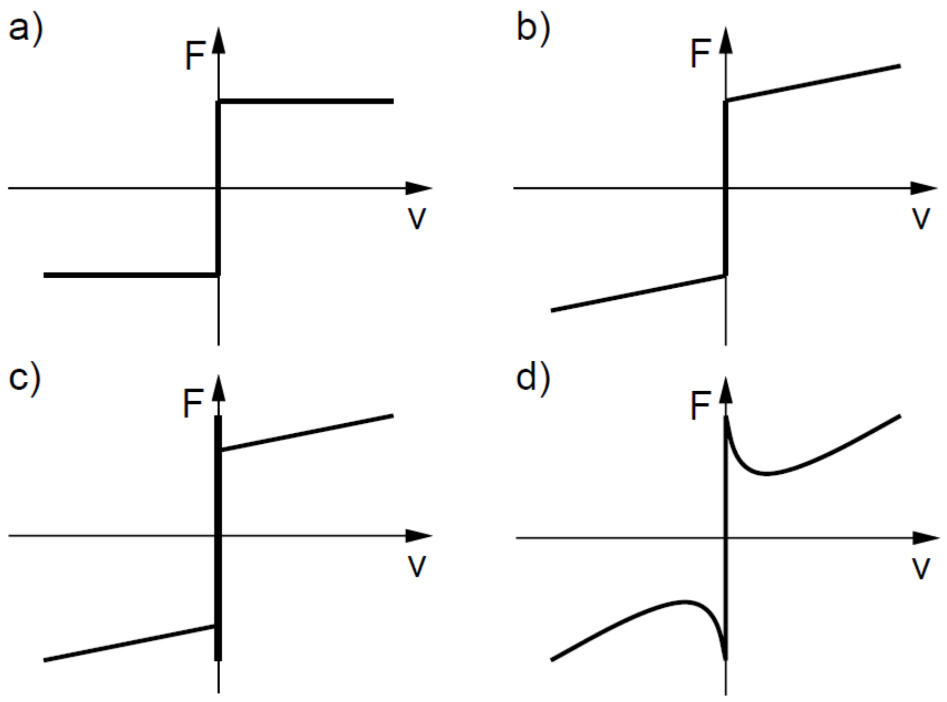


Figure 9: Static friction models evolution (from [4]).

From this picture it can be said that this evolution is quite natural and that the last model, the KFM (figure 9.d), includes the most important of the friction macroscopic effects; being the most important the static friction, the Stribeck effect and the viscous component. This is the equation for the KFM model:

$$F(v) = F_c + (F_s - F_c)e^{-\left|\frac{v}{v_s}\right|^{\delta_s}} + \sigma_2 v$$

where F_c and F_s are the Coloumb friction and static friction respectively. v_s is the Stribeck velocity and δ_s defines the shape of the Stribeck curve. The second term depends on σ_2, which is the viscous friction parameter.

The identification of its five parameters is feasible although, at the same time, the model presents some big differences with the real life phenomena. These are basically the discontinuity at zero velocity and the independence from accelerations.

The discontinuity at zero velocity is related to the requirement of a direct relationship between the stiction force F_s and the external force F_e. This is because the stiction force opposes to the external force until there is movement, so it does not depend on velocity. A better modelling of this matter could be obtained in the following manner [4]:

$$F_s = \begin{cases} F_e, & \text{if } v = 0 \text{ and } |F_e| < F_{s\_max} \\ F_{s\_max}\, sgn(F_e), & \text{if } v = 0 \text{ and } |F_e| \geq F_{s\_max} \end{cases}$$

Although this would be a more genuine approach to the handling of the zero velocity zone it does not solve the actual problem, which is the discontinuity itself.

Another way of overcoming this discontinuity, as explained in [5], is to place a high sloped line up to a very small threshold ε as seen in figure 10, transforming discontinuity in a rapid friction jump.

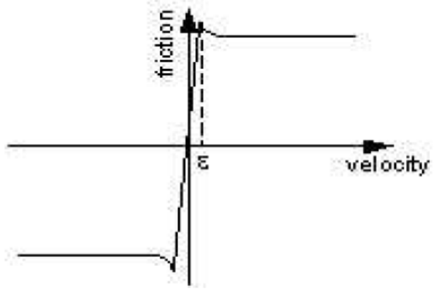


Figure 10: Use of a small threshold in order to avoid the zero velocity discontinuity (from [5]).

This correction may be useful for the numerical solution of the model, however it is far from an optimal solution for the motion stop, and velocity reversal, cases in high precision applications.

In the same direction went Karnopp when he proposed to use a small interval around zero velocity at which the virtual velocity used for computation would be zero [6]. This modification had the purpose of facilitating the detection of zero velocities and to avoid switching between different state equations for sticking and sliding. In the Karnopp model, when the real velocity is between this interval the output friction force is either a saturated version of the external force or some static value dependent on the velocity. The main problem of this approach is that it uses the external force as an input and this information is not always available. Moreover, despite the advantage of this modification in terms of computation, it does not agree with reality.

Other attempts, e.g. Armstrong's model [7], tried to use discontinuous functions so as to distinguish between the presliding and the gross sliding regimes. Although this solution may appear suitable it brings the problem of the transition between functions, which is not obvious. Apparently, it would suffice to find the velocity from which sliding starts, but this one varies depending in other factors like accelerations so in the end the implementation of the model becomes complex.

Some other improvements were worked trying to get the best performance out of the static models, but again, not taking into account the dynamic aspects of the system. However, these modifications provided little advantage over the first static models.

One of the first dynamic models presented in literature was the Dahl model [8]. It was primarily intended for simulating systems with ball bearing friction although it has been later used for adaptive friction compensation in diverse situations with success. Dahl modelled the classical solid mechanics stress-strain curve by the following differential equation:

$$\frac{dF}{dx} = \sigma_0 \left(1 - \frac{F}{F_c} sgn(v)\right)^{\alpha}$$

where σ_0 represents the stiffness coefficient and α is a shaping parameter of the stress-strain curve of the studied system (figure 11). From the equation it can be noticed that the friction force depends on the displacement and the velocity sign but it is independent of the velocity. This independence from the velocity makes it possible to use the theory of hysteresis operators.

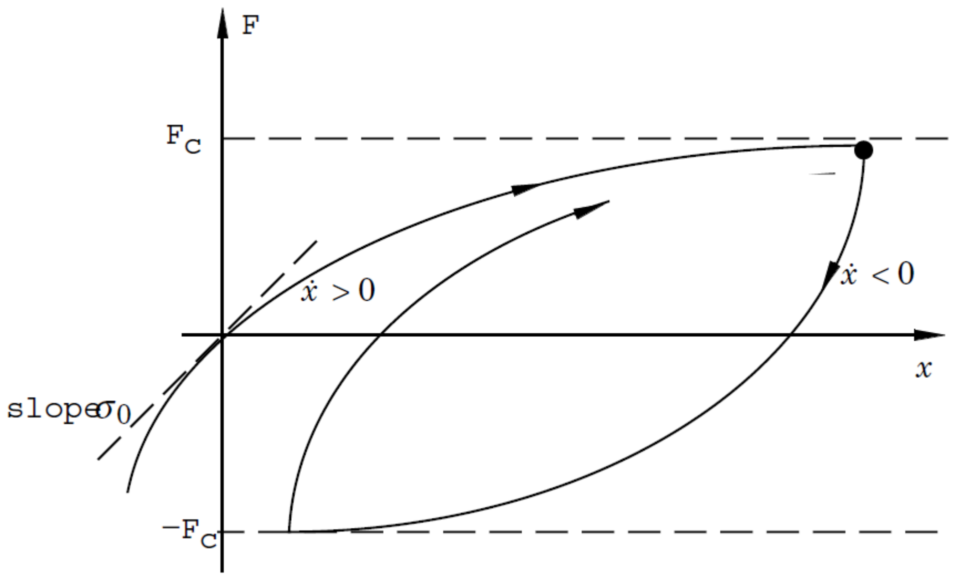


Figure 11: Friction force vs. displacement curve for the Dahl model (from [4]).

A higher parameter α will result in the hysteresis from the image above having sharper bends at the velocity reversal points, and therefore, a flatter oval shape. As can be observed the friction force $|F|$ will never be greater than F_c if the initial value is chosen so as $|F(0)| < F_c$.

To obtain a time domain form of the above referred equation one can employ the following transformation:

$$\frac{dF}{dt} = \frac{dF}{dx}\frac{dx}{dt} = \frac{dF}{dx}v = \sigma_0 \left(1 - \frac{F}{F_c} sgn(v)\right)^{\alpha} v$$

As can be seen this equation represents a generalization of the ordinary Coulomb friction, hence it does not capture the Stribeck effect neither the stiction.

One of the first models that tried to capture the physics at microscopic scale was the Bristle model [9]. A number *N* of random scattered contact points are bonded as explained before through flexible bristles. As the relative displacement between the two surfaces increases the strain acting on the bristles increases, and these at the same time act as springs that produce the friction force. When one bristle breaks, another one forms up in a different location. Therefore the friction force can be expressed as the addition of the forces acting on the different bristle bonds:

$$F = \sum_{i=1}^{N} \sigma_0 (x_i - b_i)$$

where *N* is the number of bristles, σ_0 the bristles stiffness, x_i the relative position of the bristles, and b_i the location where the bond was formed. When $|x_i - b_i|$ reaches a certain prefixed value the bond breaks and a new one is formed at a random location.

Two interesting features can be found in this model. First, the bristles stiffness, σ_0, can be made velocity dependent if needed. And second, this model recreates the random nature of friction. This randomness can be increased with the number of bristles, however, the overall computation performance decreases. Another possible disadvantage is that, due to the lack of damping on the bristles, the motion in sticking might become oscillatory.

Trying to improve the high computation cost of the Bristle model the same authors proposed the Reset Integrator model [9]. This new approach kept constant the strain of a bristle just before it would break instead of letting it snap. Additionally the equations are formulated in a way that, when this happens, the friction force decreases:

$$F = \big(1 + a(z)\big)\sigma_0(v)z + \sigma_1 \frac{dz}{dt}$$

where $\sigma_1 \frac{dz}{dt}$ is a damping term that is only active when sticking, being σ_1 the corresponding damping coefficient. $\sigma_0(v)$ is an arbitrary function that represents the friction force when slipping, while $a(z)$ adds the stiction behaviour to the equation with the following definition:

$$a(z) = \begin{cases} a, & |z| < z_0 \\ 0, & \text{otherwise} \end{cases}$$

If $|z| < z_0$ there is sticking and the friction force is a function of *z*. The parameter *z*, which is the extra state used to define the strain in the bond, can be defined as follows:

$$\frac{dz}{dt} = \begin{cases} 0, & (v > 0 \text{ and } z \geq z_0) \text{ or } (v < 0 \text{ and } z \leq -z_0) \\ v, & \text{otherwise} \end{cases}$$

This model is indeed more efficient than the previous one, but it presents some discontinuities in *z* and it adds the necessity of computing the bristles' strain, which requires an additional computational effort.

The LuGre model [10] represents another evolution of the Bristle model. It incorporated a different approach for the simulation of the interaction between bristles from both surfaces. An internal state denoted by the variable *z* is introduced. It models the average deflection of the bristles, which itself generates the friction force. When a bristle is sufficiently deflected it will start to slip. The model equations in standard parametric form are the following:

$$\frac{dz}{dt} = v - \sigma_0 \frac{|v|}{g(v)} z$$

$$F = \sigma_0 z + \sigma_1 \frac{dz}{dt} + \sigma_2 v$$

Here, σ_0 and σ_1 represent the material stiffness and damping coefficients, respectively. σ_2 is the coefficient of the viscous term of the equation.

The function *g(v)* includes part of the KFM model and usually is written as:

$$g(v) = F_c + (F_s - F_c)e^{-\left|\frac{v}{v_s}\right|^{\delta_s}}$$

where F_c and F_s are the Coloumb friction and static friction respectively. v_s is the Stribeck velocity and δ_s defines the shape of the Stribeck curve. In [11], a complete description and validation of the properties of this model can be found.

The LuGre model already takes into account the most important friction effects except the nonlocal memory. Figure 12 depicts a torque vs. position plot that results from following the path seen in the left side. It shows how the LuGre model is able to reproduce the real phenomena but is not able to retake the previous position when exiting from a local hysteresis loop (see the detail of those points at the right hand side of the figure). Although in [11] this is called the reversal point memory it refers to the nonlocal memory effect explained in [1] for the presliding.

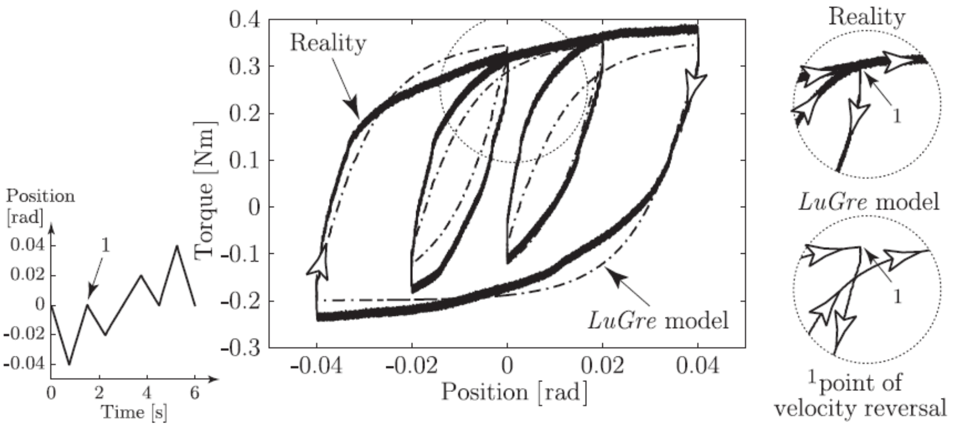


Figure 12: Friction torque vs. position comparing the LuGre model with reality (from [11]).

As explained in [5], this model has got high popularity in the past years. The reason for this is the right balance between computational cost and simulation accuracy and, in spite of the criticism due to the non-inclusion of the nonlocal memory, it is still employed in many applications and it is reported to lead to good results.

Later, the same authors proposed an extension in form of the Leuven model [12]. The most important improvement of the model consists of the swapping of the state *z* by a hysteretic function $F_h(z)$ with nonlocal behaviour [13]:

$$\frac{dz}{dt} = v\left(1 - sgn\left(\frac{F_h(z)}{s(v)}\right)\left|\frac{F_h(z)}{s(v)}\right|^n\right)$$

$$s(v) = F_c + (F_s - F_c)e^{-\left|\frac{v}{v_s}\right|^{\delta_s}}$$

$$F = F_h(z) + \sigma_1\frac{dz}{dt} + \sigma_2 v$$

The incorporation of the hysteric function improves the model's performance in nonperiodic presliding, however induces considerable implementation difficulties that make the model less attractive. To overcome these problems $F_h(z)$ can be implemented using a Maxwell Slip model, so its value is equal to the sum of hysteresis forces of each element [14]:

$$F_h = \sum_{i=1}^{N} F_i$$

The Maxwell Slip model [15] consists of several elements as the Bristle model but they behave in a different way. Essentially the model is composed by *N* elasto-slide elements in parallel (figure 13). Each element is composed by a block with an input *z* and an output F_i. These blocks are connected with the other elements by means of their own spring with constant k_i and they have their own characteristic maximum force W_i. Each block has also its own state variable ζ_i which represents the position of the block.

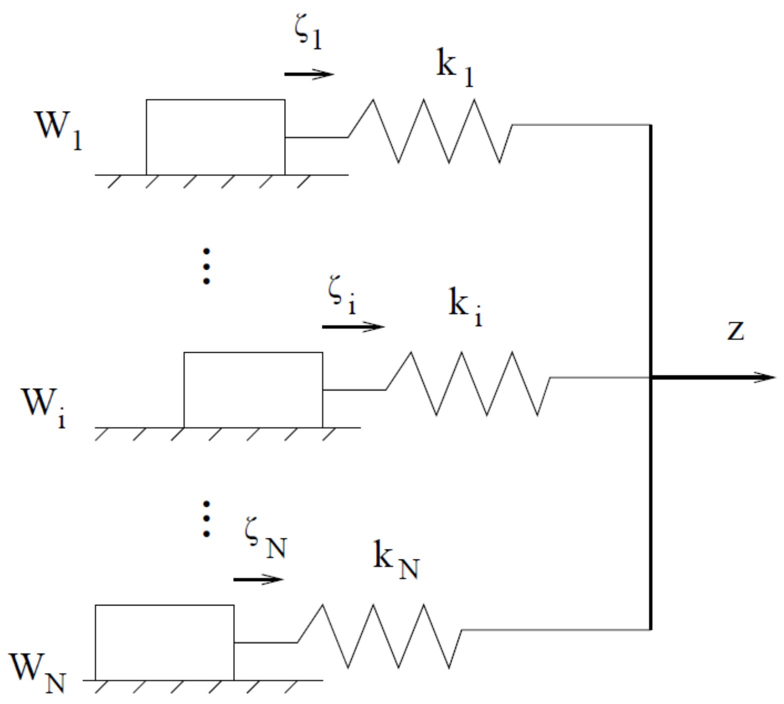


Figure 13: Maxwell Slip model of N elements (from [13]).

One of the latest models appearing in the literature is the Generalized Maxwell-slip model (GMS) [16], which combines the Leuven and the LuGre models trying to recreate the friction in a more realistic form. This model is not totally physics-motivated since it incorporates parts of previous models which are empirical. Despite this, its foundations are based on the processes governing at microscopic scale and its results agree well with the observed macroscopic behaviour.

The GMS model applies the rate-state mechanism of the LuGre model to each block of the Maxwell-Slip part of the Leuven model instead of the simple Coulomb law. The GSM equations are as follows [1]:

$$\frac{dz_i}{dt} = \begin{cases} v, & (if\ sticking) \\ sgn(v)C_i\left(1 - \frac{z_i}{s_i(v)}\right), & (if\ slipping) \end{cases}$$

$$F = \sum_{i=1}^{N} (k_i z_i + \sigma_i \dot{z}_i) + f(v)$$

The element *i* will remain sticking as long as $z_i \leq s_i(v)$. The GMS model is a powerful approach to friction modelling because it describes all present effects caused by friction while maintaining itself as a feasible option from the computational speed point of view. This has made it, together with the simpler LuGre, one of the most employed models in dynamic systems with relevant friction behaviours.

2.4 Model selection

As it can be seen from the previous chapters friction is a phenomenon with complex behaviour that goes through different domains and scales. As a result, many solutions have been brought up over the years, from the simple static models as the KFM up to the more complex dynamic ones, like the GMS.

Now the arising question is what model to choose from all the available. To answer this question it is necessary to address two other questions:

- What do we want to model with it?

That is, what is it that we want to model? Usually it may depend on the final application. For example, it is not the same to seek an average friction estimation on a shaft in order to measure its wear than to require the exact instant friction in a motor block for compensation purposes.

The last question is:

- Why do we want to model it?

It is highly related to the previous but not exactly the same. This is the ultimate goal, what are we going to do with our model? One could also formulate these questions in reverse order: first think of why is a friction model needed and then come up with the necessary characteristics to be modelled according to these needs.

Once this is cleared up, one should find the right balance between solution performance and complexity. The model should then be the one which achieves the desired results with the least development cost.

Answering the second question first, as stated before, one of the sub-goals of the project is to expand the already available model by applying friction compensation and improve the behaviour of the youBot base for control purposes.

The main reason behind this is that in the actual control mode the base acts as the first link of the robot, forming part of the whole kinematic chain. This means that when moving to some far point the control algorithm distributes the whole movement needed to achieve it into smaller ones for each joint. However, the velocity control on the wheels does not work properly because the feedback data is not good (especially at low velocities).

This is why, instead of using a closed loop velocity control approach, the necessary force to move the joints a specified distance will be directly commanded to the youBot in open loop. The problem is that if some of these joints were to behave in a strange way –because of a high friction, for example– the robot would not be able to reach the objective.

Now, the best and complete solution would be to identify the friction in all joints but this would imply a higher complexity. First, and in relation with the aforementioned first question, from where is the friction modelling wanted? From the motor only? From the motor plus the gear box? Since friction's main drawback is the loss of energy between input and output, one should address it as a whole. In this project the friction that has been modelled comprises the block between the input electrical signal until the output shaft torque.

The next decision to be made is whether this is feasible or not for the different joints of the robot. This essentially depends on the set of experiments which are necessary to identify the parameters of whatever friction model one were to choose. Majorly, these experiments require the execution of long movements –several full rotations in this case– which sometimes cannot be fulfilled because of the actual structure of the robot. This happens for example for all the joints of the youBot's arm which have a limited motion range, making it almost impossible to carry any identification test. A possible solution to this would be to dismount the joints one by one and do the experiments on the standalone motors, but then the resulting friction model would not account for the other components inside the motor block as intended.

Moreover, as it is well known, friction is dependent on the normal force between the interacting surfaces. This brings even more difficulties to the friction identification since the friction at a certain joint of the arm for example, would depend on the positioning of the next links at that moment because of gravity and inertial forces [17], [18]. However, a load dependent friction model is beyond the objectives of the present MSc thesis and should be undertaken in a larger project if necessary.

Another valid option to the problem would be a complete system identification and validation as explained in [19] or [20]. Although this would greatly benefit the robot's behaviour for any control needs, it requires more complex steps and so a longer time would be needed to achieve acceptable results.

Despite having performed a full review on system identification procedures, a complete identification of the youBot falls beyond the scope of this study. If there would be a need for a superior performance it is highly recommended to proceed this way. Nowadays there are some techniques that can provide a good estimate of the full system parameters with a single meticulously-prepared motion of the robot [17].

2.4.1 Selected model

The initial model selected for this project was the LuGre model. As explained above it provides a quite good balance between problem solving and computation cost, which as described before is defined through the following set of equations:

$$\frac{dz}{dt} = v - \sigma_0 \frac{|v|}{g(v)} z$$

$$F = \sigma_0 z + \sigma_1 \frac{dz}{dt} + \sigma_2 v$$

$$g(v) = F_c + (F_s - F_c) e^{-\left|\frac{v}{v_s}\right|^{\delta_s}}$$

where the parameters F_c, F_s, v_s, δ_s and σ_2 must be identified from the analysis of the steady state behaviour of the friction phenomena, while σ_0 and σ_1 must be identified by observation of the dynamics of the system.

This model describes most of the friction phenomena without bringing too much complexity to the identification and validation steps. It is not as good as the GMS but is the second best model to bring a powerful approach to friction modelling yet keeping its predecessors simplicity.

2.5 Experimental campaign

As commented above, all friction models have some empirical coefficients which must be estimated by direct comparison with experimental results.

Additionally, it is important to remark that although one should have a model in mind before doing the experiments, the results can be different than expected. Because of this, it is advised to do a first set of experiments to unmask any possible unforeseen behaviour from the setup in order to rethink the identification process again.

As stated before, friction is majorly composed of two different regimes, the steady state and the dynamic one. Thus, different experiments are required to obtain the parameters for each part of the model and so they will be explained separately.

2.5.1 Steady state experiments

As explained in [13], the typical experiment necessary to identify the parameters mentioned above is fairly simple. The main idea is to record as many pairs of torque-velocity points as necessary to draw the complete Stribeck curve. To do this one should use velocity control and wait for it to get steady before registering the torque required to maintain that velocity. Because there are no accelerations the friction force is equal to the measured torque for that constant velocity. An example of this kind of experiment can be found in [3].

The youBot was positioned in a way in which its wheels would not have contact with the ground nor with any other object. Even though the friction model will be used while the youBot is on the ground (the wheels will be loaded) after shuffling different possibilities it was decided to do the experiments with this configuration because it was the only feasible option that did not require an expensive setup.

First the direct signal of the youBot velocity was employed for the tests but the analysis of results showed that the estimation of the velocity was very poor, as suspected. The data would appear saturated with high frequency noise. The noise is not that significant at mid-high velocities but at low velocities it clearly disturbs the original signal. Therefore, the velocity was estimated offline from the position measurements and employing an SVF filter in order to reduce the noise. This resulted in a noticeable noise reduction and a cleaner signal. However, this was only for correct analysis of the recorded data. The velocity control was still done by the youBot using its own velocity estimation.

A LPF (low pass filter) was used on the motor torque measurements. Despite improving the signal the benefit was marginal since what is needed is the average of the collected data points.

The tests were performed following the procedure proposed in [3], which is depicted in figure 14. Each measurement consists of a total of 90 seconds (T1 + T2 + 10 s extra). There would be a waiting for the signal to stabilize of 50 seconds (T1). From then and during 30 seconds the actual measuring would be carried out (T2).

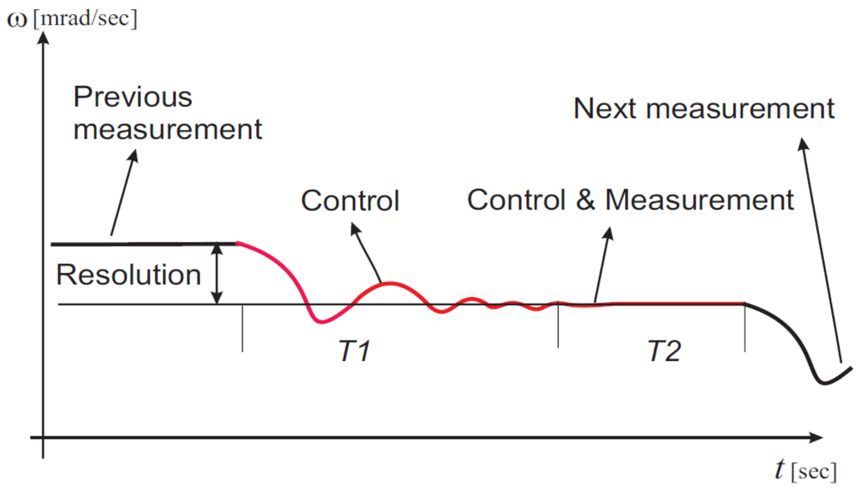


Figure 14: Test procedure for one measurement (from [3]).

At lower velocities the curve changes more rapidly so a higher number of measurements were taken. For the higher velocities, knowing just some points is enough since a straight line is expected.

In the next page table 3 shows a summary of the different performed tests. The covered velocity range goes up to 10 rad/s, which corresponds to a wheel velocity that the youBot will hardly exceed in practical cases. Different sets were needed in order to cover different ranges of velocities with different resolutions. Some sets overlap because in certain cases it was necessary to assure the validity of the obtained results.

From velocity (rad/s)	To velocity (rad/s)	Δw	N of measurements
0	0,5	0,05	10
0	0,5	0,005	100
0,5	0,75	0,005	50
0,75	2	0,05	25
0,7	4	0,1	33
2	10	0,5	16
0	-1,5	0,05	30
-1	-4,5	0,1	35
-4,5	-10	0,5	13

Table 3: Measurements sets for the steady state parameters estimation.

Most of the times the friction force is not symmetrical, what is to say, it is different for positive than for negative velocities. This is why also some measurements were taken with negative velocities to assure that the plot was, if not identical, similar to its positive counterpart.

The experiments were carried out in all four wheels simultaneously because they may have different friction effects on their correspondent motor block. If this is the case it is necessary to model friction at each wheel independently.

The software interface used with the youBot when realizing the experiments has been Orocos under Ubuntu Linux OS. More information on the employed software is given in the section *Software* under *Mains used* (1.3). A data acquisition program was specifically developed so the different signals' ports could be traced adequately. The files were saved as *.csv* (Comma Separated Values) so they could be easily imported into Matlab.

Once the data is collected a plot friction force vs. velocity can be composed in order to proceed to the parameter identification. Nowadays very powerful software is available with which curve fitting becomes quite easy. In this project the *Curve Fitting* toolbox from Matlab has been used for this purpose.

In the following page can be found figure number 15. It shows the entire set of friction force results obtained from the previous experiments. As it can be observed, all the curves (each one representing the friction of a wheel) have a similar overall shape.

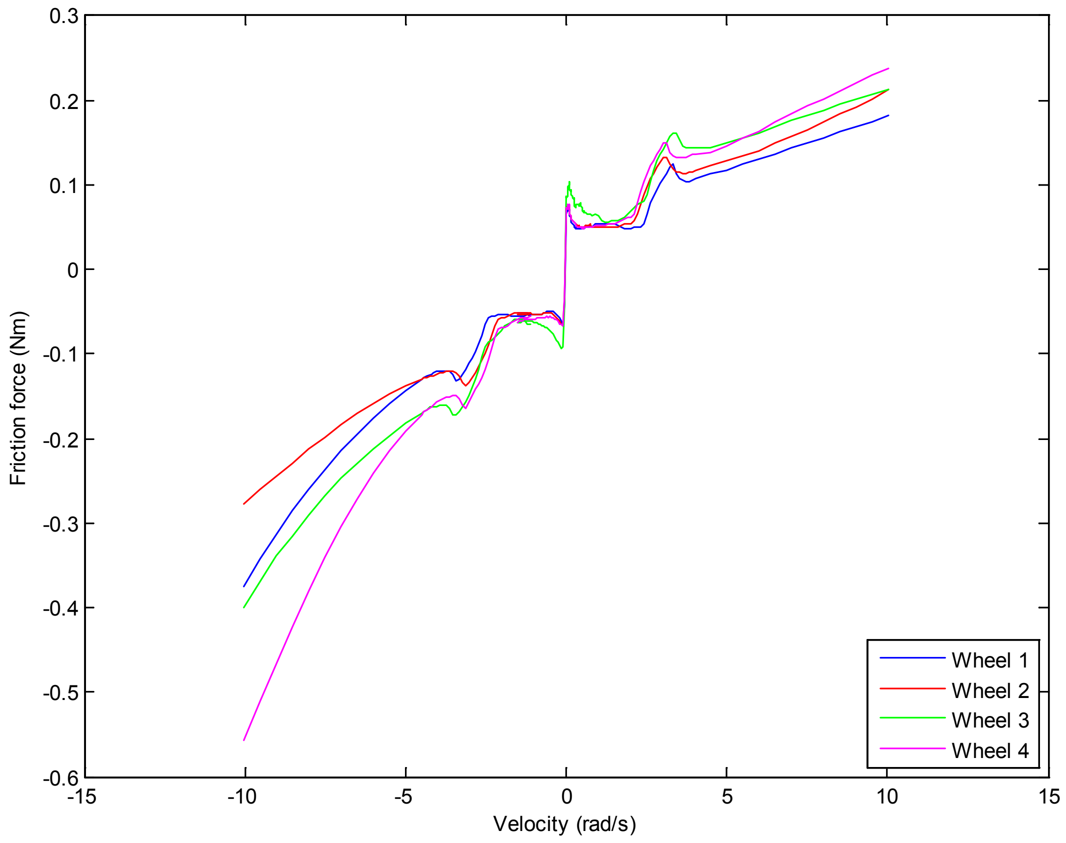


Figure 15: Friction force vs. velocity plot for the different motor blocks of the youBot.

As one can see the plot shows the expected behaviour until 1 rad/s. For velocities between 1 and 4 rad/s the friction drops considerably instead of increasing proportionally with velocity. At higher velocities the curves follow a straight line according to theory. The only difference between the positive and the negative part is at the high velocity range. Instead of describing a straight line, the friction increases exponentially with increasing negative velocity. This could be caused by a poor backdrivability on the geared transmission.

The observed behaviour between 1 and 4 rad/s is strange and does not follow the expected trend. When the friction force should start increasing with increasing velocity, it stays low and then it increases abruptly before connecting with the straight line that represents viscous friction.

It should be noticed that this unexpected behaviour has been observed in all four wheels and for both positive and negative velocities. To assure that this had no relation with the wheel position and therefore, with the possible effect of gravity load on the axle, an extra experiment was carried out with youBot’s left side up. In the following image can be seen how a spirit level is being used to assure the wheel axle perpendicularity with the ground.

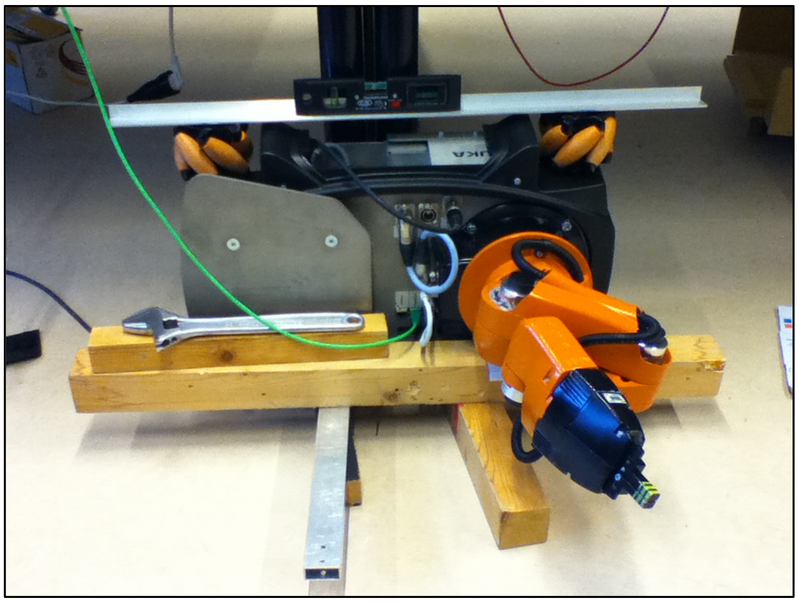


Figure 16: Experiment with the youBot left side up.

The same results were obtained in this last test, proving that the friction on the motor block of the youBot's wheels deviates from the standard behaviour at mid-velocities.

It is important to remark that the standard behaviour is based on simple interacting surfaces (block over surface or standalone motor) while in the studied case a complete motor block was analysed, including the geared transmission, bearings and other components.

Another possible explanation could be of thermal origin, where the motor-generated heat would influence the lubricant viscosity and thus the friction. This, however, would affect the experiments in all their range and not only a specific region.

More research would be necessary in order to identify the possible causes of this non-standard behaviour.

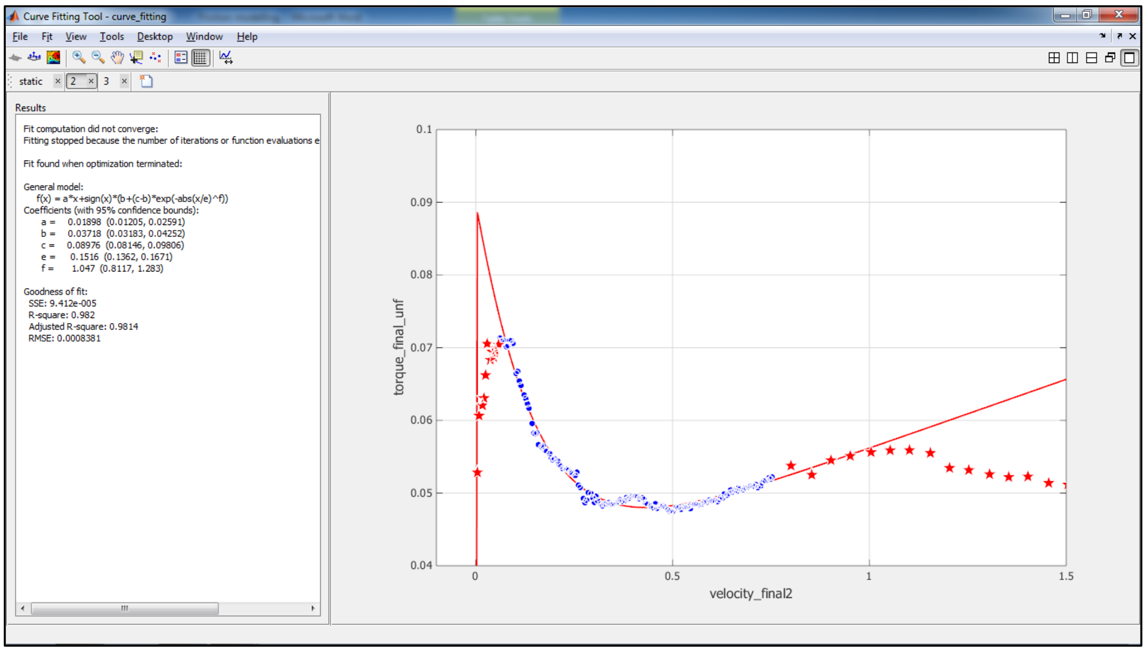
As it can be observed the actual friction force on the youBot motor blocks is rather complex and none of the above mentioned models is able to describe it properly.

The LuGre model provides a very good option for the modelling of the dynamic part of the friction including velocity reversals. However, this model is not able anymore to simulate the obtained friction behaviour at the gross sliding regime.

Since the dynamic part affects mostly the low velocity regime –where the presliding occurs– there was one feasible solution to this problem. The LuGre model would be used for the low velocity part of the friction while at higher velocities, where accelerations are not that dominant, one could better use a customized curve to describe that region.

So with this in mind the parameters which pertain to the static part of the LuGre model were obtained by fitting the analytical curve to the low-velocity part of the acquired results. To achieve this, the KFM model expression was employed because it is an equivalent of the static part of the LuGre model and they share the same parameters.

Figure 17 shows a comparison between the fitted curve and the experimental results for wheel number one. The measurements data was pre-processed in order to leave only the interesting points needed for the estimation steps (the red stars represent the excluded points).



```
Fit computation did not converge:
Fitting stopped because the number of iterations or function evaluations e

Fit found when optimization terminated:

General model:
    f(x) = a*x+sign(x)*(b+(c-b)*exp(-abs(x/e)^f))
Coefficients (with 95% confidence bounds):
      a =    0.01898  (0.01205, 0.02591)
      b =    0.03718  (0.03183, 0.04252)
      c =    0.08976  (0.08146, 0.09806)
      e =     0.1516  (0.1362, 0.1671)
      f =      1.047  (0.8117, 1.283)

Goodness of fit:
  SSE: 9.412e-005
  R-square: 0.982
  Adjusted R-square: 0.9814
  RMSE: 0.0008381
```

Figure 17: Snapshot of the fitted results for the static parameters identification (wheel 1).

As it can be observed the model is able to reproduce these results obtained at low velocities with high accuracy.

The parameters that were obtained for the first wheel are presented in table 4:

Parameter	**Estimated value**
F_c	0,03718
F_s	0,08976
v_s	0,1516
δ_s	1,047
σ_2	0,01898

Table 4: Static parameters obtained for the LuGre model of the first motor block.

2.5.2 Dynamic state experiments

In order to obtain the remaining parameters of the LuGre model –which are of a dynamic character– new specific tests are required.

Several methods for this purpose can be found in the literature. The main idea behind them is to submit the system to very small forces so as to have zero velocity. The range of intended displacements is at micro-level.

As a first estimation a very easy experiment was done. It is also explained in [21] and more detailed in [22]. The goal of this simple test is to capture the moment at which the first micro-displacement happens. To do this a very slow-increasing torque ramp was be applied to the motor while monitoring the shaft’s encoder.

The next equation describes the simple relation behind this experiment:

$$\sigma_0 = \frac{\tau_1}{x_1}$$

where τ_1 represents the torque at which the wheel encoder detected the first displacement with value x_1.

This is done in open-loop control and provides only a rough estimation of σ_0. Once σ_0 is calculated σ_1 can be estimated by using the following empirical equation ([23], [24]):

$$\sigma_1 = 0.2\sqrt{J\sigma_0}$$

where J is the total inertia on the wheel.

With this simple approach a good first set of values can be obtained which will be used as initial guess for the following identification procedures. Next figure shows the results from this first test:

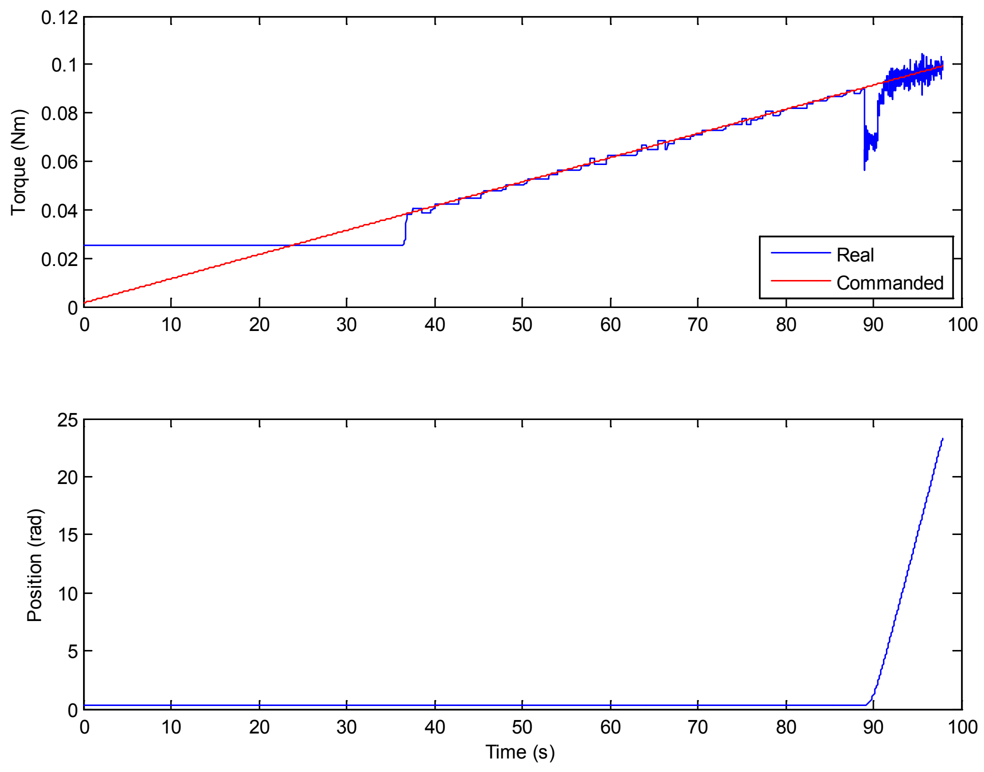


Figure 18: Torque ramp experiment in one of the youBot's wheels.

where the red line is the commanded torque and the blue line corresponds to the actual motor input. As it can be appreciated the real motor torque stays at a certain value and once this is reached it starts to follow the demanded signal (here it displays some lag because the curve's slope is quite slow). This fault forces the motors' input signal to a small offset value of about 0.03 Nm when the commanded value is below it. It is thought that the reason behind this has a physical nature which could not be overcome. The cause could be a design error in the motor controllers. However, this could not be proved or investigated since it fell out of this project's scope and also due to lack of time.

The torques at which the wheel encoder detected the first displacements are shown in table 5. Two different tests were realized for consistency.

Position (rad)	**Commanded torque (Nm)**	**Real torque (Nm)**	σ_0 ($\frac{\mathrm{Nm}}{\mathrm{rad}}$)
0,2677 -> 0,2678	0,0377	0,0251	251 – 377
23,9072 -> 23,9073	0,0283	0,0156	156 – 283

Table 5: Experimental results for the torque ramp experiment.

From these, an averaged value of 250$\frac{\mathrm{Nm}}{\mathrm{rad}}$ was chosen for σ_0. And correspondingly a value of 0,127 $\frac{\mathrm{Nms}}{\mathrm{rad}}$ was used for σ_1. Although the previous experiments were not very consistent they are meant only as a first guess for the actual identification process.

The next experiment useful for identifying the dynamic parameters of the model is the Dahl curve [11]. Here the motor input is a sinusoidal wave. The amplitude must not overreach the static friction force because the goal is to have only micro-displacements. For this the only necessary signals to be traced are the actual motor torque and the encoder data.

It is enough to do 7 to 10 cycles to get a representative outcome. Once the data has been gathered a friction force vs. displacement plot can be laid down. In the next figure can be seen its theoretical shape:

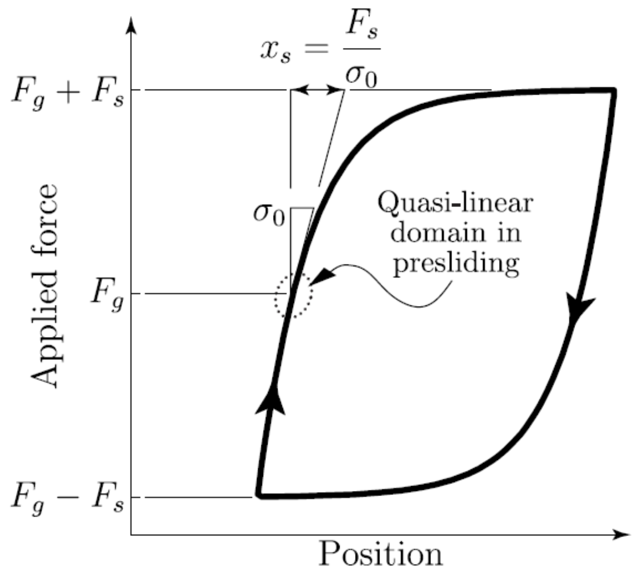

$x_s = \frac{F_s}{\sigma_0}$
$F_g + F_s$
σ_0
Quasi-linear domain in presliding
F_g
Applied force
$F_g - F_s$
Position

Figure 19: Theoretical shape of the Dahl curve (from [11]).

As it can be observed, in the Dahl curve there is a *quasi-linear* behaviour in the presliding regime. Taking into account the correspondent offset force F_g one can get σ_0 as the slope at the centre part of the Dahl curve.

However in the experiment realized on the youBot the following plot was obtained:

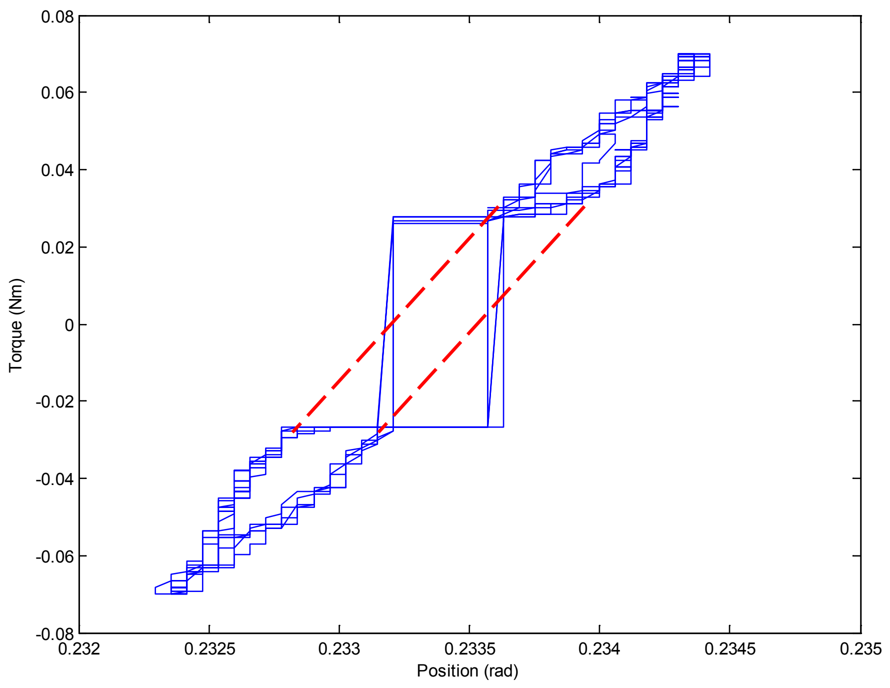


Figure 20: Obtained results for the Dahl curve experiment.

Two dashed lines have been inserted in the figure to facilitate the comparison with the theoretical plot. The obtained results are similar to the Dahl curve but have a rectangular shape in the centre of the figure. This shape is caused by the same problem existent in the previous experiment. This offset torque appears in both directions and spoils the quasi-linear region expected at the centre of the curve. This turned the Dahl curve as an unfeasible method for identifying the dynamic parameters of the LuGre model for the youBot.

By close observation of the results one can notice how the abscissae variations are of the order of 10^{-3} rad which indeed corresponds with the micro-displacements scale. Additionally, it is clear that the encoder resolution it is not high enough for this kind of procedure since the lines in the figure have a stepped shape.

Last but not least, there is also the inconvenient that the Dahl curve only allows the estimation of σ_0, hence another type of experiments is needed for the identification of σ_1.

The latter is normally achieved by other kind of experiment that consists of imposing velocity reversals to the system while staying in the micro-displacements region. It is in this kind of movements where the dynamic parameters are highly excited and therefore a good identification can be achieved.

A sinusoidal signal centred on the x-axis (when F_g is zero) will be enough in most of the cases although better wave shapes can be studied and used [11]. For example, depending on the commanding signal controllability one can play with different frequencies and see which bring the better results. In any case, the commanded forces have to be below the static friction force so the velocities are kept null. It is important to remark the fact that all these dynamic experiments are carried out in open loop.

This presliding experiment should allow to identify the parameters σ_0 and σ_1 by the use of simple optimization techniques.

Again, the Matlab application was used for this purpose, specifically the Simulink toolbox: Simulink Design Optimization - Parameter Estimation. This tool requires the inputs and outputs from the real system and a simulation model to work with. At each iteration it computes the simulated output using the original input and compares it to the real output. Different optimization methods as Nonlinear least squares or Simplex search can be used together with various converging algorithms.

The mathematical expression used to model the system is the dynamics equation:

$$J\frac{d^2x}{dt^2} = \tau - \tau_F$$

where *J* is the motor block inertia $J = J_{rotor} + \frac{J_{wheel}}{GR^2}$ and τ_F is the total friction force opposed to the wheel movement calculated by the LuGre model. The above was represented by the following Simulink model:

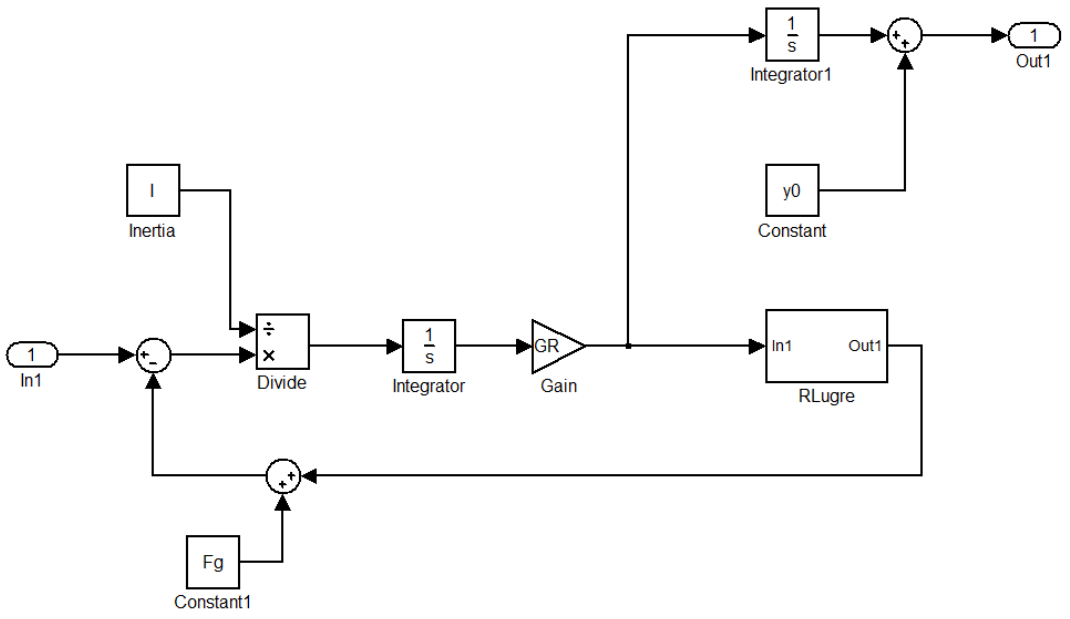


Figure 21: Simulink block diagram of the system.

As it can be seen, there are two offset blocks: F_g, which represents the offset friction force and y_0, which is the first value of the position measurements. The latter was used to align the simulated and real position curves from the beginning because sometimes the encoder would start with a value different than zero. As explained before the inertia block *"I"* comprises both the rotor inertia and the reflected wheel inertia at the rotor. Also, since the friction force has been calculated in relation to the wheel velocity a *GR* block has been included to assure consistency with the analytical model.

The LuGre friction scheme is depicted in the following diagram:

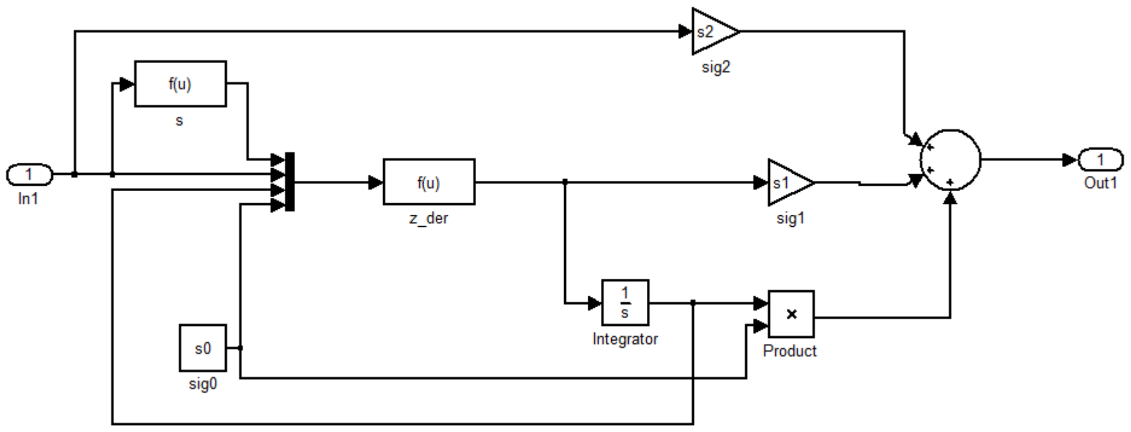


Figure 22: Simulink block diagram of the LuGre friction model.

Here the *"s"* block represents the *g(v)* function and in *z_der* the derivative of *z* is calculated, as stated in the LuGre model equations.

The main problem in our case was the minimum offset current/torque seen at the motor's input. Due to this the input torque resulted as seen in next plot:

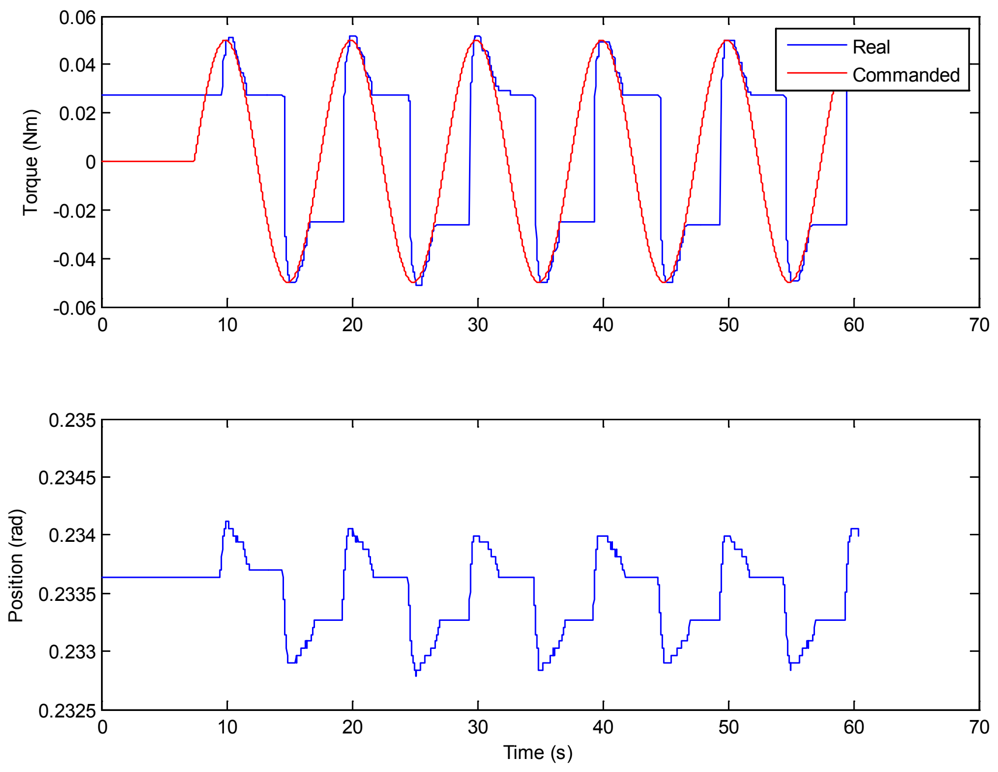


Figure 23: Centred torque input and output position.

It can be noticed how the position evolution is affected again by the same problem since it depends on the applied torque. As explained before the internal state z of the LuGre model has its major role at the zero velocity crossings, which means that the most important part of this experiment is being lost due to this unwanted effect. The same is applicable to the starting part of the plot because this state has to develop according to the initial conditions which are necessary for the model to work. As an additional step and realizing that the most feasible experiments usually used for the dynamic parameters estimation did not lead to good results an extra experiment was carried out.

Since the offset current on the motors was an issue with no solution for the moment, the idea was to bring the system to the stiction border in order to encourage a 'stick-slip' behaviour. In the next plot the results from this experiment can be seen:

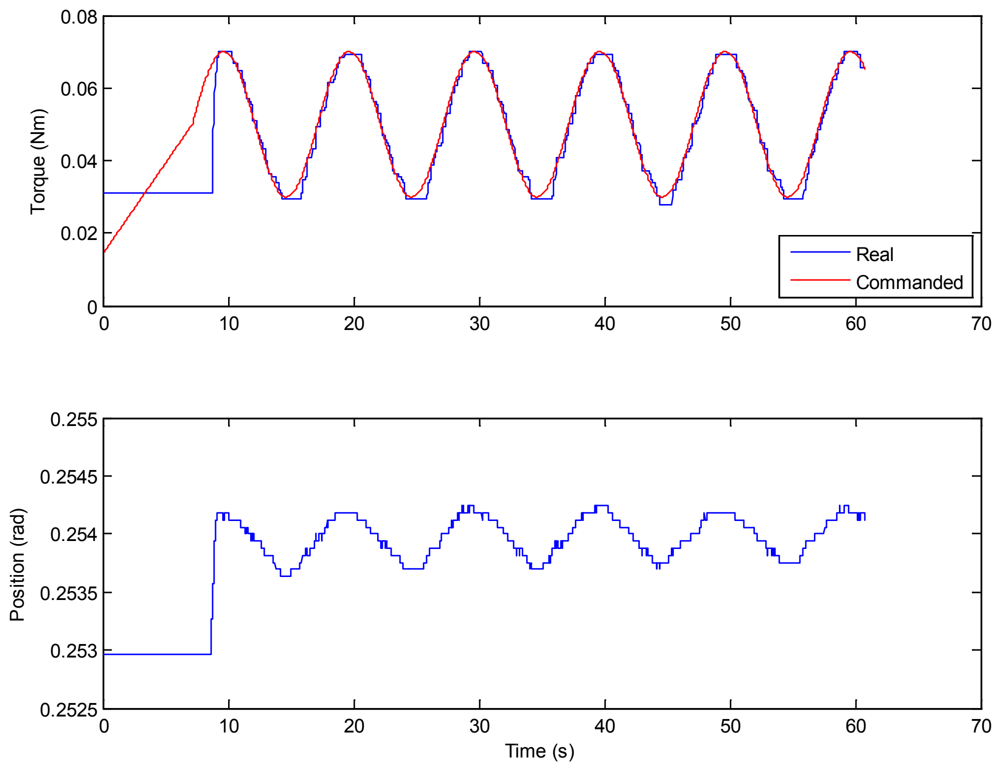


Figure 24: Positive torque input and position output.

Looking at the lower plot one can see how these are indeed micro-displacements since the position variations are very small. It appears that when the torque decreases the position does the same, what could be interpreted as an example of friction memory. However, when closely inspected one finds that the position local minimums are each time higher, which means that there is a small positive global displacement. Furthermore, the position evolution shows a stick-slip behaviour, only that in the sticking phase the position decreases instead of maintaining its value. It has not been proven but this could be due to the motor magnetic fields or even caused by vibrations, further investigation would be needed to understand this effect.

Despite not having velocity reversals the problem with the initial conditions and the unwanted starting point is still there in this experiment, so it became again not adequate as an estimation procedure.

Also here and in general, the encoders' resolution may be limiting the possibilities of identifying the dynamic parameters of a friction model with success. The encoders mounted in the base motors have 4000 pulses/rev while in similar studies encoders of up to 200000 or even 655360 divisions per revolution have been used, [11] and [25] respectively. In [23] is again stated that although nowadays robotic platforms dispose of good encoders for standard applications when estimating the parameters necessary to model the presliding regime of friction much precise encoders are needed.

2.6 Solution selected

Before knowing the results of the presliding parameters identification experiments, the idea was to model the non-standard region of the steady state part of the experimental results with custom made curves. In this way the LuGre model would have been used as it was designed and the only thing changed would have been the term σ_2, which takes importance once accelerations disappear. This was a valid modification because once the system dynamics activity decreases the internal state z and its derivative decrease to an almost negligible value. Therefore, instead of having a linear viscous term the idea was to include the peculiar shape recorded on the steady state experiments. This solution would have made feasible the use of the LuGre model to do something that otherwise would not have been possible with the already available models.

Unfortunately, as commented above, the experiments carried out in order to identify the parameters for the presliding part of the LuGre model were not successful. The main problems encountered being the offset current in the motors and the low resolution of the encoders for these high precision tasks.

The final decision was then to make use of the KFM model together with the custom made curves and employ a purely static model. However, the use of the KFM model turned out to be quite limited so it was later decided to curve-fit the entire set of experimental static results with user-made lines.

The Curve fitting toolbox mentioned before was used to tailor different curves to the friction data obtained during the steady state identification procedure. Through the use of discontinuous functions and using the velocity as the switching agent the complete static friction force simulation was achieved, for both the positive and negative parts. In the next page can be seen the equations for wheel number one in a Matlab function used to plot the friction force for a certain velocity range (the equations for all four wheels can be found in 7.1 *Appendix A*) followed by the overall fitted curve on top of the experimental results (figure 25).

```
function RYoubot

% Range of velocities to simulate.
v(:,1) = -10:0.001:10;

% Parameters for the different equations of the curves.
a =  0.05615;
b = -8.68;
c =  0.04245;
d =  0.2383;

a2 =  0.07022;
b2 = -0.1932;
c2 =  7.554e-10;
d2 =  6.858;

p11 = 0.07582;
p12 = -0.1201;

a3 =  3.243e9;
b3 = -7.739;
c3 =  0.07271;
d3 =  0.09647;

p21 = 0.0125;
p22 = 0.05584;

% The friction force is calculated in function of the velocity.
for i = 1:length(v)

    ryoubot(i,1) = 5*abs(v(i));
    ryoubot(i,2) = a*exp(b*abs(v(i))) + c*exp(d*abs(v(i)));
    ryoubot(i,3) = a2*exp(b2*abs(v(i))) + c2*exp(d2*abs(v(i)));
    ryoubot(i,4) = p11*abs(v(i)) + p12;
    ryoubot(i,5) = a3*exp(b3*abs(v(i))) + c3*exp(d3*abs(v(i)));
    ryoubot(i,6) = p21*abs(v(i)) + p22;
    ryoubot(i,7) = 0.06 + 0.4*abs(v(i));
    ryoubot(i,8) = 0.071 + 0.01*abs(v(i));

    % A different equation is used depending on the absolute value
    % of the velocity.
       if abs(v(i)) < 0.7
        n = 25;
        ryoubotb(i,1) = 1/(1/ryoubot(i,1)^n + 1/ryoubot(i,2)^n + 
1/ryoubot(i,7)^n + 1/ryoubot(i,8)^n)^(1/n);

    elseif abs(v(i)) >= 0.7 && abs(v(i)) < 1.5
        n = 30;
        ryoubotb(i,1) = 1/(1/ryoubot(i,2)^n + 1/ryoubot(i,3)^n)^(1/n);

    elseif abs(v(i)) >= 1.5 && abs(v(i)) < 2.5
        ryoubotb(i,1) = ryoubot(i,3);

    elseif abs(v(i)) >= 2.5 && abs(v(i)) < 3.2
        n = 30;
        ryoubotb(i,1) = 1/(1/ryoubot(i,3)^n + 1/ryoubot(i,4)^n)^(1/n);

    elseif abs(v(i)) >= 3.2 && abs(v(i)) < 3.5
        n = 20;
        ryoubotb(i,1) = 1/(1/ryoubot(i,4)^n + 1/ryoubot(i,5)^n)^(1/n);
    elseif abs(v(i)) >= 3.5 && abs(v(i)) < 4.59
        ryoubotb(i,1) = ryoubot(i,5);

    elseif abs(v(i)) >= 4.59
        ryoubotb(i,1) = ryoubot(i,6);

    end

    % A sign is added to the friction force value.
    ryoubotb(i,1) = ryoubotb(i,1)*sign(v(i));

    % Correction negative values (by overwriting).
    a4 = -0.003067;
    b4 = -1.606;
    c4 = 0.0007382;
    d4 = -1.966;

    a5 = -4.42e+005;
    b5 = 4.899;
    c5 = -0.05591;
    d5 = -0.1906;

    ryoubot(i,9) = a4*exp(b4*v(i)) + c4*exp(d4*v(i));
    ryoubot(i,10) = a5*exp(b5*v(i)) + c5*exp(d5*v(i));

    if v(i) < -2.6
        ryoubotb(i,1) = ryoubot(i,9);
    end
        if v(i) < -3.59
        ryoubotb(i,1) = ryoubot(i,10);
    end

end
```

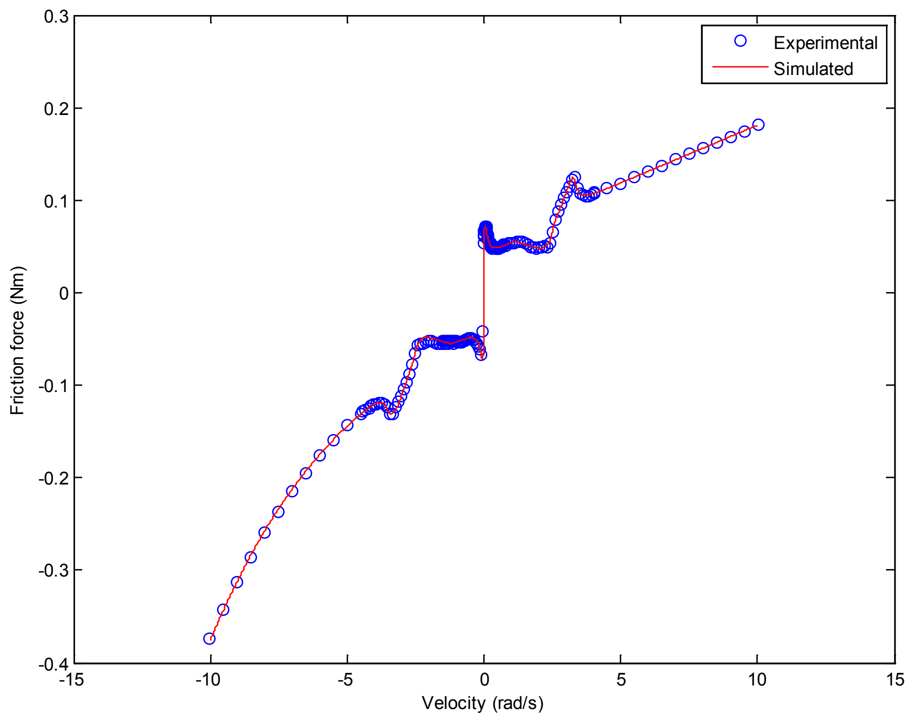


Figure 25: Experimental friction and fitted static model.

2.7 Applications

2.7.1 Bond graph motor design

The developed friction model was implemented in the available simulation control model by means of the bond graph methodology. Using a more realistic model of the youBot would improve the overall performance.

In the original simulation model the friction on the wheels was set to a simple constant value. Additionally, the wheels' motors were not modelled, so it made sense to implement an adequate model for the motor in combination with the new friction model.

Bond graphs allow one to represent physical dynamic systems in a multi domain layout. The arrows (bonds) represent bi-directional exchange of physical energy. This natural way of representation allows for a more intuitive and powerful depiction of a system. More information on bond graph can be found in [26].

In the previous implementation of the control model the "effort" from the energy bond coming out from the Interaction control block was being applied directly to the wheel. In real life this would be like putting energy on a wheel and magically move it. This can be better modelled by adding a motor block which receives this "effort" translated into a current signal (this is done by the block 1/K). Next can be seen the developed model for the motor:

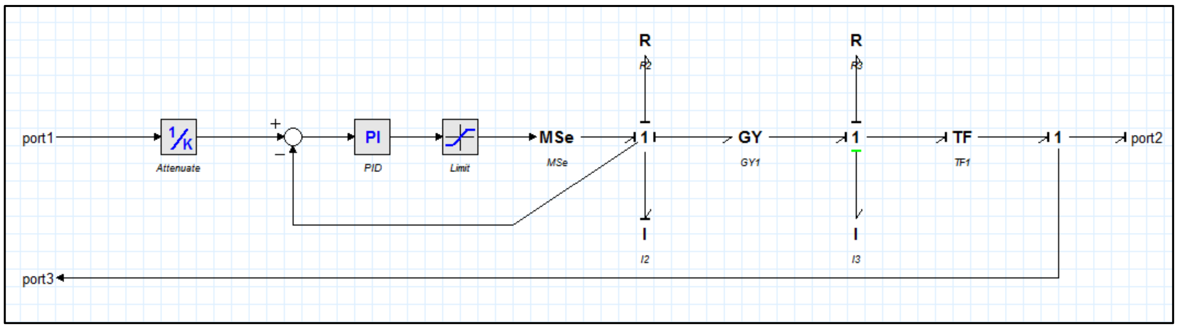


Figure 26: Bond graph model of a motor.

There is feedback current control through a PI and a voltage source (MSe) limited for safety reasons. Then the current is transformed into torque through the use of a gyrator (GY). This torque is then reduced by a transformer (TF) working as a gear block and fed to the wheel. The terminal resistance and inductance are included as well as the rotor inertia and a resistance to adjust the no-load speed of the motor. This last resistance should be removed from the motor model once the modelled friction is added.

The developed friction model can be connected to the rightest one port since this one represents the velocity on the wheel axis. It can be also put in the wheel block itself if attached to the same velocity port.

2.7.2 Friction compensation

In the actual setup the implemented control mode for the base was built using a proportional controller. It worked on the error between the commanded and real velocities on the virtual frame of the base (this is explained later in this page).

This implementation led to a poor performance control of the base because the velocities would vary in a highly dynamic way and also due to the fact that the velocity data itself that came from the youBot was quite noisy and of low reliability.

If the correct torque could be sent to the wheels their velocity would not be needed to control the base and this would result in an increase of the youBot's drivability. But when doing this the ideally calculated torque command would not be enough since there exist real life phenomena that dissipate energy through the mechanical chain. This would derive also in a situation in which the youBot would not move as demanded.

The phenomenon majorly responsible for these losses is friction. That is why the friction compensation could be added to the control cycle so as to counteract for this unwanted effect and to make possible the application of the above mentioned control scheme.

The friction compensation can be described with the block diagram of figure 27. The friction observer estimates an approximate value of the friction force from the current velocity of the wheel. This force is then added to the control signal with the opposite sign than that of the real friction, cancelling out each other.

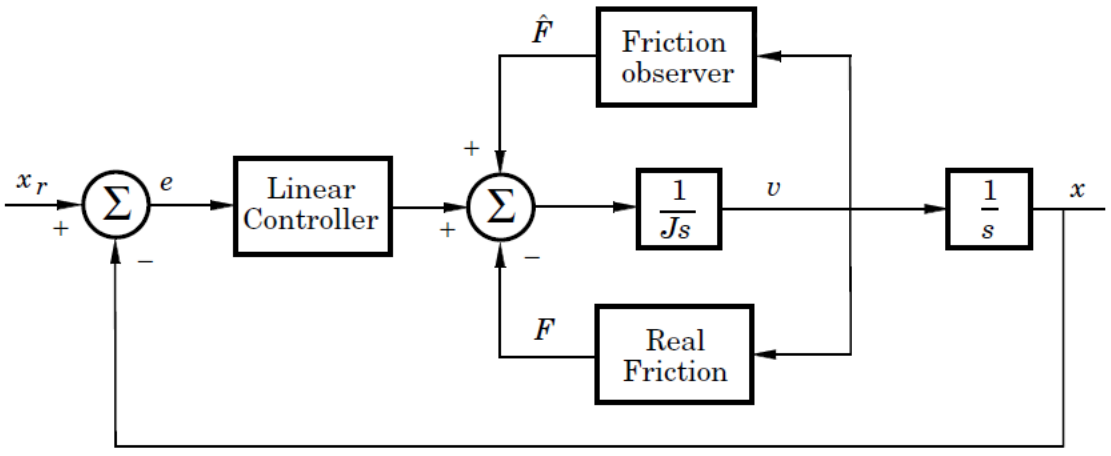


Figure 27: Friction compensation scheme (modified from [4]).

Before proceeding with the final application on the control model, a small experiment was carried out, based on [22]. The main idea was to apply an ideal signal to the system while compensating for the friction and see whether or not the output was also ideal or not.

The employed control law for the experiment was the following:

$$u = -K_p(x - x_r) + \hat{F}$$

where K_p is the proportional gain applied to the error between the actual position *x* and a reference position x_r, and $\hat{F}$ is the estimated friction to be compensated.

This experiment was done on the wheel number one (figure 28). The output should oscillate without decaying (friction subestimated) or getting unstable (friction overestimated).

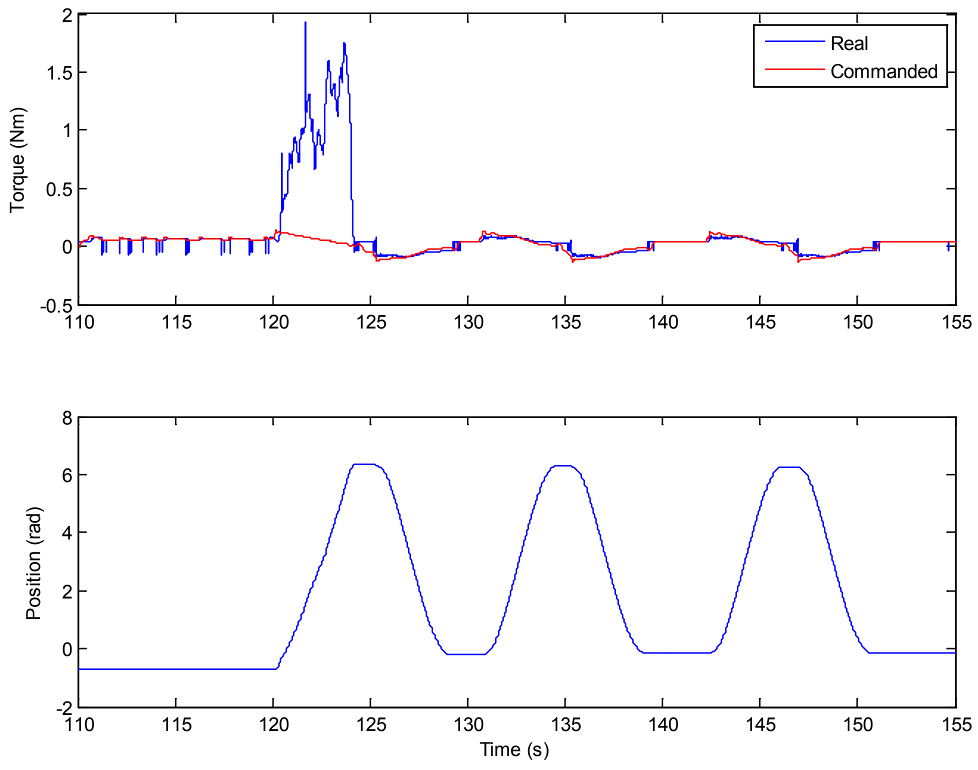


Figure 28: Obtained results with the friction compensation enabled in wheel 1.

In this test the first crest was manually driven to get the system engaged and then the wheel went back and forth almost three times. Without the friction compensation, after giving it a first push the wheel would return to its original position and stay there. So it does help the wheels to overcome the friction effects. Here the reference position was 2 rad and it can be clearly seen that the maximum position reached goes over 4 rad. This is because an extra constant torque of about 0.01 Nm was added to the motor current input signal when playing a bit with the experiment parameters. However, it is not an objective to achieve a perfect oscillatory system and since the estimated friction is added to movements in both directions it does not get unstable.

Since the friction observer was already built the only thing left was to adapt the control model for this purpose. However, before continuing it is necessary to explain the Link Map.

The purpose of the Link Map is to map the angular velocities of all four mecanum wheels into one virtual linear velocity attached to the base frame. It was already implemented when this project started but it required some adjustments for it to function as desired. It is based on the sum of the individual forces that each wheel imposes on the centre of mass (CoM) when it is rotating in one or another direction. Next can be seen a drawing for better understanding:

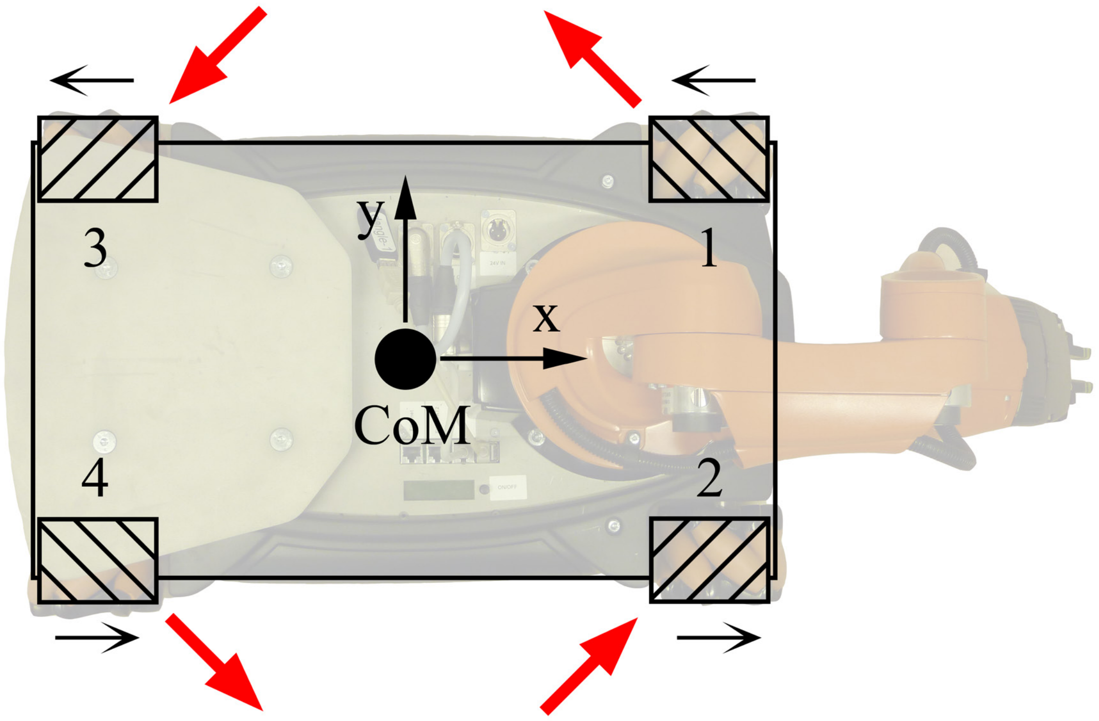


Figure 29: Forces (red) acting on the youBot CoM for positive rotations of the wheels (black).

Here the black arrows represent the positive velocities of the wheels while the red arrows are the force that each one of them exerts on the base frame when rotating in this direction. It can be seen how the direction of the gridded areas is the opposite than the one of the wheels. This is because the position of the rollers that matters is the one in which there is contact with the ground. So the rollers' direction shown is how it looks from an under view.

Taking the frame origin near the CoM (depicted in an approximate position) the combination of these forces will drive the youBot in one direction or another:

$$\begin{bmatrix} Yaw \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

where the Yaw is the angular velocity around the vertical z axis (positive in counter clockwise direction) and w_i are the angular velocities of the wheels.

The control model sends a wrench composed from a moment around the z axis and the two force components in the x and y direction. Using the Link Map these are translated into independent torques for each wheel. Then the friction is estimated and summed up to the outgoing signal. The latter is also done individually for all four components corresponding to the different wheels.

As just described the output part of the model has the following distribution:

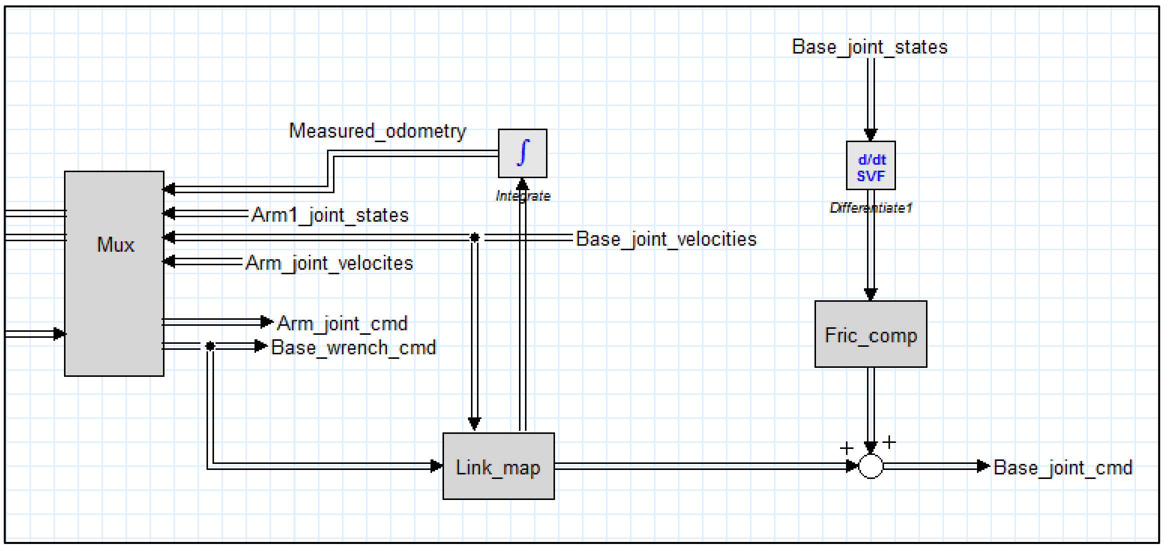


Figure 30: Friction compensation implementation on the youBot control model.

It can be seen how the velocities used for the *Fric_comp* block are obtained by applying a SVF (State Variable Filter) to the position data. The *Link_map* is also used to convert the *Base_joint_velocities* vector into a twist vector which is later integrated, giving as a result the base odometry used in the control loop.

Finally, the friction compensation block was included into the control model for the youBot and several tests were performed. No quantitative assessment was performed because the real friction on the youBot cannot be measured. As a matter of fact, it was not possible to compare the obtained simulated friction with any real value. However the control was significantly improved and for instance, when the friction compensation was enabled, the robot did move as commanded, in contrast with the previous control version which was not able to induce any starting motion on the youBot base.

Using the new friction compensation component the robot's base movement was a bit slow but this is something that could be solved with the proper tuning. However, this was not possible to be carried out in the present thesis because of lack of time.

2.8 Conclusions

In this first part of the project a friction model has been developed for the motor blocks placed at the youBot base. Each motor block has been experimentally tested. Then the experimental results have been studied and a customized friction curve dependent on the velocity has been developed for each wheel.

Friction phenomena and its effects on mechanical systems have been revisited and some of the most used friction models have been analysed in terms of their suitability to reproduce the experimental results. The parameter identification and validation procedure for a model and the different options available for doing so have also been discussed. It has been found too that it is very difficult to match theory with experimental results. None of the studied models have been able to satisfactorily describe the obtained experimental results. Therefore, the experimental friction-velocity curve has been directly implemented into the newly created motor model.

The developed friction model has improved the simulation of the dynamics of the system, and it has also been used to improve the control of the youBot base through friction compensation for its motor blocks. The final results have successfully proven that the friction model is useful for compensating the friction at the wheels' axes.

3 Haptic interface

3.1 Introduction

People may not be aware of it but haptic interfaces are here and they have come to stay for a long time. But before giving some examples it is better to address a common question: what is haptics?

A normal definition for haptics is everything that refers to touch or touch-related capabilities. However, for this project is more adequate the definition of haptic technology given by the International Society for Haptics (http://www.isfh.org): '*Haptic technology, or haptics, is a tactile feedback technology which takes advantage of the sense of touch by applying forces, vibrations, or motions to the user*'.

As every technology, haptic interfaces did not start as they are known nowadays. They come from scientific progress built over knowledge, seeking new and better possibilities for interfacing humans with robots.

With the growing of the industry many hazardous tasks appeared which needed to be done from the distance. Safety reasons were the initial motivation but it quickly became clear the convenience of decoupling the actuated part from the actuating part. In the following a very short of the highlights of the haptics history is given. The information has been extracted from [27] where it can be found in more detail.

The first haptic devices –starting in the 1950s– where known as Bilateral Master-Slave Manipulators (MSMs). They were composed of a master and a slave of similar complexion and characteristics. The main difference was that the master was the one equipped with a gripper or handle and some kind of triggering mechanism with which, the human operator could actuate the slave. Both parts would be connected using simple mechanical systems like chains, cables or other electromechanical components. This low complexity resulted in a more straightforward application but it also reduced the capabilities of such a device, e.g. the load had to be managed using the operator limited force input.

As their popularity increased the range of applications broadened. Two new terms appeared with this spreading: (haptic) Teleoperation and Telepresence. The former refers to the possibility for someone to interact with a remote location using its own senses. The latter goes one step forward and could be ideally depicted as an immersive teleoperation in which the user would feel the same as if he or she were at the remote site.

These two concepts, as most of the times, came from science-fiction and pulled science into the emergence of haptic feedback developments. Another breakthrough was the idea of Virtual Reality. In VR the operator interacts with an artificial environment created by a computer rather than with a physical place.

In the 1980s, a new concept known as *Mini-masters* appeared. As opposite to the MSMs, the master end did not have to be precisely equal to the slave in structural terms. They were small master controllers mainstreamed in VR areas such as molecular modelling or Nano-scale surface simulations.

Later appeared the Servomanipulators. The cable connections gave them the advantage of mobility over the MSMs and since they used servos depending on the necessity, they could handle large loads with ease. The DC servos –at the beginning they were AC-driven– were disposed in both ends so as to have force feedback. In this way, the force feedback ratio (known as *force boost*) could be chosen as desired. Another improvement was the size reduction which made feasible the application of more compact designs.

One very interesting application of haptics was the Exoskeletons. The goal was to power up the human strength-related capabilities. This was achieved by direct human slaving or, in other words, making the human play both roles (master and slave). Although in the beginning they were big and bulky they accomplished their mission and let a normal human lift and move high loads, something that otherwise would have been impossible. However they had some electro-mechanical imperfections and a limited workspace which could result in possible safety hazards for the operator.

Despite this, due to the rapid introduction of the VR into the sector, the exoskeletons were adapted for low-cost body movement registering. Naturally, they would feed haptic information to the immersed user as well, renewing the interest in this type of body systems.

So it can be seen how the early manipulators and telerobots –which were nothing more than remote handling systems for the nuclear, subsea, space and military markets– evolved into the telepresence with the haptic interfaces and the Virtual Reality.

The manufacturing industries promoted this rapid growth being able to provide progressive miniaturization while decreasing production costs. This has made the haptic interfaces more accessible to the scientific communities, bringing together resources and research potential for what is to be the haptics take off.

3.1.1 Modern haptics

This small section attempts to cover the actual state of the haptics sector, its advances and the most straightforward applications. Most of the information has been extracted from [28], [29] and [30] and it is presented in a generic form, because this subject is quite extensive and cannot be fully addressed in this project.

After many years of studying haptic interfaces the research is still centred on how to feel, manipulate and explore three dimensional objects and their different features: shape, weight, surface texture, temperature, etc. All of this information can be transferred through the touch sense into the human brain, enriching the different tasks one may do.

Now there are two main approaches to haptic interfaces, those which focus on the force feedback and those which bring more attention to the tactile side of touch. Some interfaces may need just one of the above but it is sure that both combined will give the more realistic results.

Apart from the psychophysical studies on human haptics, most of the research is being done in the area of haptics rendering. Here are included the acquisition of models, the collision detection, the latency and the force feedback among others. It is very interesting as well those studies which have as objective the capture, storage and later retrieval/reproduction of haptic data.

On the mechanical part, now there is a wide range of high performance stylus-based masters (or equivalent grasping object) as well as haptic gloves. The last allow stimulating the different sections of a hand as the fingertips, fingers or the palm, which is essential for the grasping of virtual objects. There exist also Thermal gloves which display temperatures on the hand allowing not only to experience the environment temperature, but also that of the objects which are being virtually grasped.

Some effort is being delivered also to the creation of Microelectromechanical systems (MEMS), which consist on arrays of miniactuators. These apply small force vectors to the hands or other body region creating a smoother and continuous pressure rather than a single focalized one as usual. In relation with the previous system are those which apply vibrations. They are useful for recreating different surface roughness or as an extra channel for delivering system events.

A very useful yet simple idea are the two dimensional haptic screens. Mostly developed for the visually impaired, they give them the capability to explore an image through the touch sense. There are also some haptic screens which can deliver three dimensional stimuli.

Modern applications:

In the medical sector haptics research is focused on surgical simulation and medical training. Some examples are the simulation for palpation of subsurface liver tumours or for bone marrow harvest and training of needles insertion. Key is how to model and simulate tissue as realistic as possible.

Another interesting application is for museums displays. Due to their usual “hands-off” policies there is much information lost when one can only see the art collections. By digitalizing three dimensional models of sculptures and decorative arts the public can fuller appreciate these through haptic technologies. Some ideas include interactive exploration where more than one person examine the piece or the reproduction of high detailed surfaces with indents, textures, etc.

Continuing with the arts, haptic devices can be used as well as an input for painting, sculpting and CAD related applications. Different pressures may imply ampler brush stroke or higher impact of the chisel on the stone. Together with CAD one can intuitively feel the assembly or disassembly of mechanical objects.

Haptics are also a great way of visualizing scientific phenomena or and reproduce three dimensional models physically. Examples of this are the simulation of soils with different layers and complexities, exploration of cities or subsea terrain or for visualizing chemical and biological molecules/structures.

Military or research can make use of haptics as well. They provide an alternate information path with a low-bandwidth. The information to transmit could be the proximity to an objective, directions to it or warnings of threads. It is most useful in scenarios where the other senses are limited, like sand storms, in space or for when things happen too fast, like extreme sports.

Some interaction techniques regard haptics as a form of expanding an interface’s information bandwidth. Usually most interfaces depend solely on vision and hearing and this can saturate these channels. Haptic devices can be helpful for re-balancing these situations. The dragging, sliding or pressing of buttons can be accompanied with haptic feedback. This applies for most of the virtual environments and in games it has been utilized for many years now.

In relation with the last lines are also the assistive technologies. The drawbacks of having pure visual and sonorous interfaces are even more accentuated for impaired people. As an example, GUIs (Graphical User Interface) can be adapted to guide the user along them and avoid unwanted incidents. Using the hand for exploring virtual objects can be also achieved by having haptic devices with multiple contact points, as explained at the beginning.

3.2 Haptic control

The following chapter is a brief introduction to the world of haptics and their role in the teleoperation segment. This introductory chapter has mainly been extracted from [31].

In a haptic control basic scheme five components are usually found: the human or operator, the master, the controller, the slave and the environment. The slave can exist physically giving place to a standard haptic teleoperation scheme or it can be part of a VR system inside a simulated environment.

Regardless of the end goal, there is a common objective. As explained before, the operator should feel like he was actually realizing the task himself. In order to accomplish this, basically it is necessary to compensate for the master mass and other inertial components so it does not interfere in the operator's sensing of the environment. Additionally, the master-slave connection should be infinitely stiff so as to transfer all the information (forces, displacements) between them without any additional distortion or lag.

In figure 31 can be seen a scheme of a haptic teleoperator. The dotted lines indicate which parts of the setup should not be felt by the human operator.

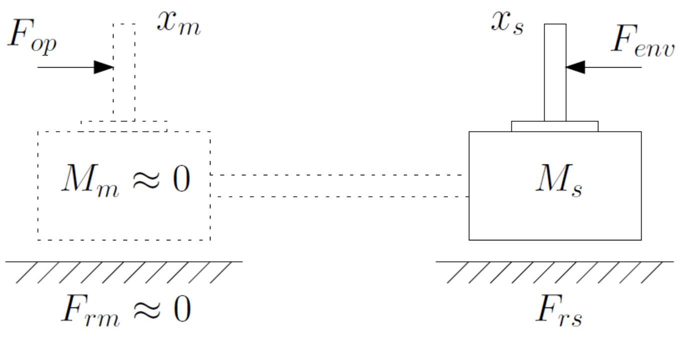
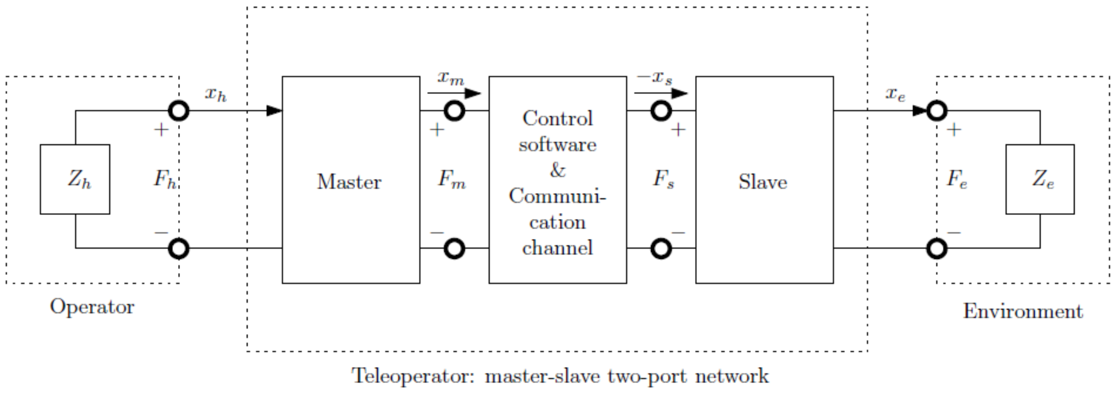


Figure 31: Ideal haptic teleoperator (from [31]).

3.2.1 System representation

In the majority of cases a teleoperation system is laid out using two-port models. These come from the network theory developed on electrical circuits. In figure 32 a basic haptic teleoperation system has been depicted using two-port models for the master, the controller and the slave:



x_h
x_m
$-x_s$
x_e
Z_h
F_h
Master
F_m
Control software & Communication channel
F_s
Slave
F_e
Z_e
Operator
Environment
Teleoperator: master-slave two-port network

Figure 32: Two-port model representation of a haptic teleoperator (from [31]).

The configuration of the two-port models will define the overall system behaviour. The relation between the inputs and outputs, that is, their causality, can be chosen as needed depending on the actual physical implementation of each block. These are easily obtained studying the mechanical structure of the setup (usually for the master and the slave) and or from the used model during simulation (usually for the controller).

Also, in the above figure it can be noticed how the position is used instead of the velocity. It would be natural to use the velocity together with the force since the port power would be directly known from the terminal variables. Despite this, this notation introduces a pole/zero pair at the origin causing problems in stability analysis conditions. Furthermore, usually the position is the data which can be directly obtained from the encoders as opposed to velocity.

The two-port mathematical expression that defines the relation between these variables is a 2 by 2 square matrix. The models which are used more often to represent this matrix are referred to as *z-parameters*, *y-parameters*, *h-parameters*, *g-parameters*, and *ABCD-parameters*.

From these the most popular when representing a haptic teleoperation system are the hybrid parameters (*h-parameters*). It is the one used in the previous representation of a haptic teleoperator (figure 32) and the relations that it establishes can be seen below:

$$\begin{bmatrix} F_h \\ x_e \end{bmatrix} = \begin{bmatrix} & H & \end{bmatrix} \begin{bmatrix} x_h \\ F_e \end{bmatrix}, \text{ with } H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} = \begin{bmatrix} \left.\frac{F_h}{x_h}\right|_{F_e=0} & \left.\frac{F_h}{F_e}\right|_{x_h=0} \\ \left.\frac{x_e}{x_h}\right|_{F_e=0} & \left.\frac{x_e}{F_e}\right|_{x_h=0} \end{bmatrix} =$$

$$= \begin{bmatrix} \text{input impedance} & \text{force scale} \\ \text{position scale} & \text{output admittance} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}_{ideal}$$

Despite being the *h-parameters* the most used, the *z-parameters* are also utilized in the field and other representations could be used if properly applied.

Additionally, instead of using a teleoperator with the combination master-controller-slave that connects operator and environment, other possibilities exist. For example, the operator could be represented together with the master and communicate via the controller with another block composed by slave and environment.

3.2.2 Haptic teleoperation system components

According to what was previously explained the four different components needed for the construction of a haptic teleoperation system are the operator, the environment, and the master/slave couple in between them.

Master/slave actuators

In the majority of cases the block corresponding to these two elements is defined using either an impedance or an admittance model. Since this type of model is not capable of describing many physical phenomena usually encountered in the real system, their inclusion implies the use of other controllers so as to cancel all the existent non-linearities and/or disturbances. Moreover, for this configuration to work properly it is needed that all compensators (friction and gravity among others) accomplish their task perfectly.

The application of an impedance/admittance model to the real system generally involves the use of estimation methods, usually online, for the correct finding of the physical damping parameters. This seeks an optimization from the performance point of view because what may work seamlessly in simulation usually needs further development in a real application.

Another aspect to remark is that when the master and the slave actuators have similar physical structures a higher grade of performance can be achieved. However, it is strange to encounter identical master-slave manipulators and in most of the cases, unpractical. This is mainly due to the fact that they both have to adapt to different mediums. While the master has to provide the operator with an easy interface for control purposes, the slave has to interact with a specific environment and so it needs to be prepared for those special circumstances.

Operator

When modelling a haptic teleoperation system the operator is usually defined using an impedance model as well. This selection is of great relevance because it implies that the human is a passive element of the system. However, it is common sense that a human is not a passive agent most of the times. Even less in this case, when he takes the role of operator, that is, the one who commands, which is an active action by definition.

Yet even if the operator is not passive at all, he can usually be regarded as a stabilizing agent rather than a destabilizing one. So, from here two different approaches are exposed over the vast literature:

- Passive operator: The operator does not manipulate the system seeking its own objective. He instead just contributes to the overall dynamics of the system with a stabilizing role.
- Interactive operator: The operator manipulates the master device providing input signals based on his goal and perceived info about the state of the system.

The majority of times the passive operator model is the one utilized. It can be a simple impedance model but more complex schemes are also used like second order mass-damping-stiffness systems. However, this type of model requires some hand-tuning afterwards in order to increase their performance.

The second kind requires of more complicated techniques to integrate the operator's inputs. Some models regard the human input signals as disturbances with upper and lower bounds. This is an attempt to consider the wide range open-loop human behaviour. Other models go even further and try to predict the human input from his current actions and the actual state of the environment.

Environment

The working environment can be modelled similar to that of the operator. The main difference being that the environment is not a controlled element and as such it can be either a stabilizing or a destabilizing agent. Sometimes the forces acting on the slave actuators may be bringing it to a stable situation while in others they may just oppose to it. This complicates somehow the prediction of its interaction with the slave end. Despite this, some models try to estimate the environment, making it easier to track its behaviour on the master.

3.2.3 Performance measures

The growth of haptic systems and their derivatives demands certain side developments. Since they are a new kind of devices and setups, they need to be dealt with and evaluated in different ways. The performance assessment of a system with these characteristics has to be done according to some standards in order to facilitate its comparison and further classification. This matter has attracted a considerable interest; see for example [31] and [32]. Next, some performance measures for the complete haptic teleoperation system are extracted from [31]. They are followed by a collection of simple design properties appropriate for the evaluation of a haptic device.

3.2.3.1 Quantitative measures

These parameters are specific for two-port representation mentioned above so they are linear performance measures. Therefore, the numerical values can be calculated from the linear system model (the H-matrix). These formulae will not be written here since they can be found in [31]).

Tracking

It is the natural way of evaluating a teleoperator. As said before, in an ideal haptic teleoperation system the positions and forces on both sides should be equal. This parameter is the result of comparing these values and taking them relative to the nominal master position or force. Position tracking is done with the slave side moving in free air ($F_e = 0$), and force tracking when the slave side is in contact with a hard surface ($x_e = 0$).

Bandwidth

The bandwidth is a parameter that evaluates the information transferring between the implicated parts, the operator and the environment. While the humans do not command positions and forces at a high frequency, they rather do it on the order of 0-10 Hz, hard environments at the slave side can present frequencies of up to 1000 Hz. This difference is quite representative and in order to diminish the loss of information the bandwidth should be set as high as possible.

The bandwidth comes from the tracking error and it is the frequency at which this error is -3 dB $\left(\frac{1}{\sqrt{2}}\right)$ below the low frequency gain. It is important to remark that it does not give any information on delays or the phase lag.

Another type of bandwidth is the one related with the information itself rather than with the transferring process. This performance measure is the one that compares the position and force ranges at each side of the haptic teleoperator, i.e. the matching of both workspaces.

Scaling Product

The scaling is a parameter defined in the haptic teleoperation scheme that represents the ratio applied when translating positions and forces from the master to the slave or vice versa. Both of these ratios can be easily calculated from the H-matrix, and their product is called the scaling product.

The scaling is often assumed to take the value of one but it can be chosen at will. For example, if a human is controlling a haptic teleoperator that deals with high loads in the slave side it would be advisable to use a diminishing ratio when translating forces into the master. This makes the felt environment less accurate but increases stability due to artificial energy loss.

Impedance

An important feature of a haptic teleoperator is the sensation transferred to the human controlling it, that is, how heavy/damped/springy it feels when the slave robot is in free air or in contact with different environments (especially during highly stiff contact). Usually this supervised impedance at the master is used as another performance measure. The felt master impedance can be obtained from the device properties (H-matrix) and the remote environment impedance.

Transparency

Once the felt impedance at the master has been obtained, it can be compared to the remote impedance to see in which measure the reality/environment is deformed. In principle, the felt master impedance should be the same as the remote impedance for every type of environment. This would mean that the operator would feel exactly the same as if he was actually doing the task. The transparency is defined as the quotient between these two impedances (admittances can also be used for this purpose). In the ideal case just explained the transparency would be unity for all frequencies; a perfect impedance match.

In general the transparency depends on both the teleoperator properties (H-matrix) and the remote impedance.

Fidelity

This performance measure is similar to transparency but expressed in a different form. Instead of just calculating the quotient, it differentiates the felt impedance at the master with respect to the remote impedance. This parameter goes beyond the direct relation between the impedances and tries to put numerical values to how changes in the remote impedance are felt at the master side by the operator. As it says in [31]: “Optimization for fidelity aims at improving the discrimination of impedances”.

3.2.3.2 Qualitative measures

Despite the fact that qualitative measures do not help as much into concretizing and specifying haptic teleoperators, they are used frequently because of their easy implementation. Usually the most used way of representing the performance of a haptic teleoperation scheme is time-plots of either the positions or the forces, or both. Comparing the specified trajectory and the one generated by the controller both position and force tracking can be evaluated.

The best tasks for this kind of experiments are the ones that are typical of the application for which the haptic teleoperation system will be used. These tasks can then be compared using specific measurements (completion time, maximum force usage, sum of square forces among others) or by how different subjects experience various simulated environments (qualify how much the real and simulated environments are alike).

In this subject it is important to use simple yet defining features in order to qualify a given teleoperation system. Sometimes it can be even helpful to use subjects that are not related to the application field neither with the teleoperation concept, since this lack of knowledge gives them a different perspective on the evaluation of the test tasks.

3.2.3.3 Haptic design

Since the linear model is a simplification of real systems its behaviour will not be exactly that of the modelled systems. There exist certain imperfections and nonlinearities that are device specific and determine the range of validity for the linear model. It is of remarkable importance to design a haptic device with the targeted teleoperation system in mind and vice versa because they both are highly correlated and should not be thought of separately. A short collection of the most important performance aspects on haptic design is presented in the following and briefly discussed.

Backlash: The total backlash accumulated over a teleoperation system is usually not constant over the working area. Since the application of some backlash reduction methods may increase other unwanted effects (like friction) a balance needs to be found.

Cost: The total cost of a teleoperator can be in a wide range of budgets. In general, the higher is the goal for performance the higher will be the price. Therefore, a compromise is always necessary while focusing on the required application.

Dexterity: It represents the 6 degrees-of-freedom workspace available over the working envelope of the system. It can be reduced by joint clashes, limits and singularities of the system.

Peak force and force thresholds: The peak force should be specified at the intended device body interface. It is also important to define the long term peak force because for long durations possible heat losses and their dissipation will lead to a thermal equilibrium, which has to be below an acceptable limit. The force thresholds are mainly two; the force needed to overcome stiction at the master and produce the first motion at the slave; and the maximal force needed to assure the correct realization of the application (while avoiding saturation).

Gravitational balance: The better balance a haptic teleoperator has the better the operator's experience will be (less fatigue). This mainly depends on the device structure and in most of the cases gravity compensation is needed. It can be achieved either using mechanical counterbalancing or with a specific controller. The former is passive and simpler but adds mass and inertia while the latter overcomes these but consuming more energy (more powerful motors and more complex controller).

Inertial effects: From the three main inertia sources felt by the operator, the master and slave manipulators inertia are the ones that need to be reduced. When doing this only a fraction of the load inertia is left and therefore the operator fatigue is reduced.

Maximum velocities, accelerations and force/load capability: These values should correspond to the range defined by the human operator and the environment. Since contacts and shocks are characterised by rapid changes in velocity, peak accelerations are of great importance so as to bring realistic movement at the master. The maximum velocities and accelerations are dependent on the high frequency behaviour.

Device-body interface, operator accessibility and operator workload: Since the hand or other body parts must be connected to the device in order to control it, three different cases of body interface can be found: the hand holds the device, the device is attached to the body, or the interaction is unilateral. The interface ergonomics is very important when reducing the operator fatigue, defining the operator's accessibility. These interaction conditions will give place as well to a specific operator workload. This includes the visual, mental and physical loading components that affect the operator fatigue and the consequent error rates, making necessary the setting of a maximum operation time.

Motion range and volume of operation: The motion range poses several problems due to the possible lack of invariance and couplings. For those devices with a high number of degrees of freedom, the combinations of translations and orientations makes the specification complicated. The volume of operation has to be kept, at the same time, small so as to reduce operator's fatigue and safety risk. This has to be accomplished without affecting too much the master and slave workspaces matching, or it will be necessary to include a rate controller to facilitate the application realization.

Other properties of haptic devices which may affect their performance are the design for backdriveability and the possible cross coupling effects. The encoders can also add noise to the control output; therefore their resolution needs to be high so the human does not feel any annoying vibrations.

It is necessary to remind that the operator safety has to be of high priority in all aspects during a teleoperator design. The operator is part of a teleoperation system and has to be kept in good functioning conditions so as to realize the desired application correctly!

3.2.3.4 Human factors for design specification

In this section some measures of human factors that can be used to specify a haptic device interface will be described. This is a critical step for the proper design specification of both hardware and software and requires a basic understanding of the biomechanical, sensorimotor and cognitive abilities of the human haptic system. [33] includes a review of the most useful features for defining the ranges in which a haptic teleoperator should operate for the case of a device-to-hand/arm interaction. A brief summary is included in the following.

Force sensing

Slowly-varying forces: The force display resolution at the master side should be equal or higher than the human sensing resolution. This will make the human feel that the forces displayed vary in a smooth gradual way. It is known from literature that the Just-Noticeable-Difference (JND) for human force sensing is around 7% regardless of test conditions.

Vibration: unintended vibrations are one of the most noticeable disturbances in force feedback teleoperators. If the human operator detects a significant level of vibration, he will disconnect from the immersive state in which he was realizing the task. In the Literature this threshold (detection of vibrotactile stimulation) is usually estimated as 28 dB below 30 Hz and it decreases at -12 dB/oct from 30 to 300 Hz before increasing again. This can be considered as a very tight constraint on the device and of fundamental importance for the teleoperator success.

Pressure perception

When the master actuator is designed to be attached to the operator’s body, the grounding location and the contact geometry have to be chosen carefully so as to create an illusion of a true earth ground. This one might be successfully produced if the pressure distribution and its effects on the grounding location are below the absolute detection and discrimination thresholds, respectively, for the human operator. Minimizing the contact area while increasing its perimeter might be a way of improving the illusion of true grounding.

Position sensing resolution

The position resolution of the human operator can be used to restrict the position sensing resolution of the device. For example, the JND of the Proximal-InterPhalangeal (PIP) and the MetaCarpalPhalangeal (MCP) joint is of about 2.5° (in relation to the fingertip position). In humans joint angle resolution is better at proximal joints than at distal joints.

Stiffness

Despite the kind of environment used –real or simulated– the different solid objects that are (or are supposed to be) rigid will be translated into a certain force display accompanied by displacement at the master side. Since rigidity is a perceptual notion, the following question arises: What is the stiffness required to convince the operator that an object is rigid? The stiffness perception studies of [34] revealed that a minimum stiffness of 242 N/cm was required to simulate a hard wall. This can be, however, diminished in virtual reality with the help of coordinated visual or sound effects.

Human Force Control

Range: To match human capabilities, the peak force on the master actuator should meet or exceed the maximum force humans can produce. This maximum controllable force varies from 16.5 to 102.3 N and increases from the most distal joint (PIP) to the most proximal joint (shoulder).

Resolution: The resolution at which the force/torque on a joint linkage must be controllable to at least the level at which humans can sense and control force in order to present a perceptually smoothly varying force. The force control resolution decreases from 1.96% to 0.87% from PIP to shoulder joints and its absolute value is of about 0.36 N, which means that humans have better control over force output with the shoulder joint than with the finger joints.

Bandwidth: The force control and perceptual bandwidths of a human operator are quite different. The human somatosensory system can perceive vibrotactile stimuli up to 1000 Hz while the upper bound of force control bandwidth is on the order of 20 to 30 Hz. The master device bandwidth when operated should at least match the force control bandwidth of the operator.

3.3 Problem statement

The main objective of this part of the present MSc thesis was to create a working interface so the available haptic device (Omega6) and the youBot could work together.

This haptic device comes with its own SDK (Software Development Kit) from Force Dimension. It provides an accessing point to the device so as to read positions and display forces.

The Omega6 SDK has been used to create a ROS package (done by Rueesch Andreas) which can be used to build part of the haptic interface. This open code package connects to the haptic device and allows the user to read from the incorporated sensors as well as to display forces along the translational degrees of freedom. The most important sensor information is the position and orientation of the Omega6 pen tip. Besides it, other states of the haptic device can be observed like the forces display status or whether an auxiliary button is On or Off.

In order to command a certain position and orientation to the youBot the Omega6 pen tip can be mapped to the youBot tooltip. The haptic state can then be sent to the youBot as the next setpoint to move to. The already developed control model uses the difference between the current position and the commanded setpoint to promote an attractive force between them in order to move the robot (the same is done with the orientations). This is implemented in such a way that all the joints (including the base) act simultaneously in a coordinated way in order to move the youBot tooltip to the desired position.

However, the above covers only one way communication and, as explained in the introductory chapters, a haptic teleoperator should be able also to transmit to the operator the force that the remote environment is imposing on the slave actuator.

Since the youBot does not have force sensors at the tooltip the environment impedance cannot be measured directly. One option to overcome this problem would be to calculate the tooltip force from the individual torques of each joint of the robot. However, this other option is neither a valid approach in this case because, as stated before, the current sensors on the youBot motors only provide the absolute value of the actual measurement. The direction of the torques is thus required to proceed with the above alternative. At the moment is being studied the possibility of using the joint velocity direction to solve this but sadly it is still not completely functional.

The most natural way of proceeding with these limitations is to use the same position and orientation error used to drive the youBot –but with opposite sign– on the haptic device. This force will move the Omega6 pen tip to the current (mapped) position of the youBot's tooltip. The overall feeling will be as if one were pulling/dragging the youBot's tooltip around. This is indeed a position error control architecture and only requires the positions of both master and slave to work properly. In fact, the displayed force at the master has not been used as a control parameter for neither the master itself nor the slave (youBot).

In the following schematic (figure 33), the forces that drive the Omega6 (master) and the youBot (slave) using the same position (and orientation) difference are shown.

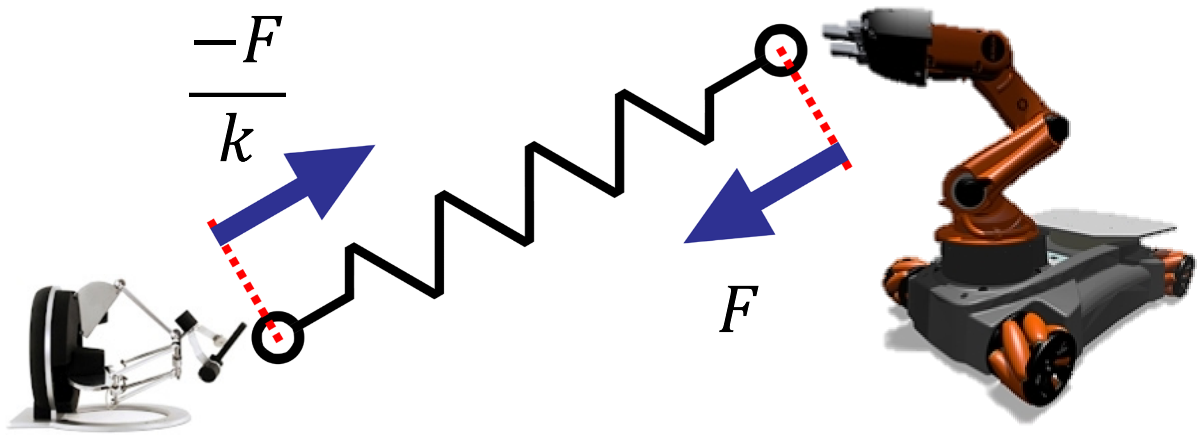


Figure 33: Mechanism analogy for the position control used in this work.

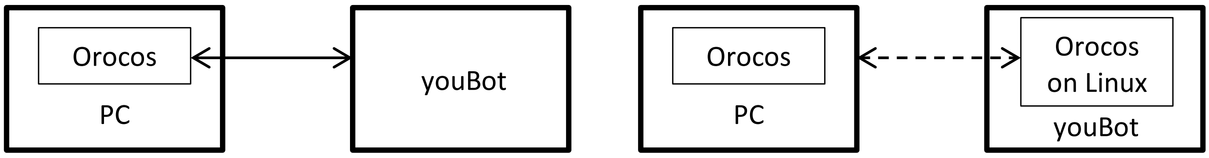
3.4 Implemented solution

3.4.1 Interface development

3.4.1.1 ROS – Orocos interface

Since the youBot is a mobile platform it has its own independent energy source. Therefore, this source is also available to control the robot wirelessly. The youBot inner mainframe works with Linux OS and is able to run Orocos in real-time. This software works as an interface with the drivers of the robot, allowing the control of the actuators and the reading of the different available sensors.

Although this is the targeted operation mode (wireless), during the development of this part of the thesis, the Orocos framework was executed on a PC station which communicated with the youBot through an Ethercat connection. In this way, the youBot could also use the building power supply for both commodity and safety reasons. It is easier to work with this setup and everything which has been developed can later be migrated to the youBot without any further complications. Figure 34 schematically depicts the two possible operation modes:



b) Control at the PC.

a) Control at the youBot.

Figure 34: Used operation modes of the youBot.

Since the available software for the Omega6 haptic device is a ROS package, there are some bridging components that are necessary for the communication between it and Orocos. Specifically, the orocos_toolchain_ros was used in this case. This stack of packages provides the option of creating an Orocos specific message type (Orocos typekit) from a ROS message. If a topic with this message type is published, the Orocos component on the youBot can subscribe to it and read the transmitted data. It also allows for an Orocos component to stream data from an output port to a ROS topic, covering the communications between the two platforms in both directions.

In order to assure the well-functioning of the haptic device with the ROS package and its connection to the youBot, two Orocos components (1, 2) were created to work as a bridge between these two. Instead of using the youBot for this first test, only one component of the control model was utilized. This was the Cartesian Space Stiffness, which is the one responsible of calculating the forces that actuate the youBot depending on the position error.

Since the main purpose is to evaluate the connection performance these components tasks were simple. The *component 1* was in charge of receiving the haptic device state from a ROS node, transforming it to a suitable message type, and emitting it to the Cartesian Space Stiffness Orocos component. In the opposite direction *component 2* would do the same but with the force data that had to be displayed in the haptic device. The scheme layout is shown in figure 35.

The Cartesian Space Stiffness was given a constant virtual setpoint located at the Omega6 system coordinates origin. In this way the Omega6 pen tip motion would always converge to a stable equilibrium located at this point.

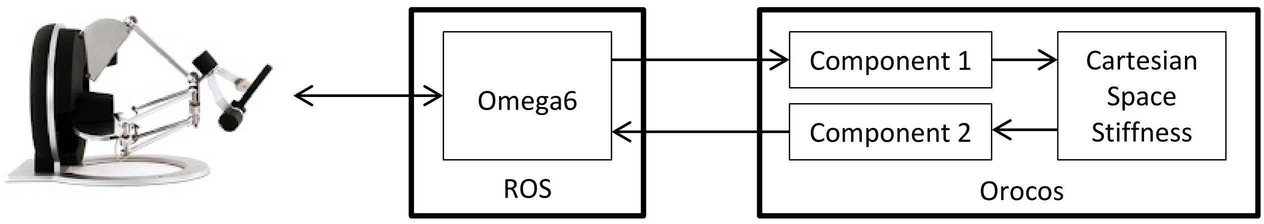


Figure 35: Communications between the Omega6 and Orocos through ROS.

In the next paragraphs the most important parts of the code will be explained for the two Orocos components created.

Component 1

The first component, *component 1* had to convert from the haptic state message type to a 4 by 4 homogeneous matrix.

Both the input and output variables are created:

```
hapticteleop_msgs::HapticState hp_state;
std_msgs::Float64MultiArray homogeneous;
```

Then in the *configureHook()* a sample message is created and sent through the output port. This ensures that all the following messages will be sent in hard real-time. The actual message data is also resized.

```
bool configureHook() {

    std_msgs::Float64MultiArray sample;
    sample.data.resize(16, 0.0);

    outPort.setDataSample(sample);

    homogeneous.data.resize(16, 0.0);

    return true;
}
```

In the *updateHook()* each time that a new haptic state message arrives, an homogeneous matrix is created and sent to the Cartesian Space Stiffness component. It can be observed how for this first test the angles were set to zero since the haptic device does not have the rotations actuated.

```
void updateHook() {

    TaskContext::updateHook();
    if(inPort.read(hp_state) == NewData)
    {
        pos_x = -hp_state.position.x;
        pos_y = -hp_state.position.y;
        pos_z = hp_state.position.z;

        ang_y = 0.0; // -hp_state.orientation.x;
        ang_x = 0.0; // -hp_state.orientation.y;
        ang_z = 0.0; // hp_state.orientation.z;

        double cos_x = cos(ang_x);
        double cos_y = cos(ang_y);
        double cos_z = cos(ang_z);

        double sin_x = sin(ang_x);
        double sin_y = sin(ang_y);
        double sin_z = sin(ang_z);

        homogeneous.data[0] = cos_z*cos_y; homogeneous.data[1] =
cos_z*cos_y*sin_x-sin_z*cos_x; homogeneous.data[2] =
cos_z*sin_y*cos_x+sin_z*sin_x; homogeneous.data[3] = pos_x;

        homogeneous.data[4] = sin_z*cos_y; homogeneous.data[5] =
sin_z*sin_y*sin_x-cos_z*cos_x; homogeneous.data[6] = sin_z*sin_y*cos_x-
cos_z*sin_x; homogeneous.data[7] = pos_y;

        homogeneous.data[8] = -sin_y; homogeneous.data[9] = cos_y*sin_x;
homogeneous.data[10] = cos_y*cos_x; homogeneous.data[11] = pos_z;

        homogeneous.data[12] = 0;homogeneous.data[13] = 0;
homogeneous.data[14] = 0; homogeneous.data[15] = 1;

        outPort.write(homogeneous);
    }
}
```

Component 2

The component which worked in the opposite direction, *component 2*, had to translate a 6 by 1 vector (wrench) to 3 by 1 vector (forces).

```
std_msgs::Float64MultiArray wrench_youbot;
geometry_msgs::Vector3 force_vector;
```

The sample output message is formed and sent in order to start the communications.

```
bool configureHook() {

    geometry_msgs::Vector3 samplevector;
    //samplevector.x = samplevector.y = samplevector.z = 0.0;
    outPort.setDataSample(samplevector);

    return true;
}
```

And the *updateHook()* takes the last 3 components of the wrench which correspond to the forces part and sends it to the haptic device. *kf* is the applied scale value.

```
void updateHook() {

    TaskContext::updateHook();

    if(youbot_force.read(wrench_youbot) == NewData) //Force feedback
    {

        force_vector.x = wrench_youbot.data[3] / kf;
        force_vector.y = wrench_youbot.data[4] / kf;
        force_vector.z = -wrench_youbot.data[5] / kf;

        outPort.write(force_vector);

    }
}
```

As can be observed in the code from *component 2* (and *component 1* as well) the mapping of the axis requires the change of sign of both x and y axis.

This simple qualitative test successfully proved that the used connection scheme was appropriate for the application. If one tried to move the device's pen tip out of the equilibrium point the latter would move back to the original position as if it were pulled by springs, just as intended.

3.4.1.2 Case adaptation

Now, since the control model on the youBot already uses the Cartesian Space Stiffness block the components developed during this first experiment could be directly connected to it without further modifications.

The only issue that had to be addressed was the mapping of the orientations. In the first experiment only the positions had been used. Now that the youBot was to be controlled with the haptic device, the desired orientation had to be included as well in the commanded signal.

After the first tests it was decided to use quaternions for the orientations/rotations because they do not have the singularities existent in the Euler angles representation. The code of *component 1* related to this part can be seen next. The command *Rotation::Rot i* (θ) creates a rotation matrix around axis i with angle θ (rad).

```
    rot_matrix = Rotation::Identity();
    rot_y_1 = Rotation::RotY(3*M_PI/4);
```

The order in which the rotation matrices are multiplied for the obtaining of *rot_matrix* is set by the kinematic structure of the haptic device itself. Once *rot_matrix* is calculated the quaternion is obtained using '*.Rotation::GetQuaternion(qx,qy,qz,qw)*'.

```
        ang_x = -hap_state.orientation.x;
        ang_y = -hap_state.orientation.y;
        ang_z = hap_state.orientation.z;

        rotz = Rotation::RotZ(ang_x);
        roty = Rotation::RotY(ang_y);
        rotx = Rotation::RotX(ang_z);

        rot_matrix = rot_y_1*rotx*roty*rotz;

        rot_matrix.Rotation::GetQuaternion(qx,qy,qz,qw);
```

The control cycle that resulted from using the haptic device state as the input for the controller on the youBot could be described as follows.

It starts with the haptic device changing its state. Once this information has been received by the controller, it compares it to the current state of the youBot tooltip and creates a force (wrench) signal that, modified by other components like gravity compensation, will be sent to the robot actuators. This evaluation takes place at the Cartesian Space Stiffness block as explained before. It compares the signals by their homogeneous matrices and then uses both translational and rotational stiffness to create the forces (and torques).

These same forces (indeed only the translational part since the Omega6 haptic device does not have force feedback in the rotational degrees of freedom) are the ones used for the force feedback. They are given negative sign and they are scaled to match the haptic device and the human operator force range. The signal is then sent to the haptic device.

The complete code for both components (1 and 2) can be found in 7.2 *Appendix B*.

3.4.2 Data filtering

The development of the controller for the youBot is at the moment being done using the modelling tool 20Sim (this program is briefly commented in chapter 1.3 *Mains Used*, under *Software*). Using a proprietary code generation algorithm the 20Sim component is converted into an Orocos component in C++.

Due to the use of the haptic device to transmit inputs for the controller a filter had to be implemented to remove the unwanted high frequency part of the signal. It would serve as well to soften any sudden movement coming from the operator that could harm the setup.

The selected solution consisted on interpolating the necessary points between the current state and the desired one in order to provide the control model with a smooth signal. This was also built on 20Sim and integrated at the beginning of the control model.

The interpolation was applied separately to translation and rotation. For the translation the position error is calculated for each variable independently as well as the combined. If the combined error is above a certain threshold, then the next point is set by adding to the current position a predefined step. The relation between the variable error and the combined error is used to weight the contribution to the final movement by each axis.

```
//Translation
   dif_1 = abs(xyz_qb[1]-xyz_qa[1]);
   dif_2 = abs(xyz_qb[2]-xyz_qa[2]);
   dif_3 = abs(xyz_qb[3]-xyz_qa[3]);
   dif = [dif_1;dif_2;dif_3];
   modulus = sqrt(dif_1^2+dif_2^2+dif_3^2);

   if modulus > min_trans then
      max_dif = max(dif);
      if max_dif == 0 then max_dif = 1; end;

      xyz_qm[1] = xyz_qa[1] + sign(xyz_qb[1]-xyz_qa[1])*trans_step*
*(dif_1/max_dif);
      xyz_qm[2] = xyz_qa[2] + sign(xyz_qb[2]-xyz_qa[2])*trans_step*
*(dif_2/max_dif);
      xyz_qm[3] = xyz_qa[3] + sign(xyz_qb[3]-xyz_qa[3])*trans_step*
*(dif_3/max_dif);
   else
      xyz_qm[1:3] = xyz_qb[1:3];
   end;
```

For the orientations the spherical linear interpolation (Slerp) method was used. It was introduced by Ken Shoemake [35] for rotations in 3D animation. It allows one to interpolate between quaternions at a specific point denoted by *t*, which goes from 0 (actual orientation) to 1 (desired orientation). The interpolated quaternion (q_m) can be calculated using the following equation:

$$q_m = \frac{q_a \sin\big((1-t)\theta\big) + q_b \sin(t\theta)}{\sin\theta}$$

where q_a represents the first quaternion and q_b the second quaternion. θ is half the angle between q_a and q_b.

This implementation always provides the shortest path to the desired orientation. Only when $\theta = 180°$ the result is arbitrary since there is no shortest direction to rotate. However during testing and probably due to decimals rounding in these cases the rotation is always clockwise (with the axis of rotation pointing upwards from the plane).

In the following page is the implemented code for the interpolation of the quaternions. It can be observed as well how after the calculation of *cos_half_theta* there is an 'if' statement to check whether it is negative or not. If this is the case it means that $q_a = -q_b$ and therefore the desired orientation is the same as the current one. The problem is that *arccos* returns a value between 0 and *pi/2* for positive inputs and between *pi/2* and *pi* for negative inputs. Since the shortest angle is pursued *cos_half_theta* is always kept positive in order to assure that *half_theta* is below *pi/2* and theta below *pi*. This modification has been included from [36].

```
   //Rotation
   //IN RADIANS!
   cos_half_theta = xyz_qa[4]*xyz_qb[4] + xyz_qa[5]*xyz_qb[5]
+ xyz_qa[6]*xyz_qb[6] + xyz_qa[7]*xyz_qb[7];

   if cos_half_theta < 0 then
      cos_half_theta = -cos_half_theta;
      xyz_qb[4:7] = -xyz_qb[4:7];
   end;

   if cos_half_theta >= 0.9999999 then
      cos_half_theta = 1;
   end;

   half_theta = arccos(cos_half_theta);
   sin_half_theta = sqrt(1 - cos_half_theta^2);

   if abs(2*half_theta) > min_rot then
      t = rot_step/(2*half_theta);
      xyz_qm[4:7] = (xyz_qa[4:7].*sin((1-t)*half_theta)
+ xyz_qb[4:7].*sin(t*half_theta))./sin(half_theta);
   else
      xyz_qm[4:7] = xyz_qb[4:7];
   end;

   if abs(cos_half_theta) >= 1 then
      xyz_qm[4:7] = xyz_qa[4:7];
   end;

   xyz_qa = xyz_qm
```

Although the translations and rotations are dealt with separately, the resulting effect is smooth enough and does not feel unnatural. It is neither slow because if the error is smaller than the specified threshold the signal will pass unmodified. This block greatly improved the previous filtering of the controller's inputs, providing extra stability to the system.

The complete code can be found in 7.2 *Appendix B*, under *youBot controller input smoothing*.

3.4.3 Velocity control

A known problem of the haptic teleoperators is, as explained previously, the matching between the workspace ranges of the master and the slave. The Omega6 haptic device has a limited workspace that could be scaled to a bigger workspace. But in the current project the youBot's workspace is virtually unlimited in the sense that it is a mobile platform and thus, it can travel to any desired position and orientation.

In order to be able to explore large areas using the limited haptic device's workspace velocity or rate control is required. In most of the cases what is done is to use an ancillary button to enter another control mode. However, in this case a more integrated solution was studied in order to bring a more intuitive experience to the operator.

To accomplish this, the haptic device position range was studied and modelled using a spheroid. This surface was then used as the switch between the two control modes. Inside the spheroid a position control approach was used in which the Omega6 pen tip and the youBot's tooltip were mapped together. If the haptic device's pen tip were to reach the spheroid, and subsequently its motion range limit, the control mode would change to velocity control. This would mean that the virtual setpoint sent to the youBot controller would move at constant speed. The direction of this movement is defined by the haptic device's axis origin and the current position of the pen tip. Furthermore, if the haptic device's pent tip would move while being above the limit established by the spheroid, it would only affect the direction of the movement without interrupting the rate control. This allows the possibility of moving the virtual setpoint to any desired position without stopping.

In order to create the spheroid the Omega6 pen tip was moved to its outermost positions during a long movement trying to find its range of motion in all the three degrees of freedom. This trajectory was recorded and used to fit inside a spheroid. The initial idea was to use a paraboloid but it did not properly fill the recorded surface so it was later decided to use the spheroid. In figures 36 and 37 can be seen the recorded trajectory and the fitted spheroid, defined with the next equation:

$$1 = \sqrt{\left(\frac{z - 0.008}{0.1}\right)^2 + \left(\frac{y}{0.1}\right)^2 + \left(\frac{x + 0.08}{0.14}\right)^2}$$

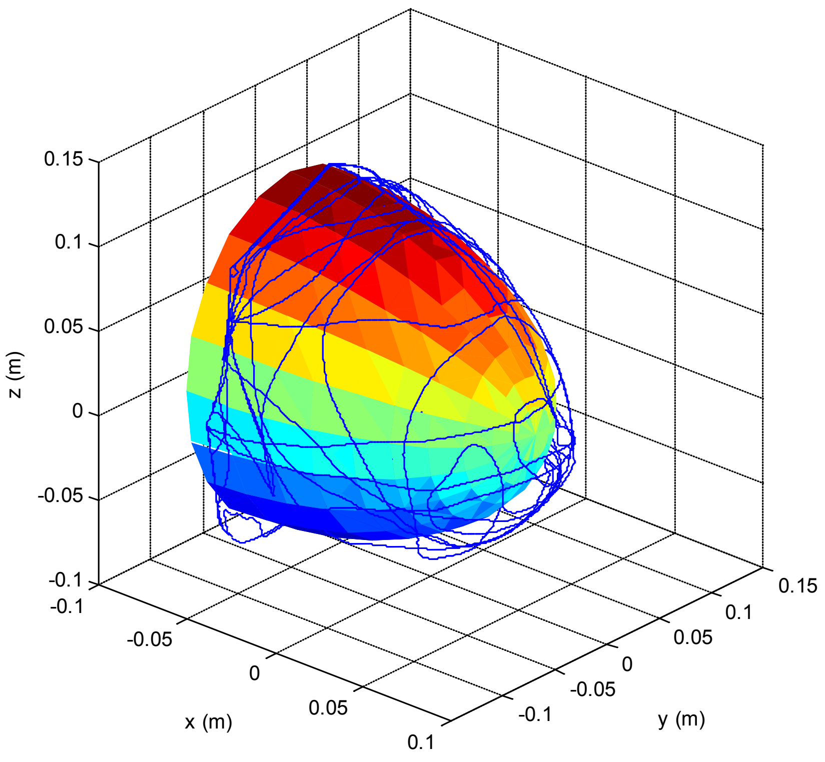


Figure 36: Trajectory along the Omega6 workspace limits and the fitted spheroid in 3D.

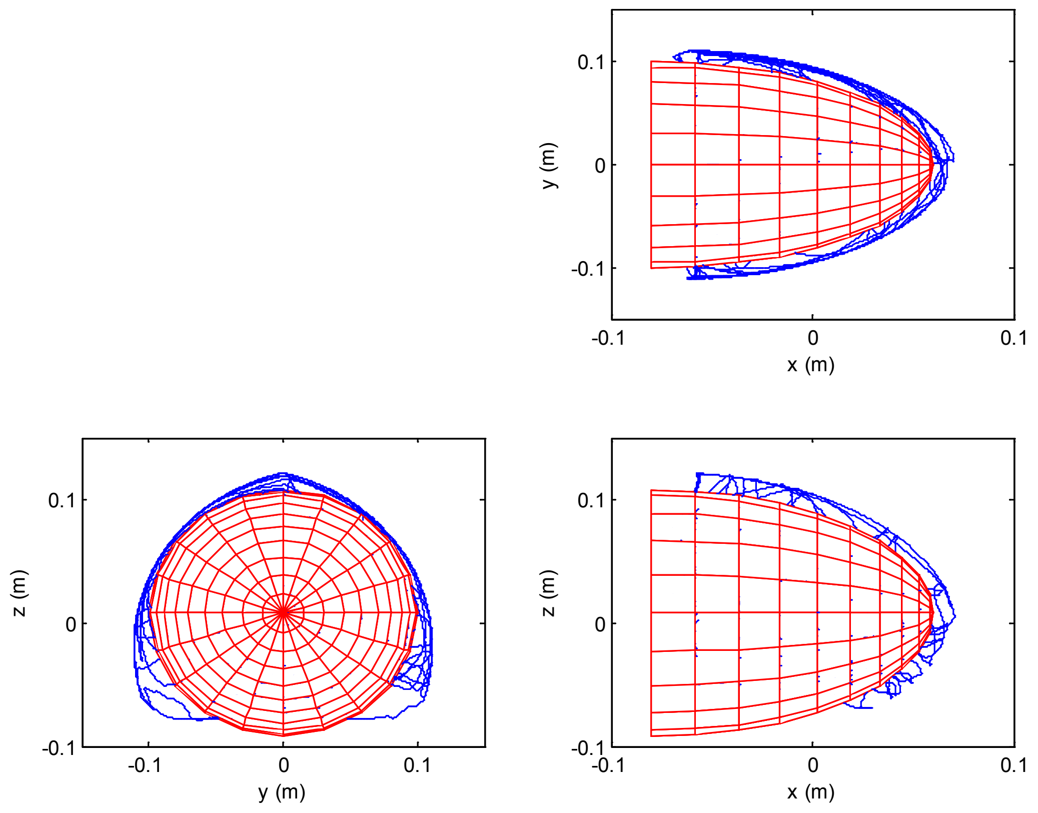


Figure 37: Top, front and side views of the spheroid used to model the Omega6 workspace.

Apart from finding a switching function, an important aspect of this implementation is to keep record of the difference between the actual position of the haptic device pen tip and the position of the virtual point sent to the youBot controller. The reason behind this is that once the pen tip reaches its motion limit, the control mode will switch to velocity control and the robot's tooltip will continue moving. If the Omega6 pen tip enters inside the spheroid again, the control mode switches to position control and the commanded position from the haptic device must match that of the youBot tooltip (or the virtual setpoint at which the youBot is going).

The last concept can be better clarified while the developed code is commented. The following code snippets belong to the *updateHook()* of *component 1* described. First the haptic device position is obtained (*x* and *y* axes are given negative sign to map the youBot coordinates frame) and two moduli are calculated: one for the switching spheroid, and the other for the direction of the constantly moving virtual setpoint.

```
        hx = -hap_state.position.x;
        hy = -hap_state.position.y;
        hz = hap_state.position.z;

        mod = sqrt(pow(hx,2)+pow(hy,2)+pow(hz,2)); //Just for velocity
direction
        mod_h = sqrt(pow((-hx+0.08)/0.14,2) + pow(-hy/0.1,2) + pow((hz-
0.008)/0.1,2)); //Spheroid modulus
```

It can be observed how, since *hx* and *hy* have had their sign changed, when calculating *mod_h* the x and y terms of the spheroid are also given negative sign. This applies for all the times in which the haptic device coordinates frame is used.

Besides the spheroid, due to the irregular shape of the haptic workspace two planes (at x = -0.042 and z = -0.066) were also used to define the surface used for the control mode switching.

Taking into account *mod_h* and these two planes there are two possibilities. Either the haptic device's pen tip is inside the switching surface (position control) or outside it (velocity control). However, there are two extra cases. These correspond to the changes from position control to velocity control and vice versa. A binary variable, *rate_control*, is used to determine whether this is the case or not.

If the pen tip is inside the switching surface and *rate_control* is false then the control mode is position control. *drift_i* is a variable that storages the cumulative difference between the haptic device's pen tip and the actual virtual point position commanded to the youBot.

```
if(mod_h < R && hx < 0.042 && hz > -0.066 && rate_control == false)
//Position control
        {
            pos_x = hx + drift_x; pos_y = hy + drift_y; pos_z = hz +
drift_z;
        }
```

If the pen tip position is on the surface or outside it then the control mode is velocity control. If *rate_control* is false, it means that this control mode has just started, so three different values are saved with the postfix '_old': the current pen tip position, the current commanded virtual point position and the current drift between these two.

The commanded position is then constantly incremented at each iteration with a value v*t. The modulus *mod* is used to scale this increment in function of the haptic device's pen tip position, *hi*.

```
else if(mod_h >= R || hx >= 0.042 || hz <= -0.066) //Rate control
        {
            if(rate_control == false) //Before start rate control
            {
                pos_x_old = pos_x; pos_y_old = pos_y; pos_z_old = pos_z;
                drift_x_old = drift_x; drift_y_old = drift_y; drift_z_old =
drift_z;

                hx_old = hx; hy_old = hy; hz_old = hz;
                rate_control = true;
            }

            pos_x = pos_x + v*t*hx/mod;
            pos_y = pos_y + v*t*hy/mod;
            pos_z = pos_z + v*t*hz/mod;
        }
```

Another possibility is to define a starting velocity and a final velocity in order to promote certain acceleration when starting the rate control mode. This was initially implemented but later removed due to the risk that could bring in the hands of an inexperienced operator.

The last case is when the pen tip position returns inside the switching surface after velocity control. Then the total drift is calculated summing up the difference between the old and the current commanded positions, the old accumulated drift, and the difference between the old and the current haptic device position.

The virtual setpoint to be commanded to the youBot is calculated by adding the total accumulated drift to the current haptic device position and the *rate_control* is set again to false.

```
else if(mod_h < R && hx < 0.042 && hz > -0.066 && rate_control == true)
//Back to position control
        {
            drift_x = (pos_x - pos_x_old) + drift_x_old - (hx-hx_old);
            drift_y = (pos_y - pos_y_old) + drift_y_old - (hy-hy_old);
            drift_z = (pos_z - pos_z_old) + drift_z_old - (hz-hz_old);

            pos_x = hx + drift_x;
            pos_y = hy + drift_y;
            pos_z = hz + drift_z;

            rate_control = false;
        }
```

3.4.4 Deployment

An important phase of the haptic interface development was the deployment of the different parts involved. This has to be done in a specific order to ensure both safety and performance.

The main components are: Omega6, *component 1*, *component 2*, youBot controller and the youBot itself. The Omega6 ROS node has to be run before starting the Orocos components using a deployment script. The script code can be found in 7.2 *Appendix B*, under *Deployment script*.

Once the Omega6 node is running the haptic device 'force display' button is pressed. Then the script, which loads and connects the Orocos components, can be executed. The only thing that moves at this time should be –besides the youBot calibration- the Omega6 pen tip moving to its coordinate frame origin. This is because at the beginning the haptic device's force feedback is set in a way that the pen tip position is at (0, 0, 0). This is programmed at *component 2* and later will be shown how it is done.

The next (and last) action that the user needs to take is to execute the function set_initial_position() in *component 1*. The first thing that this function does is to move the youBot arm to an extended vertical position. Since this component is connected to the youBot it can command and read the joint angles on the arm.

```
    m_modes = vector<ctrl_modes>(NR_OF_ARM_SLAVES, PLANE_ANGLE);
    op_setControlModes(m_modes);

    for(unsigned int i = 0; i < NR_OF_ARM_SLAVES; ++i)
    {
        joint_angles.positions[i] = unfoldedPosition[i];
    }

    joint_pos_out.write(joint_angles);
    bool done = false;
    while(!done)
    {
        yb_in.read(youbot_states);

        done = true;
        for(unsigned int i = 0; i < NR_OF_ARM_SLAVES; ++i)
        {
            if( abs(youbot_states.position[i]*180/M_PI -
unfoldedPosition[i]*180/M_PI) > 10) //in degrees
            {
                done = false;
                break;
            }
        }

    }
```

The control mode for the arm is then changed to accept torques instead of joint angles and the translational and rotational stiffness properties are given a relatively small value. Finally, the flag *to_initial_position* is set to true.

Only when *to_initial_position* is true another flag, *start_cart_control* is activated. Additionally, a message is sent to *component 2* with value 1 (*to_initial_position* = true).

```
    // Start haptic position stream
    if(to_initial_position == true)
    {
        start_cart_control = true;

        force_feedback_flag.write(to_initial_position);
        to_initial_position = false;
    }
```

Once *start_cart_control* is true and whenever there is new data from the haptic device, the virtual setpoint position and orientation is calculated (as explained before in section 3.4.3 *Velocity control*) and sent to the youBot controller.

```
    if(hap_in.read(hap_state) == NewData && start_cart_control == true)
    {
            ...

xyzquat.data[0] = pos_x; xyzquat.data[1] = pos_y; xyzquat.data[2] = pos_z;
        xyzquat.data[3] = qw; xyzquat.data[4] = qx; xyzquat.data[5] = qy;
xyzquat.data[6] = qz;

        cart_pos_out.write(xyzquat);
            ...
    {
```

The youBot controller will get the first virtual setpoint position and move the robot arm accordingly. This point is located at the coordinates (0.35, 0, 0.35), just in front of the base. This is achieved by initializing *drift_x* and *drift_z* with an initial value of 0.35 cm. In this way, when the drift is added to the haptic device position (0, 0, 0), the resulting setpoint has the desired coordinates.

At the same time the message with *to_initial_position* has arrived to *component 2*. It then waits for the moment in which the youBot tooltip reaches the initial commanded setpoint. This is accomplished by monitoring the *z* term of the force calculated by the Cartesian Space Stiffness block inside the youBot controller. As explained before, this force is the same to be displayed on the haptic device but scaled and with its sign changed. When this force is below a certain value, the flag *start_force_feedback* is set to true.

```
    if(to_initial_position == true && flag == true) //Start force feedback
when tooltip is at initial setpoint
        {
            if(youbot_force.read(wrench_youbot) == NewData)
            {
                if(abs(wrench_youbot.data[5]) < 0.1 &&
wrench_youbot.data[5] != 0)
                {
                    start_force_feedback = true;
                }
            }
        }
```

When *start_force_feedback* becomes true the haptic device force to be displayed changes from an initial mode to the actual force feedback. The initial mode actually follows the same principle used on the first experiment explained in the previous chapter. It just creates a force dependent on the position error in order to bring the Omega6 pen tip to the coordinates frame origin (0, 0, 0). In the force feedback mode the force part of the wrench coming from the youBot controller is given a negative sign and scaled using *kf*. Then it is sent to the haptic device.

```
    if(youbot_force.read(wrench_youbot) == NewData && start_force_feedback
== true) //Force feedback
    {

        force_vector.x = wrench_youbot.data[3] / kf;
        force_vector.y = wrench_youbot.data[4] / kf;
        force_vector.z = -wrench_youbot.data[5] / kf;

        outPort.write(force_vector);

    }


    if(hap_in.read(haptic_state) == NewData && start_force_feedback ==
false) //At the beggining -> (0,0,0)
    {

        force_vector.x = -haptic_state.position.x * 40;
        force_vector.y = -haptic_state.position.y * 40;
        force_vector.z = -haptic_state.position.z * 40;


        outPort.write(force_vector);

    }
```

Additionally, the moment in which *start_force_feedback* is true a message with value true is sent to *component 1*.

```
    if(start_force_feedback == true && flag == true)
    {
        rise_stiff.write(start_force_feedback);
        flag = false;
    }
```

When this is received in *component 1*, it means that the haptic device and the robot's tooltip are already mapped and the Cartesian stiffness can be increased to the default values. The stiffness (*kt* for translations and *kr* for rotations) are increased gradually in order to avoid rapid movements and for safety reasons.

```
    if(rise_stiff.read(ff_started) == NewData || ff_started == true)
    {
        if (kt < 700)
        {
            kt = kt + 0.1;
        }
        if (kr < 30)
        {
            kr = kr + 0.005;
        }
    }

    cart_stiff.data[0] = kt; cart_stiff.data[1] = kt; cart_stiff.data[2] =
kt;
    cart_stiff.data[3] = kr; cart_stiff.data[4] = kr; cart_stiff.data[5] =
kr;
    cart_stiff_out.write(cart_stiff);
```

These stiffness values are then sent to the youBot controller, specifically to the Cartesian Space Stiffness, to be used for the calculation of the wrench.

In summary, the haptic device moves the pen tip to its frame origin and waits for the youBot to be ready. The youBot first unfolds the arm and then moves the tooltip to a point located in front of it. Once this happens the force feedback is enabled at the master side and the Cartesian stiffness are increased gradually to the appropriate operation values. The teleoperation system is ready to be used.

Figure 38 depicts in the next page the different components involved and their connections (*component 1* is hap_to_xyzypr and *component 2* is tsim_to_hap). It clearly shows which of them are working under ROS and which under Orocos.

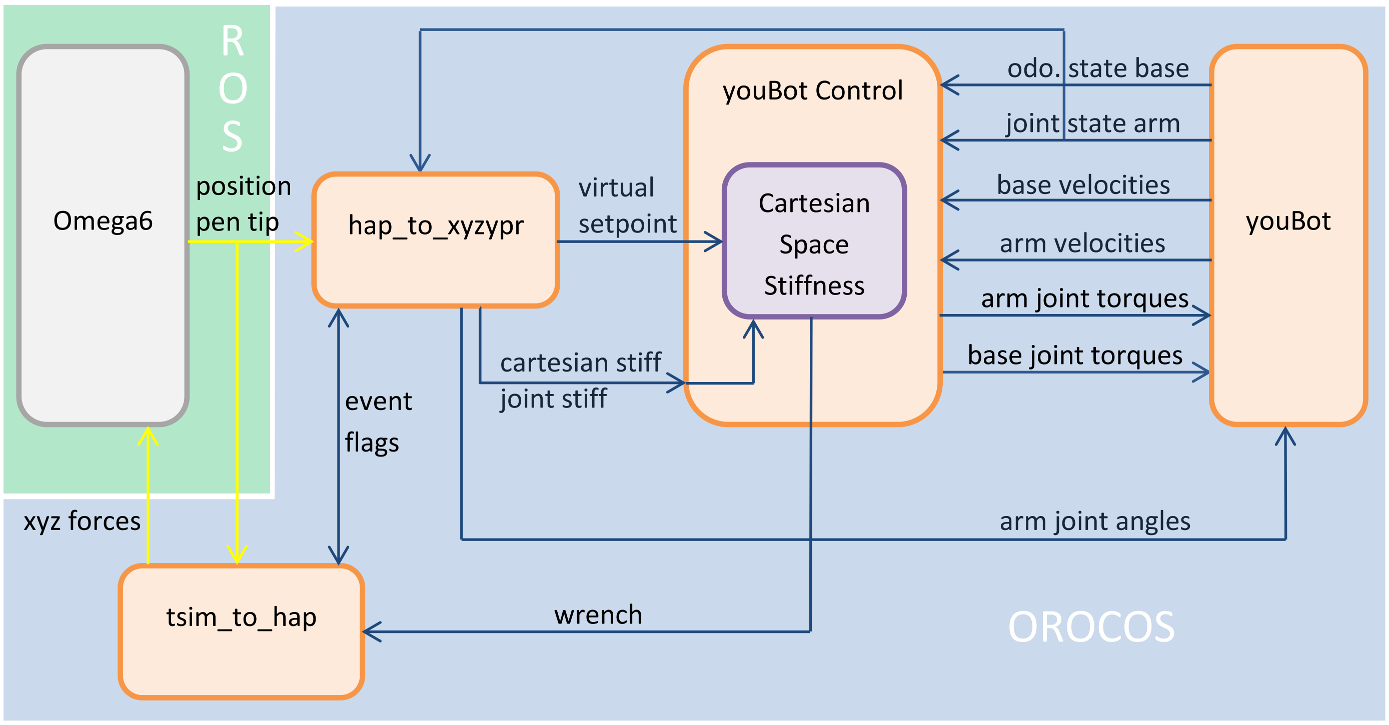


Figure 38: Haptic teleoperator components and the connections between them.

Overall, the deployment of the different components is marked by the existent forces and the necessity of activating them in a natural order. Doing this diminishes the wear of the actuated devices, decreases the time required to get a fully working system, and increases safety for both the operator and the components involved in the setup.

3.4.5 Limitations

Because the friction compensation developed on this thesis was not fully operational until quite late, the haptic interface development was carried out using only the arm of the youBot.

The fact that the base was still limited the arm's workspace considerably. This was even more noticeable when the haptic device was used to control the arm's tooltip. The physical limitations of the robot movements could be felt on the haptic device. The reason behind this is no other than the difference in the kinematic structures of both devices.

The haptic device has the rotational degrees of freedom decoupled from the translations and thus can perform –within its limited workspace– a wide variety of positions and orientations combinations in the Cartesian coordinate space. Since the arm is a jointed structure based on rotations some positions cannot be achieved without changing the orientation as well.

Moreover, if the rotational degrees of the haptic device were actuated this effect would greatly decrease because the youBot's limitations would be transferred to the haptic device and felt with torques opposing to the operator's commands. This is, however, not the case.

This is a problem that could not be overcome. Despite this, when the operator understands these limitations and realizes in which way the robot can or cannot move, it becomes easier and more intuitive to control it.

The only dangerous situation is when the haptic device itself is let free and only the orientation of the pen tip is changed. The youBot's arm will rapidly move to match the tooltip's and the demanded orientation. The problem is that, when doing so, the position is changed as well and since, the haptic device is faster than the youBot, the former will rapidly move to the current position of the robot's tooltip. This can lead to an unstable cycle if the setup is left unmonitored.

Even so, the addition of the base to the youBot's kinematic chain changes this limitation. Ideally, the base could move together with the arm in order to change only the orientation when commanded. Still, in the real system the behaviour is far from ideal and the problem persists, which means that the teleoperator has to be handled keeping this in mind.

Another limitation is related to the control architecture used, which is position control. Since the displayed force on the haptic device depends on the position error between the current tooltip position and that of the virtual setpoint, in some cases the stiffness felt at the master are differ considerably with reality.

This is most of the times when the youBot tooltip interacts with a hard surface. One should feel a high stiffness in the haptic device, but instead there is no force until the virtual point is set beyond this surface. This results in a loss of the felt realism and a poorer teleoperation experience.

3.5 Conclusions

An interface that allows controlling the youBot using the haptic device Omega6 has been developed. The developed interface maps the pen tip of the haptic device to the tooltip of the robot’s arm in order for them to replicate each other movements.

The lack of force sensors on the master side limited the possible solutions applicable to obtain force feedback on the slave side (youBot). In order to solve this, the forces displayed at the haptic device have been based on those that drive the robot’s tooltip, which in turn are based on the position error. This workaround does, however, bring the haptic teleoperator further away from reality since hard surfaces do not translate into high impedances until the virtual point has already gone across them.

A very interesting problem concerning the coupling of the Omega6 and the youBot has been identified, i.e. the sudden movements of the mobile robot’s arm are caused by their kinematic structures being different and the lack of active rotations on the haptic device. However, since this problem is implicit in the physical structures of the two components it cannot be solved until these are modified or changed.

Velocity control has been implemented into the developed solution. It allows changing from position control to rate control by bringing the haptic device’s pen tip to its motion limit. This improvement highly increases the haptic teleoperator functionality in a very intuitive manner.

The haptic teleoperator interface has been implemented so as to provide an automatic deployment and allows changing multiple parameters online for the best optimization. Despite not having achieved a perfect haptic teleoperation system, the solution brought in this project is in line with the sought objectives and works with good performance.

4 Final conclusions and future work

Apart from the specific conclusions of each main part of this project here some general conclusions and comments are exposed.

The main objective of this work was to create an interface with which one could control the youBot through a haptic device. Along the process one important goal was added to the main one, which is to apply friction compensation to the base motor blocks in order to be able to use the base properly. This was decided after studying the possible parameters estimation and validation procedures applicable to the robot and because the other joints structure would not allow for a complete system study.

The obtained friction results at the wheels' motor blocks do not correspond to the classical friction theory. The probable reason for this is that the studied mechanical system includes all motor, gears and bearings, so it has a high complexity. Since neither the initially selected model nor any other studied model can be applied, a customized static friction model was developed from the gathered experimental results. The model is a major step forward in the modelling of the friction at the base motor blocks of the youBot and improves the simulation model.

The constructed friction model was then used to apply friction compensation to the base with acceptable results. The implemented solution is enough for the base to move successfully, meeting the goal requirements.

A haptic interface has been built in order to use an Omega6 haptic device with the youBot. It allows controlling the position of the arm's tooltip, feeling at the same time the limitations that it may be experiencing at the master side. As initially planned, the impedance received at the haptic device is not the exact force that the environment is exerting on the youBot, but it is enough to feel the arm dynamics and its interaction with the real world.

Due to the slave-master different kinematic structures, the system can get unstable if the orientation of the Omega6 pen tip is changed accidentally, however this is highly unlikely to happen under direct surveillance of the user.

The developed haptic interface works as demanded and the youBot can be moved around in an intuitive and secure way. Velocity control has been added in order to extend the haptic workspace. The deployment of the system has been automatized in order to reduce the non-experienced user input and to guarantee a safe procedure.

All in all, the project goals have been successfully reached and the developed system is completely functional. During the process research in different subjects has been done and some ideas have been left out of the project for time or out-of-scope reasons. I myself have expanded my knowledge in friction modelling and haptics considerably. I have also learned how a research project develops itself and how important is collaboration to bring it to successful result.

Future work

The friction modelling of the base motor blocks is enough for the current purposes. However the friction on the arm joints greatly affects its behaviour. It could be advisable to realize a complete system parameters identification and subsequent validation. This would greatly improve both the simulation model and the behaviour of the real robot.

It could be a good idea also to fix the offset current at the motors and to improve the velocity obtaining that the youBot does from its joints' position. This would help to get a better velocity feedback and then the overall control would be improved as well.

On the haptic interface, and once the current measurements at the joints are assured to be correct, the environment impedance felt at the youBot's tooltip could be calculated from the rest of the joints' torques. This would mean that the actual force felt by the youBot could be fed back to the haptic device, creating a more realistic feeling (most noticeable for hard surfaces contact).

Finally, in order to eliminate the potential instability situation mentioned before it would be needed to get a haptic device with active rotations or with a different kinematic structure. This does not mean however that other solutions may not be found through software, but during this work none was found.

5 Budget

5.1 Introduction

The present project has been performed by student Miguel Corberán Ruiz at the premises of the Robotics and mechatronics lab of the University of Twente under the frame of the Erasmus Placement program.

The project has run from 15/09/11 to 15/05/12, comprising 6 months stay at the University of Twente and 2 extra months at home dedicated to compose the present document.

The research is in the field of robotics and mechatronics, specifically in mobile robots and haptic teleoperation. In the following a description of the incurred costs of the project is included together with a final summary of the budget.

5.2 Costs justification

Personnel

The work done by the student Miguel Corberán Ruiz has been supervised by MSc. Yury Brodskiy and MSc. Robert Wilterdink, and professor dr. ir. Peter Breedveld. Several lab technicians have assisted when necessary.

The average cost of the MSc student at UT is approximately 1000 Euro per month. Although the actual Erasmus Placement grant has been total 2200 Euro over 6 months.

The other salaries have been estimated from average salaries at UT. The reported dedication corresponds more or less to the actual dedication of the mentioned persons.

Last name and name	Category	Effort (person month) [a)]	Person month cost	Cost (Euro)
Corberán Ruiz, Miguel	MSc student	8	1,000.00	8,000.00
Brodskiy, Yury	MSc, PhD candidate	1	3,000.00	3,000.00
Wilterdink, Robert	MSc, Engineer	1	4,000.00	4,000.00
Breedveld, Peter	Scientific staff	0.2	7,000.00	1,400.00
Several	Technical staff	0.5	4,000.00	2,000.00
	Total effort	**10.7**	**Total**	18,400.00

Equipment

The main equipment employed along the project has been the youBot robot and the haptic device Omega6. They have been involved in other projects at the same time. Most of the used software has been open source, with exception of the development program 20Sim and the mathematics simulation platform Matlab.

Additionally to the personal laptop, a development PC has been used for the experimental tests, practical simulations and programming. The depreciation period for hardware is 5 years, for electronics 2.5 years and for software licenses 1 year.

The costs for depreciation of office space or lab are not included, neither the industrial benefit since this this project has been carried out at the UT.

The depreciation formula for the equipment cost has been the following:

$$Cost = \frac{A}{B}CD$$

where A is the period of employment in the project, B is the depreciation period, C is the equipment cost (without VAT) and D is the percentage of equipment use in the project.

Description	Cost (Euro)	% Use in project	Dedication (months)	Depreciation period (months)	Cost (Euro) b)
youBot	20,000.00	30	4	60	400.00
Omega6	30,000.00	30	5	60	750.00
Development PC	2,000.00	30	4	30	80.00
Personal laptop	800.00	100	6	30	160.00
20Sim license	1,000.00	30	6	12	150.00
Matlab license	200.00	30	5	12	25.00
				Total	1,565.00

Subcontracting

There has been no outsourcing in this project.

Description	Company	Cost (Euro)
n/a	n/a	n/a
	Total	0.00

Other direct costs

Only a small percentage of lab material has been necessary for the correct execution of the project. This comprises assorted connection cables and fixing products (glue, tape, etc.) among others.

Description	Company	Cost (Euro)
Consumable lab supplies	n/a	100.00
	Total	100.00

Overheads

The approximate overheads rate calculation at UT is 40 % under total cost justification model. This percentage is assumed over an average research case.

5.3 Costs summary

Type	Total cost (Euro)
Personnel	18,400.00
Equipment	1,565.00
Subcontracting	0.00
Other direct costs	100.00
Overheads	8,026.00
Total	28,091.00

TOTAL COST (EXCLUDING VAT)	**28,091.00€**
21% VAT	**5,899.11€**
TOTAL COST PROJECT	**33,990.11€**

The total cost of the project is # **THIRTY-THREE THOUSAND NINE HUNDRED NINETY EURO WITH ELEVEN CENTS OF EURO** #.

6 References

[1] Al-Bender, F..: 'Fundamentals of friction modelling', Proceedings of the ASPE Spring Topical Meeting on Control of Precision Systems, pp. 117-122, Cambridge, USA, April 11-13, 2010.

[2] Al-bender, F. and Swevers, J.: 'Characterization of friction force dynamics: Behavior and modeling on micro and macro scales', IEEE Control Systems Magazine, Vol. 28 (no. 6), pp. 82-91 (2008).

[3] Márton, L. and Lantos, B.: 'Identification and Model-based Compensation of Striebeck friction', Acta Polytechnica Hungarica, Vol. 3 (no. 3), pp. 45-58 (2006).

[4] Olsson, H., Åström, K.J., Canudas de Wit, C., Gäfvert, M. and Lischinsky, P.: 'Friction models and friction compensation', European Journal of Control, Vol. 4 (no. 3), pp. 176–195 (1998).

[5] Tjahjowidodo, T., Al-Bender, F. and Van Brussel, H.: 'Friction identification and compensation in a dc motor', Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 4-8, 2005.

[6] Karnopp, D.: 'Computer simulation of slip-stick friction in mechanical dynamic systems', Journal of Dynamic Systems, Measurement, and Control, Vol. 107 (no. 1), pp. 100–103 (1985).

[7] Armstrong-Hélouvry, B., Dupont, P., and Canudas de Wit, C.: 'A survey of models, analysis tools and compensation methods for the control of machines with friction', Automatica, Vol. 30 (no. 7), pp. 1083–1138 (1994).

[8] Dahl, P.: 'A solid friction model', Technical Report TOR-0158(3107–18)-1, The Aerospace Corporation, El Segundo, CA, 1968.

[9] Haessig, D.A., Friedland, B.: 'On the modelling and simulation of friction', Journal of Dynamic Systems, Measurement and Control Transactions ASME, Vol. 113 (no. 3), pp. 354–362 (1991).

[10] Canudas de Wit, C., Olsson, H., Åström, K.J., and Lischinsky, P.: 'A new model for control of systems with friction', IEEE Transactions on Automatic Control, Vol. 40 (no. 3), pp. 419-425 (1995).

[11] Altpeter, F.: 'Friction Modeling, Identification and Compensation'. Doctoral Thesis, École Polytechnique Fédérale de Lausanne, 1999.

[12] Swevers, J., Al-Bender, F., Ganseman, C., and Prajogo, T.: 'An integrated friction model structure with improved presliding behavior for accurate friction compensation', IEEE Transactions on Automatic Control, Vol. 45 (no.4), pp. 675-686 (2000).

[13] Lampaert, V., Swevers, J. and Al-bender, F.: 'Experimental comparison of different friction models for accurate low-velocity tracking', Proceedings of the 10th Mediterranean conference on control and Automation, pp. 833-837, Lisbon, Portugal, July 9-12, 2002.

[14] Lampaert, V., Swevers, J., and Al-Bender, F.: 'Modification of the Leuven integrated friction model structure', IEEE Transactions on Automatic Control, Vol. 47 (no. 4), pp. 683–687 (2002).

[15] Iwan, W.D.: 'A distributed-element model for hysteresis and its steady-state dynamic response', Journal of Applied Mechanics, Vol. 33 (no. 4), pp. 893-900 (1966).

[16] Al-Bender, F., Lampaert, V., Swevers, J.: 'The generalized Maxwell-slip model: a novel model for friction Simulation and compensation', IEEE Transactions on Automatic Control, Vol. 50 (no. 11), pp. 1883-1887 (2005).

[17] Vuong, N.D. and Ang, M.H.: 'Dynamic model identification for industrial robots', Acta Polytechnica Hungarica, Vol.6 (No. 5), pp. 51–68 (2009).

[18] Khalil, W., Gautier, M. and Lemoine, P.: 'Identification of the payload inertial parameters of industrial manipulators'. IEEE International Conference on Robotics and Automation, 4943-4948, Roma, Italy, 10-14 April, 2007.

[19] Swevers, J., Verdonck, W. and De Schutter, J.: 'Dynamic model identification for industrial robots', IEEE Control Systems Magazine, Vol. 27 (no. 5), pp. 58–71 (2007).

[20] Wu, J., Wang, J. and You, Z.: 'An overview of dynamic parameter identification of robots', Robotics and computer-integrated manufacturing, Vol. 16, pp. 414–419 (2010).

[21] Susanto, W., Babuska, R., Liefhebber, F. and Van der Weiden, T.: 'Adaptive Friction Compensation: Application to a robotic manipulator', Proceedings of the 17th World Congress: The International Federation of Automatic Control, 2020-2024, Seoul, Korea, July 6-11, 2008.

[22] Canudas de Wit, C., and Lischinsky, P.: 'Adaptive friction compensation with partially known dynamic friction model', International journal of adaptive control and signal processing, Vol. 11 (no. 4), pp. 65–80 (1997).

[23] Waiboer, R.: 'Dynamic Modelling, Identification and Simulation of Industrial Robots – for Off-line Programming of Robotised Laser Welding –' (Netherlands institute for metals research, 2007).

[24] Lischinsky, P., Canudas-de Wit, C. and Morel, G.: 'Friction compensation for an industrial hydraulic robot', IEEE Control Systems Magazine, Vol. 19 (no. 1), pp. 25–32 (1999).

[25] Kelly, R., Llamas, J. and Campa, R.: 'A measurement procedure for viscous and Coulomb friction', IEEE Transactions on instrumentation and measurement, Vol. 49 (no. 4), pp. 857–861 (2000).

[26] Broenink J. F.: ' Introduction to Physical Systems Modelling with Bond Graphs', Technical report, University of Twente, Enschede, Netherlands, 1999.

[27] Stone, R. J.: 'Haptic feedback: A potted history, from telepresence to virtual reality', Stone, R. J.: Proceedings of the First International Workshop on Haptic Human-Computer Interaction, Glasgow, UK, pp. 1-17 (2001).

[28] McLaughlin, M.L., Hespanha, J.P., Sukhatme, G.S.: 'Touch in virtual environments: haptics and the design of interactive systems' (Prentice Hall, 2002, ISBN: 9780130650979).

[29] Zadeh, M.H.: 'Advances in haptics' (InTech, 2010, ISBN: 9789533070933).

[30] El Sakkik, A.: 'Haptics rendering and applications' (InTech, 2012, ISBN: 9789533078977).

[31] Klomp, F.M.: ‘Haptic Control for Dummies: An introduction and analysis’. Master’s Thesis, Eindhoven University of Technology, 2006.

[32] Hayward, V. and Astley, O. R.: ‘Performance measures for haptic interfaces’. Robotics Research: The 7th International Symposium, pp. 195-207 (1996).

[33] Ryu, J. H., Kwon, D. S. and Hannaford, B.: ‘Stability guaranteed control: Time domain passivity approach’, IEEE Transactions on Control Systems Technology, Vol. 12 (No. 6), pp. 860–868 (2004).

[34] Tan, H. Z., Srinivasan, M. A., Eberman, B. and Cheng, B.: ‘Human factors for the design of force-reflecting haptic interfaces’, Dynamic Systems and Control, Vol. 55 D1, pp. 353–359 (1994).

[35] Shoemake, K.: ‘Animating rotation with quaternion curves’, Proceedings of the 12th annual conference on Computer graphics and interactive techniques, Vol. 19 (no. 3), pp. 245-254 (1985).

[36] http://www.euclideanspace.com, Accessed on February 2012.

7 Appendices

7.1 Appendix A

In this appendix can be found the 20Sim code for the friction models of each wheel with the customized curves fitted from the experimental data. Despite having developed the curves with Matlab the implementation on the youBot control model is done with 20Sim. This is why the following code is from the 20Sim models.

7.1.1 Wheel 1

```
parameters
   real a = 0.05615;
   real b = -8.68;
   real c = 0.04245;
   real d = 0.2383;

   real a2 = 0.07022;
   real b2 = -0.1932;
   real c2 = 7.554e-10;
   real d2 = 6.858;

   real p11 = 0.07582;
   real p12 = -0.1201;

   real a3 = 3.243e9;
   real b3 = -7.739;
   real c3 = 0.07271;
   real d3 = 0.09647;

   real p21 = 0.0125;
   real p22 = 0.05584;

   //Negative values (part not symmetric)
   real a4 = -0.003067;
   real b4 = -1.606;
   real c4 = 0.0007382;
   real d4 = -1.966;

   real a5 = -4.42e5;
   real b5 = 4.899;
   real c5 = -0.05591;
   real d5 = -0.1906;

variables
   real abs_vel;
   real r1,r2,r3,r4,r5,r6,r7,r8,r9,r10;
   real n;

code
```

```
abs_vel = abs(vel);

r1  = 5*abs_vel + 1e-6;
r2  = a*exp(b*abs_vel) + c*exp(d*abs_vel);
r3  = a2*exp(b2*abs_vel) + c2*exp(d2*abs_vel);
r4  = p11*abs_vel + p12;
r5  = a3*exp(b3*abs_vel) + c3*exp(d3*abs_vel);
r6  = p21*abs_vel + p22;

r7  = 0.06 + 0.4*abs_vel;
r8  = 0.071 + 0.01*abs_vel;

r9  = a4*exp(b4*vel) + c4*exp(d4*vel);
r10  = a5*exp(b5*vel) + c5*exp(d5*vel);

if abs_vel < 0.7 then
   n = 25;
   rfriction_FL = 1/(1/r1^n + 1/r2^n + 1/r7^n + 1/r8^n)^(1/n);
end;

if abs_vel >= 0.7 and abs_vel < 1.5 then
   n = 30;
   rfriction_FL = 1/(1/r2^n + 1/r3^n)^(1/n);
end;

if abs_vel >= 1.5 and abs_vel < 2.5 then
   rfriction_FL = r3;
end;

if abs_vel >= 2.5 and abs_vel < 3.2 then
   n = 30;
   rfriction_FL = 1/(1/r3^n + 1/r4^n)^(1/n);
end;

if abs_vel >= 3.2 and abs_vel < 3.5 then
   n = 20;
   rfriction_FL = 1/(1/r4^n + 1/r5^n)^(1/n);
end;

if abs_vel >= 3.5 and abs_vel < 4.59 then
   rfriction_FL = r5;
end;

if abs_vel >= 4.59 then
   rfriction_FL = r6;
end;

 rfriction_FL = rfriction_FL*sign(vel);

//Correction negative part (not symmetric)
if vel <= -2.6 then
   rfriction_FL = r9;
end;
if vel <= -3.59 then
```

```
      rfriction_FL = r10;
   end;
   if vel <= -9.52 then
      rfriction_FL = 0.0551*vel + 0.1812;
   end
```

7.1.2 Wheel 2

```
parameters
   real a = 0.05139;
   real b = -6.585;
   real c = 0.04364;
   real d = 0.2313;

   real a2 = 0.05552;
   real b2 = -0.08399;
   real c2 = 1.937e-6;
   real d2 = 4.167;

   real p11 = 0.08767;
   real p12 = -0.1249;

   real a3 = 7.255e4;
   real b3 = -4.762;
   real c3 = 0.07822;
   real d3 = 0.09864;

   real p21 = 0.01689;
   real p22 = 0.04036;

   //Negative values (part not symmetric)
   real a4 = -0.06804;
   real b4 = -0.1412;
   real c4 = -27.68;
   real d4 = 2.221;

variables
   real abs_vel;
   real r1,r2,r3,r4,r5,r6,r7,r8,r9;
   real n;

code

   abs_vel = abs(vel);

   r1 = 5*abs_vel + 1e-6;
   r2 = a*exp(b*abs_vel) + c*exp(d*abs_vel);
   r3 = a2*exp(b2*abs_vel) + c2*exp(d2*abs_vel);
   r4 = p11*abs_vel + p12;
   r5 = a3*exp(b3*abs_vel) + c3*exp(d3*abs_vel);
   r6 = p21*abs_vel + p22;
   r7  = 0.06 + 0.5*abs_vel;
   r8  = 0.077 + 0.01*abs_vel;

   r9  = a4*exp(b4*vel) + c4*exp(d4*vel);

   if abs_vel  < 0.4 then
      n = 25;
      rfriction_FR = 1/(1/r1^n + 1/r2^n + 1/r7^n + 1/r8^n)^(1/n);
   end;

   if abs_vel  >= 0.4 and abs_vel  < 1.5 then
      n = 45;
      rfriction_FR = 1/(1/r2^n + 1/r3^n)^(1/n);
   end;

   if abs_vel  >= 1.5 and abs_vel  < 2.2 then
      rfriction_FR = r3;
   end;

   if abs_vel  >= 2.2 and abs_vel  < 2.9 then
      n = 45;
      rfriction_FR = 1/(1/r3^n + 1/r4^n)^(1/n);
   end;

   if abs_vel  >= 2.9 and abs_vel  < 3.5 then
      n = 20;
      rfriction_FR = 1/(1/r4^n + 1/r5^n)^(1/n);
   end;

   if abs_vel  >= 3.5 and abs_vel  < 5.89 then
      rfriction_FR = r5;
   end;

   if abs_vel  >= 5.89 then
      rfriction_FR = r6;
   end;

rfriction_FR = rfriction_FR*sign(vel);

   //Correction negative part (not symmetric)
   if vel  <= -3.09 then
      rfriction_FR = r9;
   end;
   if vel  <= -9.46 then
      rfriction_FR = 0.03165*vel  + 0.04064;
   end
```

7.1.3 Wheel 3

```
parameters
   real a = 0.4187;
   real b = -0.3006;
   real c = 0.1296;

   real a2 = 0.03022;
   real b2 = -0.08004;
   real c2 = 0.1077;

   real a3 = 5306;
   real b3 = -3.548;
   real c3 = 0.09904;
   real d3 = 0.08162;

   real p21 = 0.01298;
   real p22 = 0.08432;

   //Negative values (part not symmetric)
   real a4 = 0.0201;
   real b4 = -1.6094;
   real c4 = -0.0248;
   real d4 = -1.5578;

   real a5 = -0.08219;
   real b5 = -0.1575;
   real c5 = -250.2;
   real d5 = 2.62;

variables
   real abs_vel;
   real r1,r2,r3,r4,r5,r6,r7,r8,r9;
   real n;

code

   abs_vel = abs(vel);

   r1 = 5*abs_vel + 1e-6;
   r2 = a*abs_vel^2 + b*abs_vel + c;
   r3 = a2*abs_vel^2 + b2*abs_vel + c2;
   r4 = a3*exp(b3*abs_vel) + c3*exp(d3*abs_vel);
   r5 = p21*abs_vel + p22;

   r6 = 0.08195 + 0.25*abs_vel;
   r7 = 0.105 + 0.01*abs_vel;

   r8 = a4*exp(b4*vel) + c4*exp(d4*vel);
   r9 = a5*exp(b5*vel) + c5*exp(d5*vel);

   if abs_vel < 0.3 then
      n = 40;
      rfriction_RL = 1/(1/r1^n + 1/r2^n + 1/r6^n + 1/r7^n)^(1/n);
   end;

   if abs_vel >= 0.3 and abs_vel < 1 then
      n = 45;
      rfriction_RL = 1/(1/r2^n + 1/r3^n)^(1/n);
   end;

   if abs_vel >= 1 and abs_vel < 3 then
      rfriction_RL = r3;
   end;

   if abs_vel >= 3 and abs_vel < 5.89 then
      n = 12;
      rfriction_RL = 1/(1/r3^n + 1/r4^n)^(1/n);
   end;

   if abs_vel >= 5.89 then
      rfriction_RL = r5;
   end;

rfriction_RL = rfriction_RL*sign(vel);

   //Correction negative part (not symmetric)
   if vel <= -2.59 then
      rfriction_RL = r8;
   end;
   if vel <= -3.53 then
      rfriction_RL = r9;
   end;
   if vel <= -9.6 then
      rfriction_RL = 0.05377*vel + 0.1425;
   end
```

7.1.4 Wheel 4

```
parameters
   real a = 0.6494;
   real b = -0.3383;
   real c = 0.09933;

   real a2 = 0.1069;
   real b2 = -0.1064;
   real c2 = 0.07555;

   real p11 = 0.008156;
   real p12 = 0.0449;

   real a3 = 0.1098;
   real b3 = -1.174;
   real c3 = 4.627;
   real d3 = -7.879;
   real e3 = 4.948;

   real a4 = 2828;
   real b4 = -3.707;
   real c4 = 0.08658;
   real d4 = 0.1062;

   real p21 = 0.01829;
   real p22 = 0.05499;

   //Negative values (part not symmetric)
   real a5 = -0.008942;
   real b5 = -1.0836;
   real c5 = 2.27e-005;
   real d5 = -2.6953;

   real a6 = -0.06523;
   real b6 = -0.2179;
   real c6 = -1801;
   real d6 = 3.471;

variables
   real abs_vel;
   real r1,r2,r3,r4,r5,r6,r7,r8,r9,r10;
   real n;

code

   abs_vel = abs(vel);

   r1 = 5*abs_vel + 1e-6;
   r2 = a*abs_vel^2 + b*abs_vel + c;
   r3 = a2*abs_vel^2 + b2*abs_vel + c2;
   r4 = p11*abs_vel + p12;
   r5 = a3*abs_vel^4 + b3*abs_vel^3 + c3*abs_vel^2 + d3*abs_vel + e3;
   r6 = a4*exp(b4*abs_vel) + c4*exp(d4*abs_vel);
   r7 = p21*abs_vel + p22;

   r8 = 0.06255 + 0.4255*abs_vel;

   r9 = a5*exp(b5*vel) + c5*exp(d5*vel);
   r10 = a6*exp(b6*vel) + c6*exp(d6*vel);

   if abs_vel < 0.2568 then
      n = 10;
      rfriction_RR = 1/(1/r1^n + 1/r2^n + 1/r8^n)^(1/n);
   end;

   if abs_vel >= 0.2568 and abs_vel < 0.555 then
      rfriction_RR = r3;
   end;

   if abs_vel >= 0.555 and abs_vel < 1.867 then
      rfriction_RR = r4;
   end;

   if abs_vel >= 1.867 and abs_vel < 2.7 then
      n = 30;
      rfriction_RR = r5;
   end;

   if abs_vel >= 2.7 and abs_vel < 5.34 then
      n = 30;
      rfriction_RR = 1/(1/r5^n + 1/r6^n)^(1/n);
   end;

   if abs_vel >= 5.34 then
      rfriction_RR = r7;
   end;

   rfriction_RR = rfriction_RR*sign(vel);

   //Correction negative part (not symmetric)
   if vel <= -2.6 then
      rfriction_RR = r9;
   end;
   if vel <= -3.31 then
      rfriction_RR = r10;
   end;
   if vel <= -9.15 then
      rfriction_RR = 0.08636*vel + 0.3112;
   end
```

7.2 Appendix B

In this appendix can be seen the complete code from the developed components for the haptic interface.

7.2.1 hap_to_xyzypr (component 1)

```
#include "hap_to_xyzypr-component.hpp"
#include <rtt/Component.hpp>
#include <rtt/Attribute.hpp>
#include <kdl/frames.hpp>
#include <math.h>

using namespace RTT;
using namespace KDL;
using namespace YouBot;

const double unfoldedPosition[] = { 169 * M_PI / 180.0, 65 * M_PI / 180.0,
-146 * M_PI / 180.0,
        102.5 * M_PI / 180.0, 167.5 * M_PI / 180.0 }; //radian!

const double foldedPosition[] = { 0.115385 * M_PI / 180.0, 0.0980769 * M_PI
/ 180.0, -0.0936 * M_PI / 180.0,
        0.14831 * M_PI / 180.0, 4.7193 * M_PI / 180.0 }; //radian!

double pos_x, pos_y, pos_z;     // position in [m]
double ang_x, ang_y, ang_z;     // angles in [rad]
double qw,qx,qy,qz;             // quaternions
Rotation rot_zyx,rotz,roty,rotx;
Rotation rot_y_1;
Rotation rot_matrix;

double t = 0.001, a, v_final = 10;  //cm/s
double hx,hy,hz,mod_h,mod; //   haptic position
double R = 1;
double pos_x_old,pos_y_old,pos_z_old;
double drift_x = 0.35, drift_y = 0, drift_z = 0.35; //initial
drift/position
double drift_x_old,drift_y_old,drift_z_old;
double difx,dify,difz,hx_old,hy_old,hz_old;
bool rate_control = false;

hapticteleop_msgs::HapticState hap_state;
sensor_msgs::JointState youbot_states;

std_msgs::Float64MultiArray xyzquat;
std_msgs::Float64MultiArray cart_stiff;
motion_control_msgs::JointPositions joint_angles;
std_msgs::Float64MultiArray joint_stiff;
motion_control_msgs::JointPositions gripper_pos;
bool ff_started = false;

bool to_initial_position = false;
bool start_cart_control = false;
bool gripper_closed = true;

Hap_to_xyzypr::Hap_to_xyzypr(string const& name)
```

```
    : TaskContext(name),
      kt(0.0),
      kr(0.0),
      v(1),
      m_modes(NR_OF_ARM_SLAVES, MOTOR_STOP)
    {

    this->ports()->addEventPort("hap_in", hap_in).doc("Haptic state in");
    this->ports()->addEventPort("rise_stiff", rise_stiff).doc("Can rise
stiffnesses to default");
    this->ports()->addPort("yb_in", yb_in).doc("Youbot state in");
    this->ports()->addPort("cart_pos_out", cart_pos_out).doc("xyzquat
vector out");
    this->ports()->addPort("cart_stiff_out", cart_stiff_out).doc("Cartesian
stiffness vector out");
    this->ports()->addPort("joint_pos_out", joint_pos_out).doc("Joint
angles vector out");
    this->ports()->addPort("joint_stiff_out", joint_stiff_out).doc("Joint
stiffness vector out");
    this->ports()->addPort("force_feedback_flag",
force_feedback_flag).doc("Start youbot force feedback");
    //this->ports()->addPort("start_adapter", start_adapter).doc("Start
youbot adapter");

    this->ports()->addPort("gripper_pos_out", gripper_pos_out).doc("Gripper
position out");

    this->addOperation("set_initial_position",
&Hap_to_xyzypr::set_initial_position, this, OwnThread);//.doc("Set endtip
to initial position");

    this->addAttribute("kt", kt);
    this->addAttribute("kr", kr);
    this->addAttribute("v", v);

    std::cout << "Hap_to_xyzypr constructed !" <<std::endl;
    }

Hap_to_xyzypr::~Hap_to_xyzypr() {}

bool Hap_to_xyzypr::configureHook() {

    //Send port samples and initialize messages

    xyzquat.data.resize(7, 0.0);
    cart_pos_out.setDataSample(xyzquat);

    cart_stiff.data.resize(9, 0.0);
    cart_stiff_out.setDataSample(cart_stiff);

    joint_angles.positions.assign(NR_OF_ARM_SLAVES, 0.0);
    joint_pos_out.setDataSample(joint_angles);

    joint_stiff.data.resize(8, 0.0);
    joint_stiff_out.setDataSample(joint_stiff);

    gripper_pos.positions.assign(1,0.0);
    gripper_pos_out.setDataSample(gripper_pos);

    force_feedback_flag.setDataSample(to_initial_position);
    rot_matrix = Rotation::Identity();
    rot_zyx = Rotation::Identity();
    rot_y_1 = Rotation::RotY(3*M_PI/4);

    mod_h = 0;
    a = v_final/(5);

    return true;
}

bool Hap_to_xyzypr::set_initial_position() {

    m_modes = vector<ctrl_modes>(NR_OF_ARM_SLAVES, PLANE_ANGLE);
    op_setControlModes(m_modes);

    for(unsigned int i = 0; i < NR_OF_ARM_SLAVES; ++i)
    {
        joint_angles.positions[i] = unfoldedPosition[i];
    }

    joint_pos_out.write(joint_angles);

    bool done = false;
    while(!done)
    {
        yb_in.read(youbot_states);

        done = true;
        for(unsigned int i = 0; i < NR_OF_ARM_SLAVES; ++i)
        {
            if( abs(youbot_states.position[i]*180/M_PI -
unfoldedPosition[i]*180/M_PI) > 10) //in degrees
            {
                done = false;
                break;
            }
        }
    }
    //Now we control with youbot_control in cart space -> write flag for
updatehook

    m_modes = vector<ctrl_modes>(NR_OF_ARM_SLAVES, TORQUE);
    op_setControlModes(m_modes);

    kt = 100;
    kr = 10;

    to_initial_position = true;

    return true;
}

bool Hap_to_xyzypr::startHook() {

    if( !hap_in.connected() )
    {
      std::cout << "Input port not connected!" <<std::endl;
      return false;
    }
    if( !yb_in.connected() )
    {
      std::cout << "Output port not connected!" <<std::endl;
      return false;
    }

    if( !rise_stiff.connected() )
    {
      std::cout << "Output port not connected!" <<std::endl;
      return false;
    }

    TaskContext* task_ptr = getPeer("youbot");
    if(task_ptr == NULL)
    {
        log(Error) << "Could not find peer YouBot_OODL" << endlog();
        return false;
    }

    op_setControlModes = task_ptr->provides("Arm1")-
>getOperation("setControlModes");

    if(!op_setControlModes.ready())
    {
        log(Error) << "Could not connect to Arm1.setControlModes" <<
endlog();
        return false;
    }

    m_modes = vector<ctrl_modes>(NR_OF_ARM_SLAVES, MOTOR_STOP);
    op_setControlModes(m_modes);

    return TaskContext::startHook();
}

void Hap_to_xyzypr::updateHook() {

    TaskContext::updateHook();

    // Start haptic position stream
    if(to_initial_position == true)
    {
        //start_hap.data[0] = 1;
        start_cart_control = true;

        force_feedback_flag.write(to_initial_position);
        to_initial_position = false;
    }

    //start_adapter.write(start_hap);

    if(hap_in.read(hap_state) == NewData && start_cart_control == true)
    {
        hx = -hap_state.position.x;
        hy = -hap_state.position.y;
        hz = hap_state.position.z;

        mod = sqrt(pow(hx,2)+pow(hy,2)+pow(hz,2)); //Just for velocity
direction
        mod_h = sqrt(pow((-hx+0.08)/0.14,2) + pow(-hy/0.1,2) + pow((hz-
0.008)/0.1,2)); //Spheroid modulus
```

```
        if(mod_h < R && hx < 0.042 && hz > -0.066 && rate_control == false)
//Position control
        {
            pos_x = hx + drift_x; pos_y = hy + drift_y; pos_z = hz +
drift_z;
        }
        else if(mod_h >= R || hx >= 0.042 || hz <= -0.066) //Rate control
        {
            if(rate_control == false) //Before start rate control
            {
                pos_x_old = pos_x; pos_y_old = pos_y; pos_z_old = pos_z;
                drift_x_old = drift_x; drift_y_old = drift_y; drift_z_old =
drift_z;

                hx_old = hx; hy_old = hy; hz_old = hz;
                rate_control = true;
            }

            pos_x = pos_x + v*t*hx/mod;
            pos_y = pos_y + v*t*hy/mod;
            pos_z = pos_z + v*t*hz/mod;

        }
        else if(mod_h < R && hx < 0.042 && hz > -0.066 && rate_control ==
true) //Back to position control
        {
            drift_x = (pos_x - pos_x_old) + drift_x_old - (hx-hx_old);
            drift_y = (pos_y - pos_y_old) + drift_y_old - (hy-hy_old);
            drift_z = (pos_z - pos_z_old) + drift_z_old - (hz-hz_old);

            pos_x = hx + drift_x;
            pos_y = hy + drift_y;
            pos_z = hz + drift_z;

            rate_control = false;
        }

        ang_x = -hap_state.orientation.x;
        ang_y = -hap_state.orientation.y;
        ang_z = hap_state.orientation.z;

        rotz = Rotation::RotZ(ang_x);
        roty = Rotation::RotY(ang_y);
        rotx = Rotation::RotX(ang_z);

        rot_matrix = rot_y_1*rotx*roty*rotz;

        rot_matrix.Rotation::GetQuaternion(qx,qy,qz,qw);

        xyzquat.data[0] = pos_x; xyzquat.data[1] = pos_y; xyzquat.data[2] =
pos_z;
        xyzquat.data[3] = qw; xyzquat.data[4] = qx; xyzquat.data[5] = qy;
xyzquat.data[6] = qz;

        cart_pos_out.write(xyzquat);

        //If button pressed, close gripper and vice versa.
        if(hap_state.button.data == true && gripper_closed == false)
        {
            gripper_pos.positions[0] = 0.010;
            gripper_pos_out.write(gripper_pos);
            gripper_closed = true;
        }
        else if(hap_state.button.data == false && gripper_closed == true)
        {
            gripper_pos.positions[0] = 0.02;
            gripper_pos_out.write(gripper_pos);
            gripper_closed = false;
        }

    }

    if(rise_stiff.read(ff_started) == NewData || ff_started == true)
    {
        if (kt < 700)
        {
            kt = kt + 0.1;
        }
        if (kr < 30)
        {
            kr = kr + 0.005;
        }
    }

    cart_stiff.data[0] = kt; cart_stiff.data[1] = kt; cart_stiff.data[2] =
kt;
    cart_stiff.data[3] = kr; cart_stiff.data[4] = kr; cart_stiff.data[5] =
kr;
    cart_stiff_out.write(cart_stiff);

    /*
    joint_stiff.data[0] = 0; joint_stiff.data[1] = 0; joint_stiff.data[2] =
0; joint_stiff.data[3] = 0;
    joint_stiff.data[4] = 0; joint_stiff.data[5] = 0; joint_stiff.data[6] =
0; joint_stiff.data[7] = 0;
    joint_stiff_out.write(joint_stiff);*/
}

void Hap_to_xyzypr::stopHook() {
    m_modes = vector<ctrl_modes>(NR_OF_ARM_SLAVES, MOTOR_STOP);
    op_setControlModes(m_modes);

    TaskContext::stopHook();
}

void Hap_to_xyzypr::cleanupHook() {
    TaskContext::cleanupHook();
}

ORO_CREATE_COMPONENT(Hap_to_xyzypr)
```

7.2.2 tsim_to_hap (component 2)

```
#include "tsim_to_hap-component.hpp"
#include <rtt/Component.hpp>

using namespace RTT;

std_msgs::Float64MultiArray twist_youbot;
hapticteleop_msgs::HapticState haptic_state;
bool to_initial_position = false;
geometry_msgs::Vector3 force_vector;


bool start_force_feedback = false;
bool flag = true;

Tsim_to_hap::Tsim_to_hap(string const& name)
        : TaskContext(name),
          kf(10.0)
    {

    this->ports()->addEventPort("youbot_force", youbot_force).doc("Twist
vector in (from youbot_control)");
    this->ports()->addEventPort("hap_in", hap_in).doc("Haptic state in");
    this->ports()->addEventPort("force_feedback_flag",
force_feedback_flag).doc("Start youbot force feedback flag");
    this->ports()->addPort("outPort", outPort).doc("Force vector out");
    this->ports()->addPort("rise_stiff", rise_stiff).doc("Set final
stiffnesses");

    this->addAttribute("kf", kf);

    std::cout << "Tsim_to_hap constructed !" << std::endl;
    }

Tsim_to_hap::~Tsim_to_hap() {}

bool Tsim_to_hap::configureHook() {


    geometry_msgs::Vector3 samplevector;
    outPort.setDataSample(samplevector);

    rise_stiff.setDataSample(start_force_feedback);

    return true;
}

bool Tsim_to_hap::startHook() {

    if( !youbot_force.connected() )
    {
      std::cout << "youbot_control input port not connected!" <<std::endl;
      return false;
    }

    if( !hap_in.connected() )
    {
      std::cout << "Haptic state input port not connected!" <<std::endl;
      return false;
    }
    if( !force_feedback_flag.connected() )
    {
      std::cout << "Force feedback flag input port not connected!"
<<std::endl;
      return false;
    }

    return TaskContext::startHook();
}

void Tsim_to_hap::updateHook() {

    TaskContext::updateHook();

    force_feedback_flag.read(to_initial_position);

    if(to_initial_position == true && flag == true) //Start force feedback
when tooltip is at initial setpoint
        {
            if(youbot_force.read(twist_youbot) == NewData)
            {
                if(abs(twist_youbot.data[5]) < 0.1 && twist_youbot.data[5]
!= 0)
                {
                    start_force_feedback = true;
                }
            }
        }

    if(youbot_force.read(twist_youbot) == NewData && start_force_feedback
== true) //Force feedback
    {

        force_vector.x = twist_youbot.data[3] / kf;
        force_vector.y = twist_youbot.data[4] / kf;
        force_vector.z = -twist_youbot.data[5] / kf;

        outPort.write(force_vector);

    }

    if(start_force_feedback == true && flag == true)
    {
        rise_stiff.write(start_force_feedback);
        flag = false;
    }

    if(hap_in.read(haptic_state) == NewData && start_force_feedback ==
false) //At the beggining -> (0,0,0)
    {

        force_vector.x = -haptic_state.position.x * 40;
        force_vector.y = -haptic_state.position.y * 40;
        force_vector.z = -haptic_state.position.z * 40;


        outPort.write(force_vector);

    }
}
```

```
void Tsim_to_hap::stopHook() {

    start_force_feedback = false;

    TaskContext::stopHook();
}

void Tsim_to_hap::cleanupHook() {

    TaskContext::cleanupHook();
}

ORO_CREATE_COMPONENT(Tsim_to_hap)
```

7.2.3 youBot controller input smoothing

```
parameters
   real min_trans = 5.0e-4; //Translation interpolation threshold
   real trans_step = 2.5e-4; //Translation interpolation step
   real min_rot_deg = 0.05; //Rotation interpolation threshold, IN
DEGREES!
   real rot_step_deg = 0.01; //Rotation interpolation step, IN DEGREES!
variables
   real xyz_qa[7];
   real cos_half_theta;
   real sin_half_theta;
   real half_theta;
   real t;
   real min_rot,rot_step;
   real dif_1,dif_2,dif_3,max_dif,modulus;
   real dif[3];
initialequations
   xyz_qa = [0;0;0;1;0;0;0];
   min_rot = min_rot_deg*pi/180;
   rot_step = rot_step_deg*pi/180;
   t = 0;

equations

   //Translation
   dif_1 = abs(xyz_qb[1]-xyz_qa[1]);
   dif_2 = abs(xyz_qb[2]-xyz_qa[2]);
   dif_3 = abs(xyz_qb[3]-xyz_qa[3]);
   dif = [dif_1;dif_2;dif_3];
   modulus = sqrt(dif_1^2+dif_2^2+dif_3^2);

   if modulus > min_trans then
      max_dif = max(dif);
      if max_dif == 0 then max_dif = 1; end;

      xyz_qm[1] = xyz_qa[1] + sign(xyz_qb[1]-
xyz_qa[1])*trans_step*(dif_1/max_dif);
      xyz_qm[2] = xyz_qa[2] + sign(xyz_qb[2]-
xyz_qa[2])*trans_step*(dif_2/max_dif);
      xyz_qm[3] = xyz_qa[3] + sign(xyz_qb[3]-
xyz_qa[3])*trans_step*(dif_3/max_dif);
   else
      xyz_qm[1:3] = xyz_qb[1:3];
   end;

   //Rotation
   //IN RADIANS!
   cos_half_theta = xyz_qa[4]*xyz_qb[4] + xyz_qa[5]*xyz_qb[5]
+ xyz_qa[6]*xyz_qb[6] + xyz_qa[7]*xyz_qb[7];

   if cos_half_theta < 0 then
      cos_half_theta = -cos_half_theta;
      xyz_qb[4:7] = -xyz_qb[4:7];
   end;

   if cos_half_theta >= 0.9999999 then
      cos_half_theta = 1;
   end;

   half_theta = arccos(cos_half_theta);
   sin_half_theta = sqrt(1 - cos_half_theta^2);

   if abs(2*half_theta) > min_rot then
      t = rot_step/(2*half_theta);
      xyz_qm[4:7] = (xyz_qa[4:7].*sin((1-t)*half_theta)
+ xyz_qb[4:7].*sin(t*half_theta))./sin(half_theta);
   else
      xyz_qm[4:7] = xyz_qb[4:7];
   end;

   if abs(cos_half_theta) >= 1 then
      xyz_qm[4:7] = xyz_qa[4:7];
   end;

   xyz_qa = xyz_qm
```

7.2.4 Deployment script

The following code pertains to the deployment script used to start all the necessary components into Orocos. The script was called using the *deployer*. The *deployer* allows to create applications from the components libraries and to interact with the running programs through a console type interface.

This code has been put here for information purposes only and will not be commented. It contains many lines of code which are implicit with the deployment of the youBot and its controller components and have not been developed in this project.

This script does the following: imports the stacks (group of components), loads the youBot and the necessary components, assigns a thread to the components, initializes the youBot and the components, connects the components, and finally, configures and starts the components.

```
import("YouBot_control");
import("hap_to_xyzypr");
import("tsim_to_hap");
import("YouBot_adapters");

//Configure the YouBot parameters on startup
runScript(rospack.find("YouBot_configurator")+"/deploy.ops");

//make youbot with events and monitor with standart output
runScript(rospack.find("YouBot_OODL")+"/deploy.ops");

loadComponent("youbot_control","youbot_control_hap::YouBot_control");
loadComponent("hap_to_xyzypr","Hap_to_xyzypr");
loadComponent("tsim_to_hap","Tsim_to_hap");
loadComponent("control_to_base", "YouBot::TSimToYouBotMsg");
loadComponent("control_to_arm", "YouBot::TSimToYouBotMsg");

loadComponent("base_state_to_control", "YouBot::YouBotMsgToTSim");
loadComponent("arm_state_to_control", "YouBot::YouBotMsgToTSim");
loadComponent("base_vel_to_control", "YouBot::YouBotMsgToTSim");
loadComponent("arm_vel_to_control", "YouBot::YouBotMsgToTSim");

setActivity("youbot_control",0.001,HighestPriority,ORO_SCHED_RT);
setActivity("hap_to_xyzypr",0.001,HighestPriority,ORO_SCHED_RT);
setActivity("tsim_to_hap",0.001,HighestPriority,ORO_SCHED_RT);

//Setup control mode (base and arm)
var std.vector<ctrl_modes>
base_ctrl_modes=std.vector<ctrl_modes>(4,TORQUE);
youbot.Base.clearControllerTimeouts();
youbot.Base.setControlModes(base_ctrl_modes);
control_to_base.initialize(TORQUE,4);

var std.vector<ctrl_modes>
arm_ctrl_modes=std.vector<ctrl_modes>(5,TORQUE);
youbot.Arm1.clearControllerTimeouts();
youbot.Arm1.setControlModes(arm_ctrl_modes);
control_to_arm.initialize(TORQUE,5);

base_state_to_control.initialize(4);
arm_state_to_control.initialize(5);
base_vel_to_control.initialize(4);
arm_vel_to_control.initialize(5);

var ConnPolicy cp_rt;
cp_rt.type = DATA;  // Use ''BUFFER'' or ''DATA''
cp_rt.lock_policy = LOCK_FREE; // Use ''LOCKED'', ''LOCK_FREE'' or
''UNSYNC''

//Setup connections from rtt to ros
//Requires H10..50 Hvp0 Htip0
runScript(rospack.find("YouBot_deployment")+"/rtt_deploy/ros_connection_hap
.ops")

connect("hap_to_xyzypr.cart_pos_out", "youbot_control.xyz_quat", cp_rt);
connect("hap_to_xyzypr.cart_stiff_out", "youbot_control.cart_stiffness",
cp_rt);
connect("hap_to_xyzypr.gripper_pos_out",
"youbot.Gripper1.gripper_cmd_position", cp_rt);
connect("hap_to_xyzypr.joint_stiff_out", "youbot_control.", cp_rt);
connect("hap_to_xyzypr.force_feedback_flag",
"tsim_to_hap.force_feedback_flag", cp_rt);
connect("hap_to_xyzypr.joint_pos_out", "youbot.Arm1.joint_cmd_angles",
cp_rt);
connect("youbot.Arm1.joint_states", "hap_to_xyzypr.yb_in", cp_rt);
connect("youbot_control.force_feedback", "tsim_to_hap.youbot_force",
cp_rt);
connect("tsim_to_hap.rise_stiff", "hap_to_xyzypr.rise_stiff", cp_rt);

connect("youbot_control.Base_torque_cmd","control_to_base.input_cmd_signal"
,cp_rt);
connect("control_to_base.output_cmd_torques","youbot.Base.joint_cmd_torques
",cp_rt);

connect("youbot_control.Arm1_joint_cmd","control_to_arm.input_cmd_signal",c
p_rt);
connect("control_to_arm.output_cmd_torques","youbot.Arm1.joint_cmd_torques"
,cp_rt);

connect("youbot.Base.joint_states","base_state_to_control.input_states",cp_
rt);
connect("base_state_to_control.output_positions","youbot_control.Base_joint
_states",cp_rt);


connect("youbot.Arm1.joint_states","arm_state_to_control.input_states",cp_r
t);
connect("arm_state_to_control.output_positions","youbot_control.Arm1_joint_
states",cp_rt);

connect("youbot.Base.joint_states","base_vel_to_control.input_states",cp_rt
);
connect("base_vel_to_control.output_velocities","youbot_control.Base_joint_
velocities",cp_rt);

connect("youbot.Arm1.joint_states","arm_vel_to_control.input_states",cp_rt)
;
connect("arm_vel_to_control.output_velocities","youbot_control.Arm1_joint_v
elocites",cp_rt);

connectPeers("hap_to_xyzypr","youbot");

stream("hap_to_xyzypr.hap_in", ros.topic("/teleop/haptic_state"));
stream("tsim_to_hap.hap_in", ros.topic("/teleop/haptic_state"));
stream("tsim_to_hap.outPort", ros.topic("/teleop/force_feedback"));


base_state_to_control.start
arm_state_to_control.start
base_vel_to_control.start
arm_vel_to_control.start

control_to_base.start
control_to_arm.start

youbot_control.configure
tsim_to_hap.configure
hap_to_xyzypr.configure

tsim_to_hap.start
hap_to_xyzypr.start
```

```
youbot_control.start

//hap_to_xyzypr.set_initial_position
```

This last action, *set_initial_position()*, can be automatized if necessary. However for safety reasons and during development it was preferred to call it manually.