ALMA MATER STUDIORUM - UNIVERSITY OF BOLOGNA

ENGINEERING AND ARCHITECTURE SCHOOL

DEPARTMENT OF ELECTRICAL,ELECTRONIC AND INFORMATION ENGINEERING – "GUGLIELMO MARCONI" - DEI

AUTOMATION ENGINEERING

GRADUATION THESIS

in
Industrial Robotics for Automation

Development of a twisted-string actuator for a cable-driven haptic interface

Candidate: Paolo Capello

Supervisor:
Chiar.mo Prof. Claudio Melchiorri

Co-supervisor:
Dott. Alberto Pepe

Academic Year 2016/2017

Session II

Dedicated to my mom and my father

Abstract

Il seguente lavoro di tesi ha avuto lo scopo di analizzare dal punto di vista della progettazione software le procedure ed i protocolli di scambio di informazioni tra i vari componenti di una nuova concezione di Interfaccia Aptica guidata da 4 tendini che muovono un braccialetto centrale collegato al braccio di un operatore cosí da fornirgli un feedback di forza. In particolare ci si é focalizzati sullo sviluppo di un firmware applicabile ai 4 motori che muovono la struttura centrale della interfaccia. Il firmware deve essere in grado di ricevere da una piattaforma Ros, usata da un operatore, pacchetti di dati contenenti i set point per i vari motori e il tipo di controllo , posizione o forza, che gli attuatori devono effettuare grazie ad uno schema PID. Inoltre l'invio di feedback all'operatore é stato previsto in modo da permettere una maggiore supervisione dell'intero funzionamento.

La realizzazione di un Ros Bridge tra l'utente e il sistema da comandare é stato implementato con la formula della programmazione ad oggetti in cui varie classi sono dedicate a compiti differenti come l'impacchettamento di dati da mandare ai motori e la contemporanea ricezione dei feedback.

Per completare tutta l'architettura si é anche sviluppato un sistema di trasformazione dei set point provenienti dall'operatore espressi nello spazio di lavoro Cartesiano in riferimenti per i singoli motori e ció é stato possibile sfruttando la matrice Jacobiana.

Una particolare attenzione é stata data all'aspetto di comunicazione dei dati e per fare ció si é dovuta usare una architettura di codice a multithread e un protocollo UDP.

La realizzazione di questo difficile ma soddisfacente lavoro é stata ottenuta grazie alla collaborazione con il laboratorio LAR della Facoltá di Ingegneria dell'Universitá di Bologna.

Contents

1 Chapter: Teleoperation in robotic applications

1.1 Introduction to robotics

Nowadays the role of robots is increasing exponentially in our lives. Although the evolution of industry 4.0 has allowed to reach the use of more sophisticated robots which are capable to dialogue one to each other inside the industrial environment, it cannot be possible any more to talk only about those ones which are present in the firms. Their usual work is to help to make components and devices through long assembly lines but this is a limited possibility for the technologies which are available.

As a matter of fact the presence of incredibly useful but complex machines like robots can help to face up to specific works of everyday life. Agriculture, aerospace explorations, underwater divings, military missions but also domotics and human safety are some of the huge variety of tasks that robot are able to do.

Figure 1.1: Example of domestic robot

The principal problem of robotic field is their programming phase aimed to control the motion and the actions that robots have to bring to completion. Some strategies have been developed in order to reach a completely autonomous robot during the years. For example it is common to see rovers that have to make explorations on other planets or with the goal of going in the deep ocean to find out some missing object on the sea bottom. They are studied to keep in touch with human user as less as possible. This interaction could be seen from the point of view of sending to him informations as feedback but it should not be present in the opposite direction which consists in controlling directly the robot by some form of guidance.

Although the progresses in the autonomous-robots creation are being made, the result of obtaining a complete interaction avoidance between device and "creator" is only an utopia. Of consequence it is still relevant to talk about one of the most interesting technique for what concerns the control of robots: Teleoperation.

1.2 Types of environment

In order to understand better the required features of teleoperated robots it is interesting to know the difference between the various environment in which they have to work and perform their tasks.

As a matter of fact the huge variety of advanced technological robotic machines can meet some favorable or unfavorable situations and they have to be equipped with the proper tools to face up to them. Moreover depending on the external environment a different grade of keeping in touch with the operator must be provided.

Usually in robotic field the best division of the working environments is done between Structured and Unstructured ones.

1.2.1 Structured environments

Starting the analysis from the structured type, it can be said that this seems to be an "easy" and ideal ambient to perform various tasks.

As a matter of fact its principal feature is that of having all the things and objects in precise positions and rarely they are moved in other space points. In practise there is a sort of clear configuration in which the robot can orient himself in the best way.

The first example of a structure environment is for sure the industrial ambient. This is generally composed by many objects which are seen by the robot as obstacles but the good aspect of the matter is that they are always is the same spatial configuration. Something like huge shelves, big pallets and machineries occupy the same portion of space and this allows the robot to reach a better knowledge of the ambient.

Figure 1.2: Example of structured environment

One main objection that the reader can do is that in the firms there is also the presence of people which introduces an aspect of randomness.

This is for sure true but it is not the relevant element because the workspace in which the robot performs the tasks is ”protected” by the possible intervention of the human.

One more example of this kind of environment is the supermarket construction. In this case the help of robots is not so widespread but it is beginning and so they can face up to a structured environment where products has to be moved or the ground has to be clean.

Talking about the equipment and complexity of the robot that has to work in this type of ambient, this deals with the union of actuation and sensing part.

As far as actuation is concerned, this has to show motion velocity feature and for sure precision. Just think about the rhythms of work inside the industries and the explanation of the previous sentence is given.

For what concerns the sensors, the exteroceptive type, so that interfacing with external environment, has not to be really performing. As a matter of fact the precise disposition of objects and obstacles allows the robot to complete tasks without sophisticated recognition algorithms or navigation.

One main task is for example an obstacle avoidance but in structured environments this can be performed simply by a gradient based algorithm which takes into consideration the a priori knowledge of the ambient and calculates the forces which drives the robot to the goal avoiding whatsoever thing in the middle.

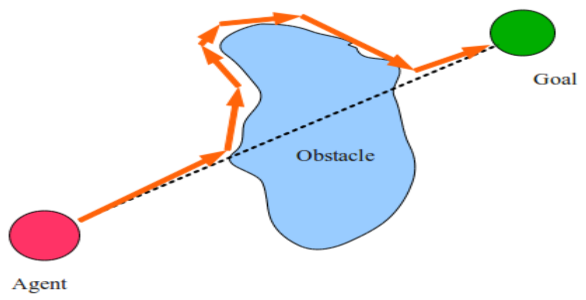


Figure 1.3: Navigation with obstacle avoidance

1.2.2 Unstructured environments

This type of environment differs a lot from the previous one both in terms of patterns that the robot can recognize and of consequence of tasks that can be performed.

Starting from the first consideration, an unstructured environment presents some element of ”confusion”. This is related to the casual disposition of objects, and so real obstacles from robot’s perspective, inside the space.

One easy example is the city environment where positions of people, cars and objects can vary continuously and without a pattern that can be evaluated with simple rules. In addition the presence of many lights of different kinds can make the robot become ”crazy”.

Another typical bad situation in which robots can perform their tasks is a work done in an outdoor ambient like mountain or wood that can be usually faced up to by military robots.

In this situation not only the presence of obstacles can make the robot’s life harder but also the harsh and rough ground with many steps and latches can reduce the operation possibilities.

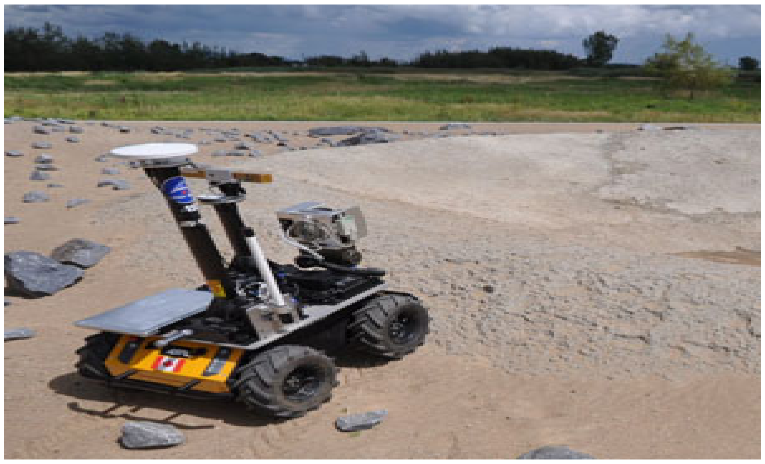


Figure 1.4: Example of unstructured environment

Finally it must be reminded that robots are taking a role of primary importance also in houseworks and old people-taking care. The house is for sure an unstructured one because the room changes are frequent and the people's presence is so important to justify the membership to the this category.

For sure in these complex situations the robot has to react in the best way it can do and to make this possible it must be built with the most advanced tools.

The principal distinction which can be made is still between performances of actuators and sensors. But differently from the structured ambient, in an unstructured one from the point of view of actuation a robot has to work well but not mandatorily with high velocities. In fact the tasks that the robot has to do in these conditions don't require any specific time but for sure they have to be taken to a final result. Also precision is not a fundamental requirement in these cases but this does not mean that it must be forgotten.

The precision is reached by the use of ad-hoc-made end effector which have to resists also to the worst atmospheric conditions, substances to touch and heaviest weights to lift.

For what concerns the sensors, the delicate aspect deals with the exteroceptive ones. Some examples are cameras or force sensors and not those that are useful for internal control of joints motion or in general to monitoring internal aspects of robots.

These exteroceptive sensors with which the robot has to be equipped must be calibrated perfectly and have to provide accurate and reliable measurements.

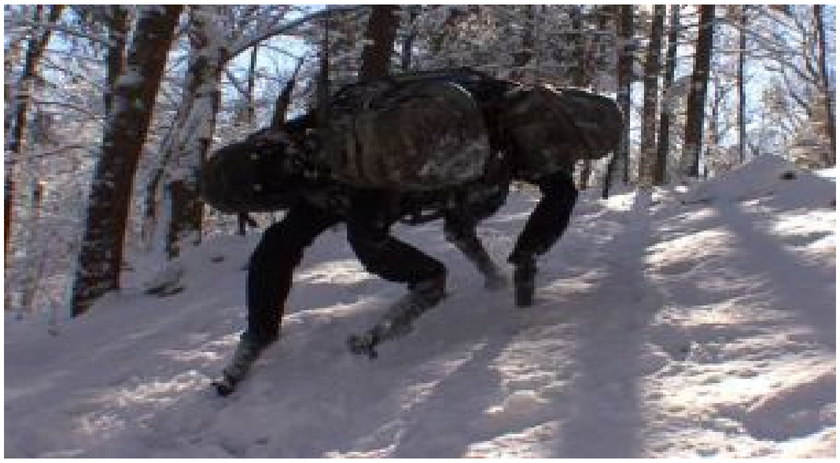


Figure 1.5: Dog military robot

The robot shown in the Figure 1.5 is the most complex robot built for hard workings in an unstructured ambient and it is equipped by the most advances sensorial instruments.

Dealing with the various tasks that can be performed in situations like the previously described ones, they can be of different type: motion planning, grasping objects, lifting weights and many more.

Motion planning for robots with many degrees of freedom is provably computationally difficult, even in highly structured environments, due to the high-dimensional configuration space.

Unstructured environments impose a number of additional difficulties for motion generation, when compared to the classical motion planning problem. In unstructured environments, a robot can only possess partial knowledge of its surroundings, objects can change their state unbeknownst to the robot, and manipulation tasks may require the end effector to move on a constrained trajectory rather than simply to reach a specific location. Each of these difficulties make the motion generation problem more difficult.

In unstructured environments, moreover, position and orientation are more difficult to be controlled, assumptions about colours and shades are difficult to justify, and the range of possible objects the robot can encounter is intractable.

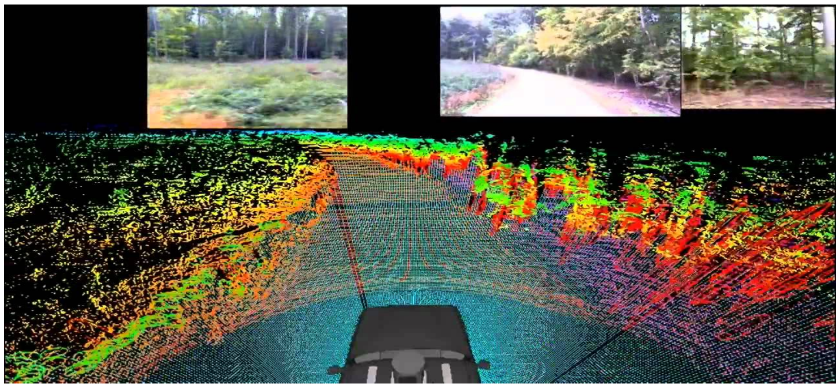


Figure 1.6: Example of vision task

To solve partially this last problem and to address perception in complex environment, robots must be able to reduce the state space that needs to be analysed.

1.2.3 The importance of human skills and perception in unstructured environments

After this analysis of the unstructured environment it is evident that it cannot be possible (and for sure not convenient) to work with a completely autonomous robot especially in some situations.

This is due to the fact that complex applications in unstructured environment still require the advanced decisional capacities of human operator.

But it is not all because also the most sophisticated robot cannot compete with the incredibly developed sensory capabilities of humans, as well as with the various possibilities of grasping and manipulating objects of all natures and strange forms.

From these considerations it is born the necessity to build always more autonomous robot but the component of communication with the operator must be maintained.

So the role of a particular technique called teleoperation is still fundamental because it allows to control and monitor the robot's work from remote and to overcome the issues that can happen.

1.3 Teleoperation in robotics

1.3.1 Brief overview of teleoperated systems

Teleoperation is a simple-to-see but really complex way to control devices or more specifically to make a robot moving and making actions to satisfy the assigned task.

By definition, teleoperation is the most standard term, used both in research and technical communities, for referring to operation at a distance. It could be applied to the most various devices but in particular to robot. An opposite concept is the "telepresence", a less standard term, which might refer to a whole range of existence or interaction that includes a remote connotation.

Historically talking the first approach to a sort of teleoperation method can be traced back even in late 1800 when the beginnings of radio communication were imputed to Nikola Tesla. He developed some of the first principles and systems to perform teleoperation.

Coming back to our days the history of modern teleoperation began at the end of the 1940s when the first master - slave manipulator was developed in the Argonne National Laboratory for chemical and nuclear material handling. After that, the development of teleoperation was fast. Adaptation of video technology to teleoperation made the first telepresence systems possible. Computer technology brought the advanced control loops into the remote end of the system, and finally brought virtual reality into teleoperation.

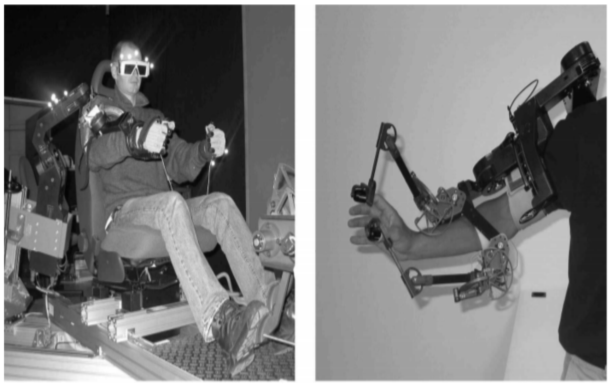


Figure 1.7: Example of haptic exoscheleton

The mechanical manipulators were soon replaced by electro mechanical servos. In 1954, Goertzs team developed the first electro mechanical manipulator.

One of the first areas where teleoperation techniques were utilized was deep-sea exploration. The deep oceans are even today regarded as so hostile that most of the deep-sea operations are made with remote controlled submarines.

Despite progress in advanced technologies, the traditional concept under teleoperation was based on the idea that the human operator would at all times be available to exercise more or less direct control.

Thinking of the most immediate example of this type of control, a teleoperation interface can be as simple as a common MMK (monitor-mouse-keyboard) interface. While this is not immersive, it is inexpensive. Those are usually driven by internet connections. A valuable modification to MMK is a joystick, which provides a more intuitive navigation scheme for planar robot movement.

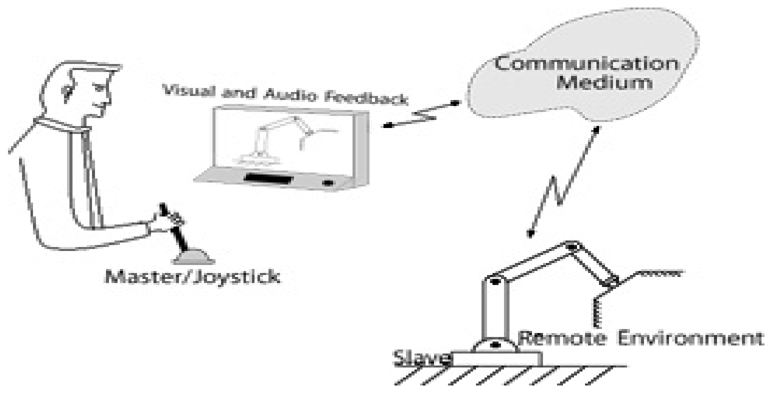


Figure 1.8: Simple example of teleoperation

Dealing with something more complex, examples of teleoperation can be found also in situation where antropomorphic robot have to be guided.

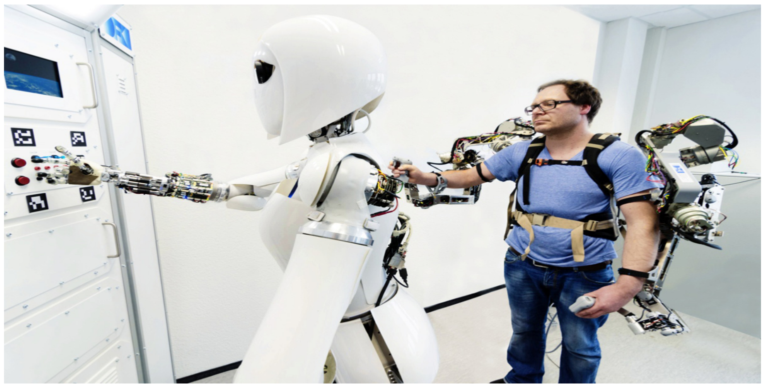


Figure 1.9: Example of teleoperated anthropomorphic robot

Two different branches which are based on the standard teleoperation are being developed nowadays: the shared technique and the supervisor one. For what concerns the shared approach the robot is considered semi-autonomous because it can do some tasks without human intervention while some other tasks can be only computed with the control of the user. These are usually applications with complex trajectories or those in which a not standard environment is expected. Talking about the supervisor technique, it can be used when robot should be completely autonomous and only a role of supervision or at most a sporadic correction of trajectory can be applied by the operator.

Detailing more the first technique it can be divided into two main categories depending on the grade of remote control from operator and the type of commands that he sends to the teleoperated device:

- Closed loop control (Direct teleoperation): The operator controls the actuators of the commanded device by direct (analog) signals. This is possible only when the delays in the control loop are minimal. A typical example of this is a radio controlled car.

- Coordinated teleoperation: The operator again controls the actuators, but now there is some internal control loop in the teleoperated robot. However, there is no autonomy included in the remote side. The remote loops are used only to close those control loops that the operator is unable to control because of the delay. A typical example of this is a teleoperated device for whom the speed control has a remote loop and, instead of controlling the throttle position, the operator gives a speed set point. Digital closed loop control systems almost always fall into this category.

1.3.2 Structure of a teleoperation system: master and slave robots

For what concerns the more technical aspects, the basic concept under all these techniques is the communication which have to be established between the device and a remote controller which sends commands to it.

In practise there's a relationship of master-slave between the operator and the controlled device. The system is formed by two parts, the control module, called cockpit and the telemanipulator, the slave robot at the remote location.

This last one has to receive packets of informations containing command as the desired trajectory to be followed or the coordinates of objects to be grasped and manipulated.

Those data are in general represented by set points for the motors present in the joints which have to move all the hardware links and this is an example of coordinated teleoperation discussed in the previous section.

Controlling commands are sent usually electrically by wire or radio/wireless. When the connection between the manipulator and operator is mechanical, the term "remote manipulation" means mechanical manipulation. In telemanipulation, the connection is electrical.

Detailing more the most widespread ways of keeping in touch with the robot they are transmission lines, radio wave, wireless, internet. Beside the different environments, a choice has to be made between communication protocols.

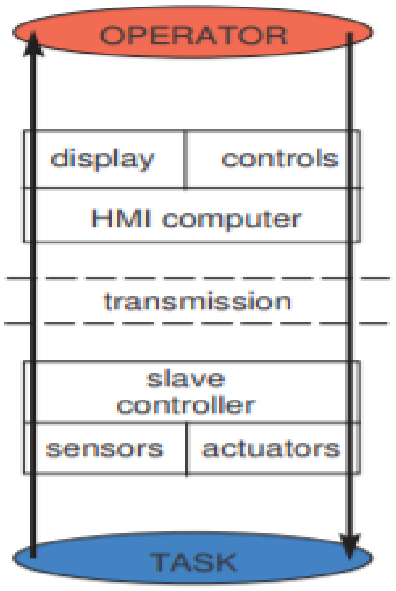


Figure 1.10: Teleoperation concept

In the figure above the part between operator and transmission is present in local environment while the remaining portion of the scheme is performed in remote environment which can be really far away from the operator.

The communication between the teleoperator and the teleoperated device is focused on the use of customised packages of informations depending on the particular application that the user has to encounter. Obviously one of the great problem in this type of control is the delay [1] between sending packets to the device and their reception. In other words the real time aspect is really difficult to be reached.

So this mechanism creates a sort of lag between the instant of sending command and the real output motion of the controlled device, fact that could take to an improper work of this one.

Another important and common issue of this technique is the correctness of the transmission. In fact not always the received packets are equal to those sent by remote controller because some information is missing. The necessary consequence is the increase of danger for environment and people around robot due to the fact that the commands result to be wrong and so the performed trajectories could be unpredictable.

[1]Transmission latency of the system

In practise packet loss is an inherent problem with most packet-switched networks due to several factors such as transmission time-outs, transmission errors and limited buffer size. Several reconstruction algorithms which address the issue take into consideration passivity perspective, that is, reconstructing lost samples while preserving passivity. Three policies can be applied in case packets are lost:

- using null packet replacement
- using previous packet
- using passive interpolation

1.3.3 Haptic interfaces as a master device in teleoperated systems

As discussed in previous sections, teleoperation is a useful and practical way of keeping in touch from remote with a device, or in particular with a robot, to which the operator sends command in order to control it and makes this to perform various tasks.

A fundamental addition to teleoperation system is represented by the feedback returned on the operator of what the robot is sensing during its activity.

Normally the feedback type is a force one and so thanks to a complex system of actuators and cables linking the control device to the operator body, it is possible to make him having a perception of the objects grasped or of the insidious environment that robot is facing up.

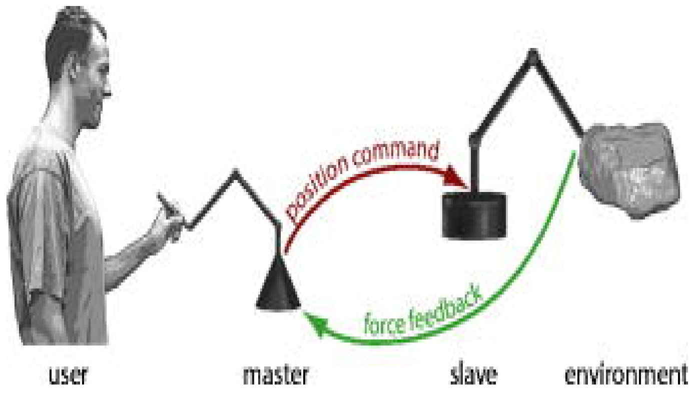


Figure 1.11: Haptic concept

In practise an haptic interface is a controlling device in terms of motion or manipulation action which also provides in the meanwhile the capability of obtaining a force feedback.

It realises a complete experience of being in the robot universe but from operator's "home" and not only the visual sense is taken into consideration but also the force one which can have a great importance in evaluating the best solutions to adopt in problematic situations or difficult tasks.

Those interfaces act as master in a master-slave action conception and a precise communication protocol has to be used depending on the particular robot's work.

As a matter of fact there are two big aspects which an haptic builder has not to forget and that are children of the same macro-problem: the delay of information sharing.

This is an already known Achilles heel from teleoperation but here the situation is even worse because the amount of data is doubled and so if a robust communication protocol is not provided, the sent informations could not arrive in time and problems of stability may happen. The second aspect is the precision of action that passes through the knowledge of operator of what the robot is sensing and so if data packages are lost due to a bad connection, serious consequences may be generated.

1.4 Range of application

The importance of teleoperation and haptic technologies in Robotics field is demonstrated by the fact that they cover a lot of possible applications where the presence of human can be thwarted by the harsh environment.

The principal fields in which the presence of teleoperated devices is required are:

- Exploration in Universe: robotic planetary exploration programs use spacecraft that are programmed by humans at ground stations, essentially achieving a long-time-delay form of telerobotic [2] operation. Recent noteworthy examples include the MER(Mars Exploration Rover) and the Curiosity rover.

[2]The shared or supervisor approach dedicated to robot is called telerobotics

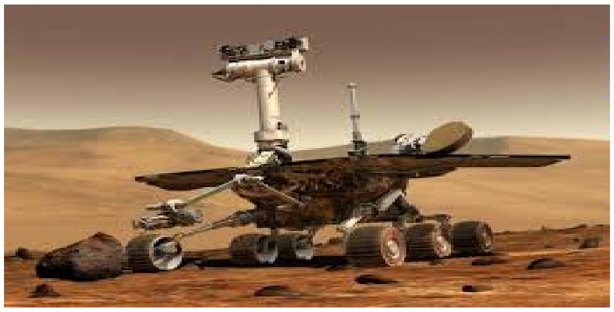


Figure 1.12: Curiosity robot

- Telemedicine: nowadays a lot of minimally invasive surgical interventions are made by the help of robots that can be more precise and avoid the problem of human hand tremor which can be really dangerous. Robots with optimal control can stay in the delicate part of patient without moving also for a long time.

Figure 1.13: Example of telemedicine

  Obviously the most fundamental issue to be faced up is the previously cited lag time between command and real operation of the robot. So these applications enter in the hard real time category.

- Applications in radioactive environment: in this case the teleoperation control allows to avoid the presence of an operator in those workplaces which can damage seriously its healthy.

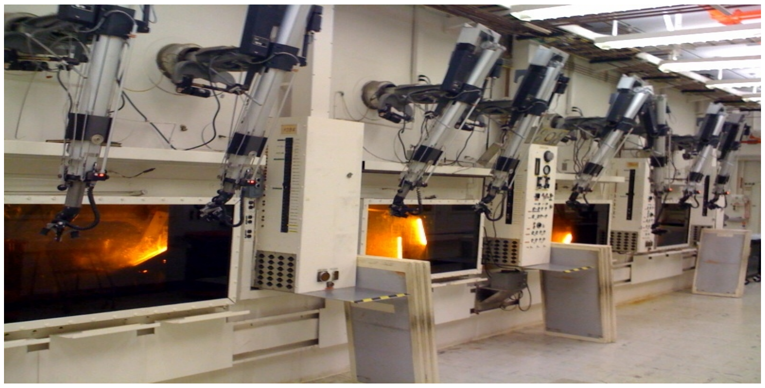


Figure 1.14: Example teleoperated robotic hand working in nuclear ambient

The precision required in this type of application is really high due to the dangerous materials to be manipulated and so special tool equips the robots.

2 Chapter: Haptic interfaces

2.1 Introduction

The role of Haptic interfaces is always more determinant in robots control field and the most various applications are performed with the use of this particular flexible tool.

But like the variability also the technical aspects' differentiation in terms of shapes, actuation and motion possibilities is interesting to be studied.

In the section below some types of haptic interfaces will be shown and their features will be evaluated.

2.2 Commercial haptic devices

A complete and common haptic interface usually includes one or several electromechanical transducers like sensors and actuators which are in contact with the operator in order to apply mechanical signals to distinct areas of the body creating the force feedback, and to measure other mechanical signals at the same distinct areas of the body to send informations to controlled devices. Whether these signals should refer to forces, displacements, or a combination of these, is still the object of debate. Another important part of a complete interface is the computational system driving the transducers. The function of this computational system is to provide haptic rendering capabilities.

Usually those fundamental components of haptic interfaces are mixed in a sort of multi-degree serial structure with many links and joints like that reported in Figure 2.1. This system allows to control perfectly a complex device which needs to have all possibilities of motion in workspace.

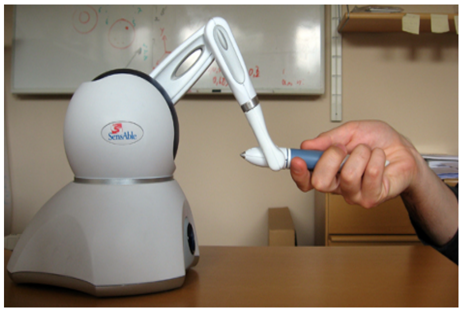


Figure 2.1: Example of haptic system for robotics

Another example of standard haptic device is the exoscheleton which can have big dimensions and embrace large part of human body but there could be smaller systems like those of simple hands.

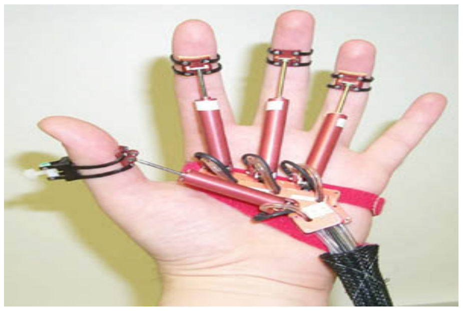


Figure 2.2: Example of haptic glove

In all these examples the common detail is the presence of an actuation made by electrical motors or hydraulic/pneumatic pistons.

Despite the constant spread of this kind of motion standard transmission, nowadays the market of haptic Interfaces has seen the birth of a new conception of transmission: the tendons.

Concerning the encumbrance that this type of transmission provides it is evident that some advantages are obtained with respect to standard solutions. Moreover the force-weight ratio guarantees a solid, useful and secure structure with a lot of potentiality in terms of flexibility of application.

2.3 Cable-driven haptic devices

Cable-based interfaces are promising candidates to solve limitations related to workspace, inertia and cost, at the expense of limited stiffness. The cable transmission minimizes the actuators contribution to the end-point inertia and encumbrance, providing a considerable force-weight ratio. The usage of cable transmissions is not a new concept in haptic interfaces design, as some wire-based haptic displays have been proposed in the literature.

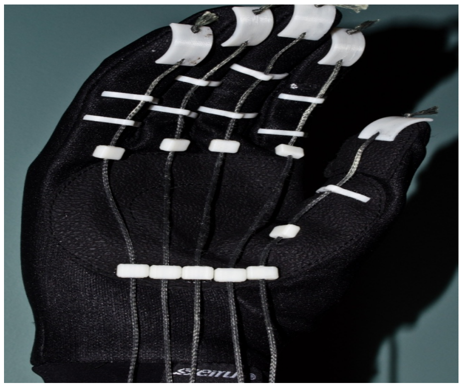


Figure 2.3: Example of haptic glove

Some existing technologies are wearable haptic interfaces based on parallel wires in an underactuated configuration which can help to address blind people and it is the basis of the three-cable haptic interface development.

A 4-wire driven 3-DoF planar haptic device exists, while other solutions with a 4 strings 3D spatial interface are very widespread. Over-actuated solutions for 6 DoF with 9 and 8 strings have the advantages of low-inertia, low-cost, and high safety.

2.4 Twisted string transmission vs standard transmissions

The smartest solution in terms of tendons transmission consists in twisting two or more strings of a particular material in order to obtain a compact and lightweight piece of rope. The created tendon is usually linked from one end to a small motor in order to obtain the twisting of the strands and in the opposite end the motion of the load.

There are many examples of use of this transmission typology. Just think about an application where heavy loads need to be taken from one side to another with a linear motion or where pulleys give motion to some device but they are put into motion by the displacement induced by strings' shortening.

But many other examples can be treated. In case of anthropomorphic robots this particular system of transmission is usually used to recreate the muscular system. As a matter of fact if a tendon is seen as linked to a sort of biceps [3] and another tendon is linked to triceps, it is easy to make the rotation of an artificial arm due to the combination of shortening and elongation of fibers like shown in Figure 2.4.

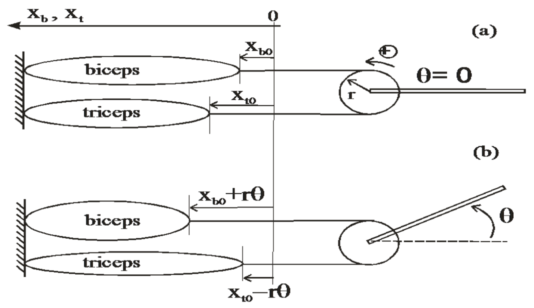


Figure 2.4: Example of artificial arm created by twisted string actuation

Moreover many cases of robotics hands are being developed because of the same strands' principle of twisting and shortening.

[3]the muscles are motors in the application

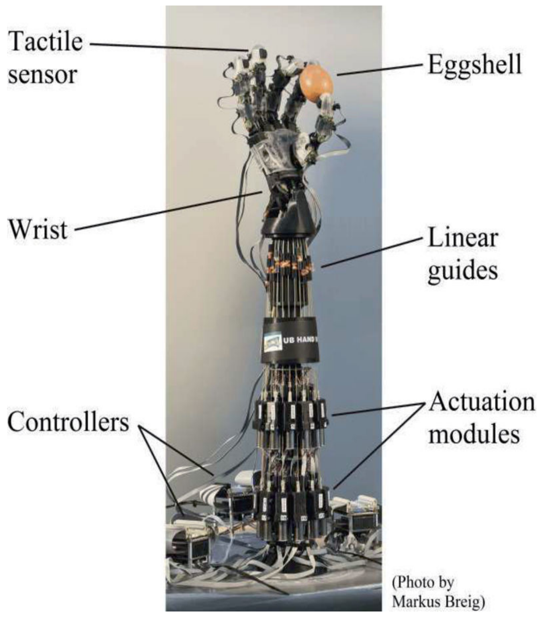


Figure 2.5: Example of artificial hand created by twisted string actuation

With respect to standard solutions of transmissions and those made by normal tendons, the main advantages of twisted string actuation system are very significant and justify its increasing use.

The most important ones are:

- the use of very small high-speed motors
- a direct connection between the motor and the tendon without any intermediate mechanisms such as gearboxes, pulleys, or ballscrews
- a direct conversion from rotational to linear motion
- nearly absent friction contribute(only an axial bearing is needed)
- a high reduction ratio

3 Chapter: Goal of the project and system description

3.1 Goal of the project

The thesis project wants to go in deep of teleoperation field with the final goal of developing some specific parts of an haptic Interface. In particular the project's aspects that have been deepen are:

1. The communication protocols analysis and the development of a firmware by which a motor can take informations and set points from PC and move the structure while sending back adequate force feedback.

2. The calibration of TSA actuator's force sensor.

3. The coding of the transpose Jacobian J^T to pass from workspace of robot to joint space with the aim of knowing which forces the TSA actuators have to apply in order to have the required force applied in the real environment.

4. The graphical description of the haptic system to let the user be aware of singularity positions of robot and so avoiding them with the motion of the haptic structure

3.2 Overall physical system description

One of the main project's aspects is to design an haptic interface able to move its end effector freely in the space under the intention of the operator and at the same time apply reaction forces in the Cartesian space along the three linear directions.

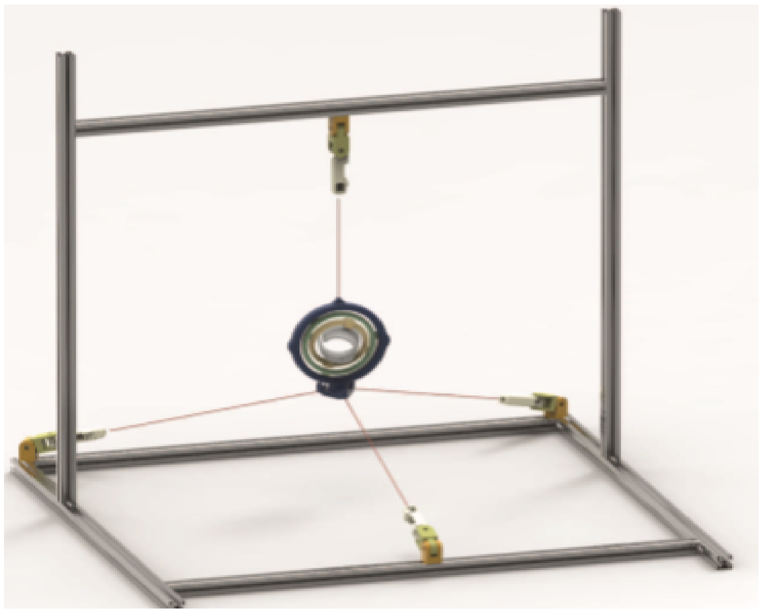


Figure 3.1: Cad rendering of cable driven haptic interface

The aim of this particular device is the control of a remote robot and the recreation of some force feedback sensation on the user body which represent the interaction forces of the telemanipulated robot directly to the working environment.

To make these feedback be sensed by the operator it is necessary that this one is in contact with the haptic system and to have this result a central bracelet where the forearm is inserted is designed. The device takes shape by three gimbals one mounted into the other and fixed by some orthogonal pivots. Thanks to this already described structure the bracelet can rotate by all three axes of rotations and so the operator is free to orient as he wants the wrist. Moreover this mechanical device leaves to him the freedom to use the hand to accomplish other tasks, such as teleoperating a robotic gripper.

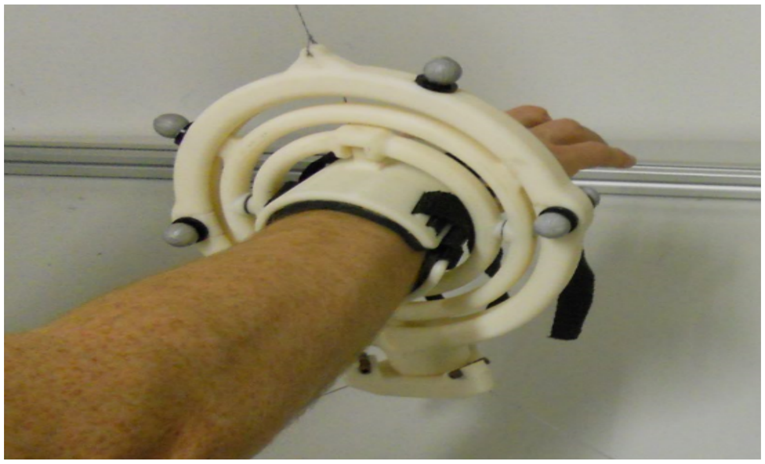


Figure 3.2: Operator forearm inside central bracelet

The central bracelet is then linked to a physical metallic structure that can be described as a large tetrahedron. But how to reach the connection?

This is possible thanks to the fact that some tendons are present and link the various extremities of the bracelet to those of the metallic structure. Of consequence the operator forearm is free to move in all space directions because it is suspended in the air and its unique constraints are the lengths of the tendons.

The large metallic scaffolding helps to sustain the whole tetrahedral architecture.

Talking about the useful workspace for the system it has to be underlined that due to the presence of tendons and that of the central structure in which the bracelet is inserted, the 6 DoF are achieved and so the whole workspace is touchable. However the number of motors is four because the central bracelet is moved by tendons themselves. In a situation like this, a minimum number of n + 1 actuators is necessary to control motion and forces in a n-dimensional space. For the project purpose it is interesting to control only movements and forces along the three Cartesian directions and so four actuators are enough. In particular the knowledge of the force feedback is available for the three Cartesian axis and this is the main result that the user wants to have for a complete and satisfying force sensation.

Tendons are responsible to create the sense of force feedback on the user due to their capacity of elongating and shortening themselves so producing a resistance on human forearm inserted in the bracelet.

The whole central structure and the bracelet were created by a 3D print using ABS plastic material also for a design matter in terms of weight.

Dealing with one single tendon(but it is the same for all the other) it can be noted that its extremities are linked to one anchor point of the bracelet at first side while on the second side a connection to the output shaft of the motor's module is provided.

The bracelet is obviously a mobile frame while the modules are considered as fixed frames because of their integration with the global scaffolding.

In reality the motion of the motors can be forgotten in total because they need to follow that of central bracelet to which they are linked. In practise their motion need to be only of rotation type thanks to the system architecture.

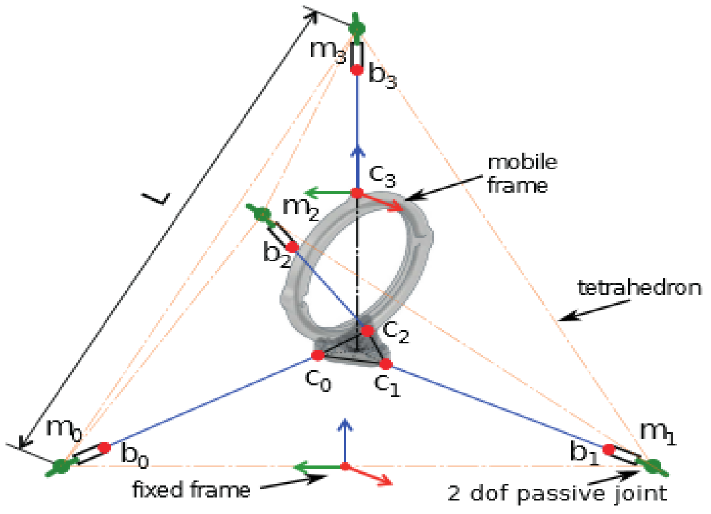


Figure 3.3: Connections between haptic interface mechanical parts

To allow the motor's module to be always aligned with the fixing points on the frame and the corresponding point on the mobile frame, an universal joint has been used to fix the TSA modules to the frame.

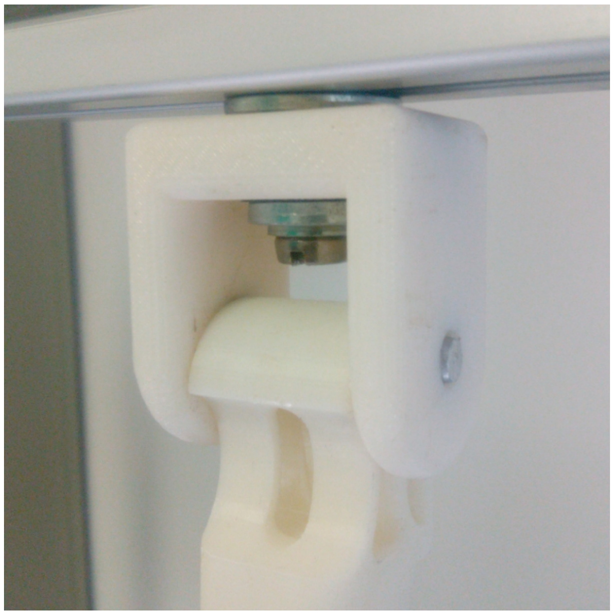


Figure 3.4: 2 Dof module joint

For the sake of completeness it must be said that the choice of putting the modules on the structure's extremities would allow to reach the whole available workspace if also all the tendons were connected to a single central anchor point in the bracelet. Moreover that idea would reduce to zero the torques generated on the bracelet by the tendons themselves.

Unfortunately that configuration is not practically implementable since the mobile frame has to hold at its middle the forearm.

For this reason there's the necessity to shift down the three connection points corresponding to the ones of the tetrahedron.

The real haptic structure is shown in the figure below

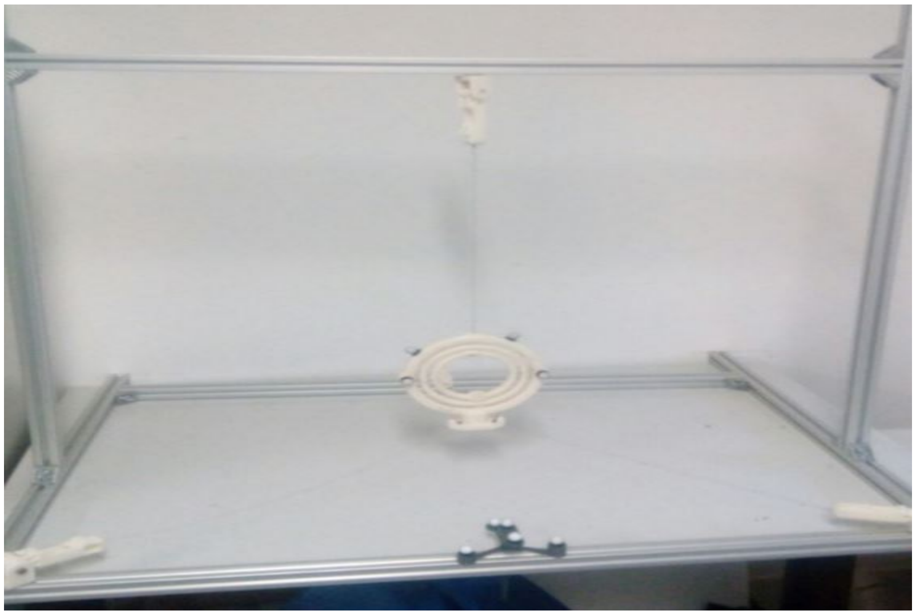


Figure 3.5: Real haptic interface structure and connections

After having described the overall structure of haptic system and focused on the mechanical features of the TSA module it is necessary a forward step: how the motor can perform its work and which data it has to process in order to realize position and force control.

The sequence of devices that have to talk each other is not really complex but some more details are provided in this section.

Starting from the haptic side, for sure the central bracelet is the main object to be controlled and to make it moving four tendons are present.

Focusing on one tendon, this is linked into the other extremity to the DC motor which is able to make it elongating or shortening by a wrapping action.

The motor needs the help of an H-bridge driver to take the necessary energy to perform its motion.

From its side the driver take as input some signals like PWM laws given by a dedicated controller (and of course there's one controller for each motor).

Now the chain is almost at the end because the information that the controller has to receive to perform its implemented position and force control algorithm, are the one transmitted by an upstream PC.

In this central workstation the Ros environment runs and the position or force operational modalities can be set by an operator and sent to the motor's controller.

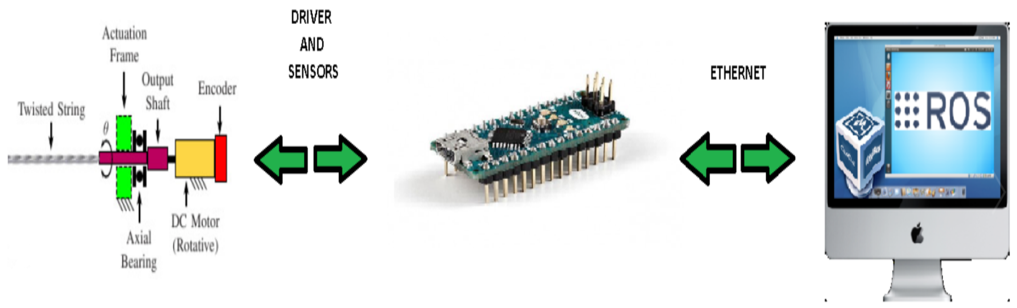


Figure 3.6: Scheme of talking devices

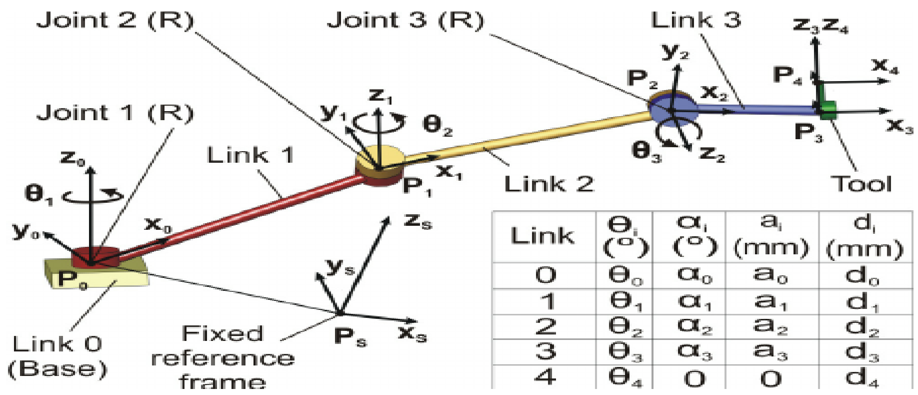
In practice the right-to-left sequence is used to transmit commands for the motor and to move haptic's end effector, while the opposite flux of information is used to let the operator know the bracelet's feedbacks.

3.3 Kinematic model

After having described the physical structure of the haptic interface it is important to evaluate the possibilities of motion and in order to do that the kinematic model needs to be derived from geometric considerations.

But to deal with this complex problem a general overview about the most common robot structure has to be done.

A robot is a system composed by many links connected by joints and the goal is to move an end-effector positioned at the terminal part of the chain by imposing some forces of actuation to the various joints. To know how the system can move and so how to position in a precise manner the end effector or having this applying a certain force on the environment, it can be possible to realize a full kinematic model which relates the motions of the links. A common approach is to use the Denavit-Hartenberg convention that ideally fixes some frames to the various links exploiting precise rules. Then, analysing how the frames have been positioned and which coordinates' transformations there are between the links due to the joints' movements, the kinematic model is computed.



Link	θ_i (°)	α_i (°)	a_i (mm)	d_i (mm)
0	θ_0	α_0	a_0	d_0
1	θ_1	α_1	a_1	d_1
2	θ_2	α_2	a_2	d_2
3	θ_3	α_3	a_3	d_3
4	θ_4	0	0	d_4

Figure 3.7: Denavit-Hartenberg to create kinematic model

This tool can be used to perform a direct computation of end-effector position variables in terms of joints coordinates but also the inverse passage and so, given the coordinates to which taking the end effector, computing the joints' variables.

Dealing with the actual problem of haptic interface, this system can be seen as a real robot because it shows all the fundamental components of a normal robot. In particular the end effector is the bracelet inserted in gimbals while the various joints are the TSA motors. Starting from their motions the central end effector can be positioned where ever the operator wants.

Now the real question is how to build the kinematic model form.

In order to answer a simple overlook on the haptic interface must be done. As a matter of fact it can be easily noted that the central bracelet is flying in the air because it is linked to four motors by twisted string concept as said in the previous chapters of system description. The tendons are attached to the base of the bracelet itself and on the other extremity to the final pin of the motors' module.

In this way it can be possible to notice that those linking extremities form two points in the space and the distance between them can be computed.

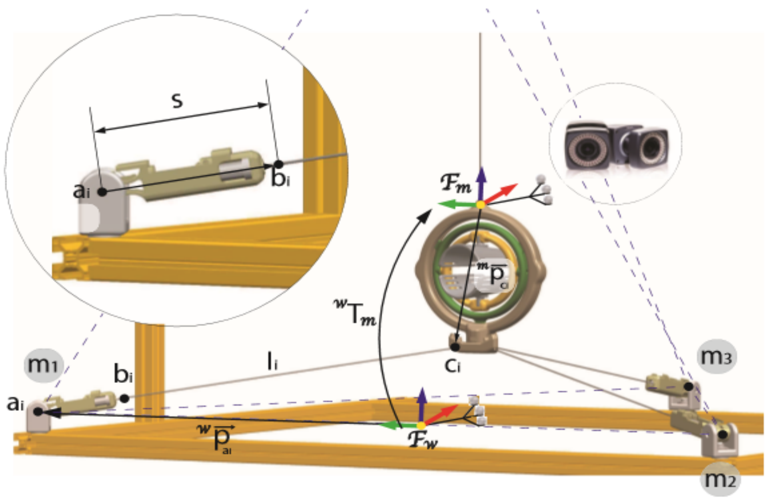


Figure 3.8: Crucial points to compute tendons' lengths

Considering only one motor and naming the meeting point of axes by which its module can rotate, a and the extremity in which the tendon is linked to the bracelet's base c, also called anchor point, these points have respectively coordinates $[xa, ya, za]$ and $[xc, yc, zc]$. They are obviously related to a central world frame Fw present on the structure as shown well in Figure 3.8.

In particular the fixed base points are expressed directly to the World reference frame while the anchor ones are expressed firstly in a Mobile reference frame called Fm integral to the bracelet but thanks to an Homogeneous transformation of coordinates it is possible to easily pass from the mobile frame to the world one.

How to take information online about those coordinates will be explained in next chapters.

So finally, basing on these tendons' information the inverse kinematic model is derived because knowing the desired position to which taking c, the tendon's length results to be:

$$l = \sqrt{(xc - xa)^2 + (yc - ya)^2 + (zc - za)^2} - s \tag{1}$$

where s is the length of the motor's module expressed in meters.

The direct kinematic model can be derived by inversion of Equation (1). Anyway, for the purposes of this work, the computation a closed form direct kinematic expression is not needed since the end-point position will be estimated by a motion tracking system.

3.4 TSA module

3.4.1 Mechanics of TSA

The tendons which are linked to the central structure of the haptic Interface can be moved by some DC motors inserted in special box made by ABS plastic material and created by a 3D rapid prototyping.

This small frame is customized for the purpose of accomodate the motor and provides some other adroitness in order to obtain a fully working system. One of them is the bearing between the end of the electrical machine and the plastic frame so that a reduced friction contribution is achieved during rotation of shaft and the possibility of damaging the motor during force transmission becomes low. Another important part of the whole box is the backward part which allows to connect the frame to the fixed scaffolding by the use of some prismatic and revolute joints.

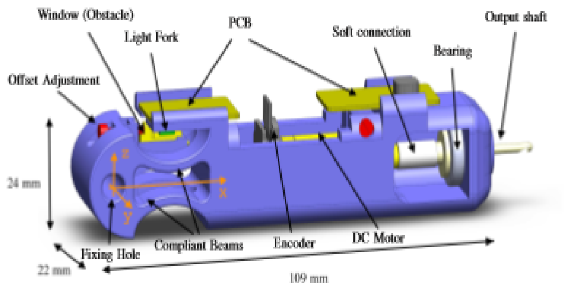


Figure 3.9: Actuator box

The particular device that jumps out from the union of the small motor and the tendon is called TSA [4] and represent a specific category of actuator not because of some particularity in the electrical machine itself but for the transmission type it provides.

[4]Twisted-String-Actuator called in this way for the particular link that the motor has with tendons able to shorten themselves

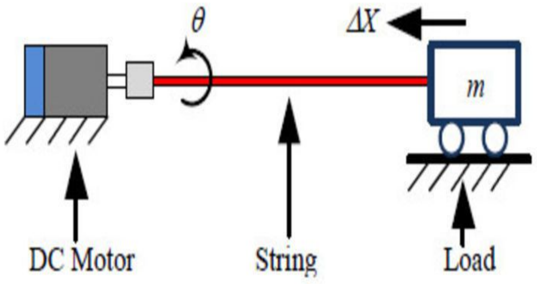


Figure 3.10: Twisted String Actuator principle

As a matter of fact, from the first point of view, they are simple DC machines of about 12 Watt with the normal feature of torque increasing with the square of armature current. In general haptic Interfaces with TSA actuation system use very small-dimensions motors which provides high speed and low torque.

One of the main TSA's strength point is the possibility to make the tendon twisting and so shortening. This fact allows the load attached to the other extremity of the string to move in the direction in which the tendon is shortening.

So the motion is linear but originally achievable through the rotation of the shaft of DC motor. In practise the system works thanks to a transformation between rotational and linear motion and considering also the forces at stake there's a significant reduction ratio which plays the role of increasing the torque sensed by the load reducing the speed. The result is obtained successfully without using mechanical parts like gears which would weigh down the entire actuation.

This last aspect is the real point of strength of the twisted string devices and it is the reason for which they are hugely spreading for many applications.

3.4.2 Tendons

Each 0.24 mm diameter tendon present in haptic structure is composed principally by two parallel filaments made of Dyneema material which is the world's strongest fiber. As a matter of fact it is made from Ultra High Molecular Weight Polyethylene (UHMWPE) and offers maximum strength with minimum weight.

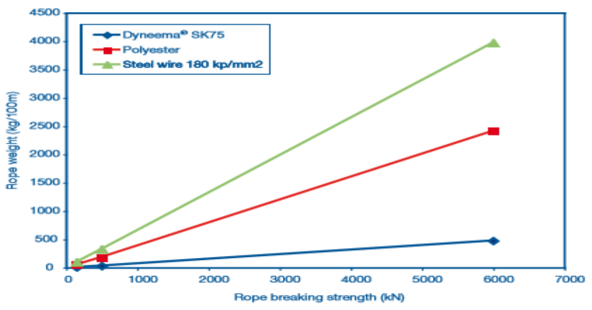
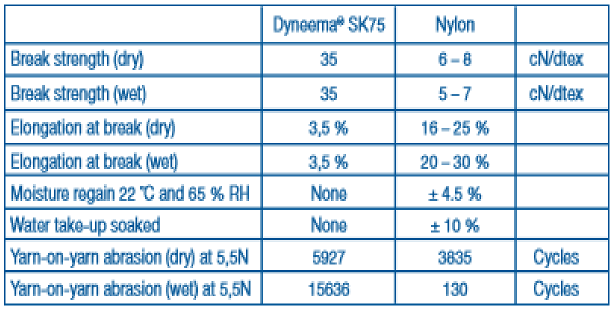


Figure 3.11: Table of comparison between fibers

This feature is really useful in a structure like that of haptic interface because firstly tendons have to face up to the internal stress in reaction to the high traction force imposed by the rapid twisting. Secondly they need to be light and flexible due to their action of linking the central structure and the motor so due to the need of flying in the air. As a matter of fact being more heavy would require thicker fibers and so the possibility of easily twisting would be lost.

The revolutionary features of this particular material can be used not only in "light" applications like that of moving a small structures as in case of haptic Interfaces but also in huge and "heavy" works. It is the case of marine applications or industrial environment where loads have to be lifted or moved in total safety for the operator.

Talking about the heavy marine sector, mooring lines made with Dyneema have proven to be a very workable solution. They are much lighter and easier to handle than other types. They are as strong as steel-wire lines of the same diameter while they are less than one-seventh the weight. Furthermore taking into account the thickness of the fibers, comparing those with same strength it is evident the convenience to use Dyneema because it ensures 40 per cent less diameter and 70 per cent less weight.



	Dyneema® SK75	Nylon	
Break strength (dry)	35	6 – 8	cN/dtex
Break strength (wet)	35	5 – 7	cN/dtex
Elongation at break (dry)	3,5 %	16 – 25 %	
Elongation at break (wet)	3,5 %	20 – 30 %	
Moisture regain 22 °C and 65 % RH	None	± 4.5 %	
Water take-up soaked	None	± 10 %	
Yarn-on-yarn abrasion (dry) at 5,5N	5927	3835	Cycles
Yarn-on-yarn abrasion (wet) at 5,5N	15636	130	Cycles

Figure 3.12: Properties of various fiber

In the industrial environment the material is used in ropes for lifting applications. In particular lightweight slings with Dyneema are ideal for repetitive ones. They can be easily handled and quickly placed around the load, enabling faster, more productive lifting.

For what concerns the mechanics of this particular way of power transmission, it is evident that the principal analysis will be focused on the kinematic and dynamics of the twisting phase of the two sections of tendon.

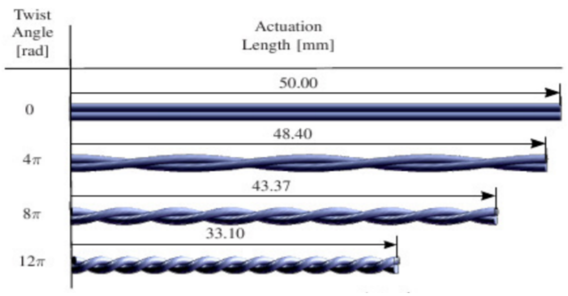


Figure 3.13: Shortening of tendons linked to TSA actuators

These considerations need to take into account respectively the position of the load moved by the shortening principle of the fibers and the axial force produced by TSA device.

Dealing with the motion of the load it is a linear one achieved by the conversion from the rotative one of the associated DC motor.

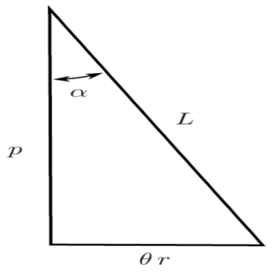


Figure 3.14: Kinematic of tendons

Calling θ the motor angular position and L the total length of the untwisted strand, the position p of the load is computed inverting the relation:

$$L = \sqrt{\theta^2 * r^2 + p^2} \tag{2}$$

In practise the solution is given by a simple geometric approach using Pitagora's theorem

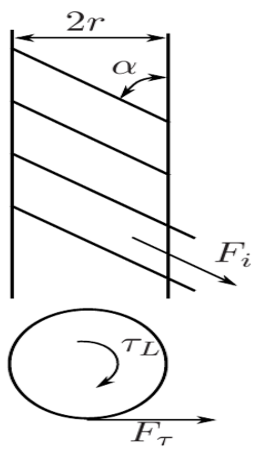


Figure 3.15: Dinamics of tendons

Referring to Figure 3.15, taking into account the forces acting on the system, the resulting total axial force Fz acting on the transmission system is:

$$F_z = n * F_i * \cos\alpha = \frac{\tau_l}{r * \tan\alpha} \tag{3}$$

where n is the number of the strand which link the motor and the load, F_i is the force acting on each fiber and α is the angle of twisting.

Until now one important assumption has been done: the absence of stiffness of the fibers. But in order to have a more complete analysis of mechanics of tendons, this needs to be considered. The strands are assumed to act as linear springs, with the capability of resisting to tensile forces only and not compressive forces, as in standard cable-based transmission systems. An important fact is that the total length of a strand L changes with respect to the unloaded length $L0$, depending on the fiber tension Fi and the strand stiffness K. In conclusion the force acting on each strand is computed as:

$$F_i = \frac{K}{L_0} * (L - L_0) \tag{4}$$

where L is obtained by the kinematic analysis.

3.5 System architecture: TSA controller

3.5.1 Arduino Nano

An Arduino is an open-source control platform based on flexible, easy-to-use hardware. It can be used for a huge variety of projects in which a system has to be controlled. So it can be said that for most of custom control applications or activity researches it is the most indicated and flexible architecture available on the market.

Arduino is programmed using its specific IDE which is Arduino IDE. It provides to the user many libraries for the most various projects starting from the simple goal of making a LED blinking to the complex application of controlling a kinematic structure with many links. So it is a very general purpose controller which can exploit many libraries. For this specific thesis project the most useful ones are *Ethernet.h* and *UDP.h* for the communication creation with the central PC and the *PID.h* one to control tendons and making the bracelet moving.

One more library that has been introduced for this particular project is *Thread.h*. This allows to divide the code in as many threads as the user wants temporizing each one to have a better division of computation by the Arduino CPU. For our purposes using this library is very useful because it allows to divide the two parts of the code devoted respectively to the packet interpretation while saving important data for motor and that where feedbacks are sent to PC.

This division aspect takes to a major reduction of lag between the generation of packet and motor real actuation because there is no interference with the opposite direction of data transmission.

Going more in detail about the particular platform of Arduino controller family that is used in the haptic project, this is an Arduino Nano.

Nano controller is called so due to its small dimensions that provide the same functionalities and computational capabilities of their "brothers" controller of bigger dimensions.

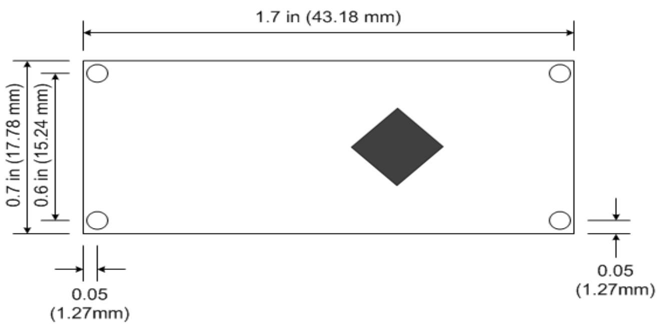


Figure 3.16: Arduino Nano dimensions

Obviously due to its practical feature it can be mounted on the single motor present in the haptic system and so there are four Arduino Nano in total, each of these mounted on its respective actuator.

One fundamental aspect to consider is that the basic unit of transmission using UDP protocol on Ethernet is the byte so all the data involved in this communication are sent as bytes packets. In case there would be some different type data like float or double, they have to be converted in an array of bytes.

Arduino Nano acts as slave controller and receives packets of informations directly from PC. This is made possible by an Ethernet connection with computer through assembling Arduino with a special EthernetShield which enables the device to receive and send information through Ethernet way of communication.

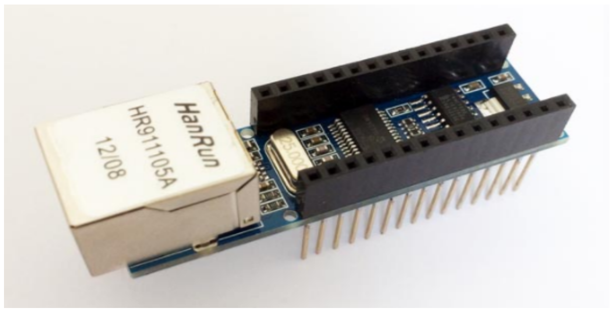


Figure 3.17: Ethernet Shield

The aim of this small platform is to implement the real control of the motor to which it is associated. As a matter of fact the system needs four different motors and of consequence four associated controllers to create the tetrahedron structure.

Each controller has developed inside the same PID scheme but obviously with different Kp, Ki, Kd gains depending on the particular hardware on which it is working on. In case of haptic project the four TSA differ a bit in some parameters and for this reason there's the necessity to calibrate rigorously each controller to obtain similar outputs in terms of time response and precision at steady state.

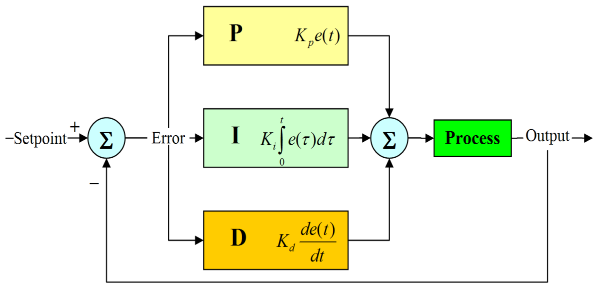


Figure 3.18: Scheme of PID control

As usual the proportional term is the first responsible of the set-point's tracking and it allows to have a certain quantity of robustness to noise. Using only this term creates oscillations until they are cancelled by the damping properties of the controlled system. To obtain a better response it is useful to use also derivative and integral terms. The derivative term is used to reach the steady state in as less time as possible with a good transient while the integrator is useful for the principal aim of the control which is tracking the reference with zero error at steady state.

In particular this last term is fundamental in the control of TSA actuators because the bracelet's force or positioning and of consequence of the whole controlled robotic device, needs to be really precise.

For what concerns the error on which the controller has to act, it is computed as the negative feedback sum between outputs of the motor and a specific set point. The provenance of the output signals is given by a measurement of sensors like the encoder for the position and an optoelectronic sensor for the force, while the set points are made available directly from the PC to which Arduino is linked using Ethernet.

The code for the PID creation is a little portion of the Arduino firmware while the greatest part is devoted to two main goals. Firstly to receive the packet from the user definition and to handle it in a manner that useful data are used by PID function while secondly to manage signals of encoder and force sensor measurements to be used in feedback control loop and sent back to PC.

3.5.2 I/O description

As mentioned until now the main work of Arduino controller is that of receiving packets of information from the Ros workstation, implementing position and force control algorithm and sending the feedback to the operator. The passage and transfer of data between the various devices is allowed by the presence of Ethernet communication which is possible due to the link of Arduino with the Ethernet cable present in the laboratory.

This cable can be seen as an input for Arduino but a more interesting aspect to be analysed is the real I/O pin system provided by this controller.

It has many pins of input-output to perform the standard functionalities of receiving feedback measurements, elaborate them to compute a control law and to produce an output.

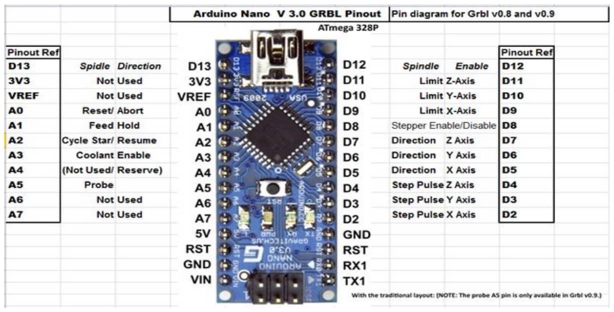


Figure 3.19: Arduino Nano pinout

The main division between pins are in digital and analogic ones. The first category is used to manage signals which can be put in a discrete range of values while the second one allows to treat continuous signals of which it is important to evaluate all the temporal flux and not only some "steps".

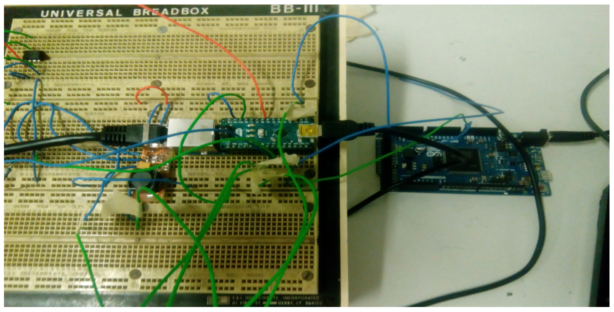


Figure 3.20: Arduino Nano electrical links

For this reason it is understandable why one digital pin (in particular number 3) has been chosen to acquire position feedback data thanks to encoder sensor. As a matter of fact this last one gives the result as "encoder tics" which are discrete finite values.

On the other hand the Arduino's acquisition of the force data sensor is performed by an analogic pin because of the necessity to evaluate all the variations in time of the signal.

These already described are the main inputs of the controller but in reality one more pin has to be described: the so called Vin. This is responsible to take in entrance the required energy to work properly. This is quantified in 5 volts and not the unique 3 volts that the connection with an external device through the USB port can give.

In the provisional setup built to validate the position and force control, the necessary quantity of energy taken by Arduino Nano is given by a second linked Arduino, the Due model, which is directly connected to an energy source.

For what concerns the output pins they are principally:

- the PWM [5] law computed by the control algorithm and used to drive the motor's shaft
- the direction command to let the motor rotating in the logical direction depending on the positive or negative value of the PWM law

The first output is of analogic type and set on pin 5 while the second one is of digital type and set on pin 6.

Those outputs are useful to make the motor working properly thanks to the presence of driver which takes the command in terms of duty cycle [6] from the computed law and transform it in a current signal for the TSA actuator.

3.5.3 Driver

The driver which effectively gives current to the TSA is composed by an electronic card, model MC33887 which is an integrated circuit of NXP.

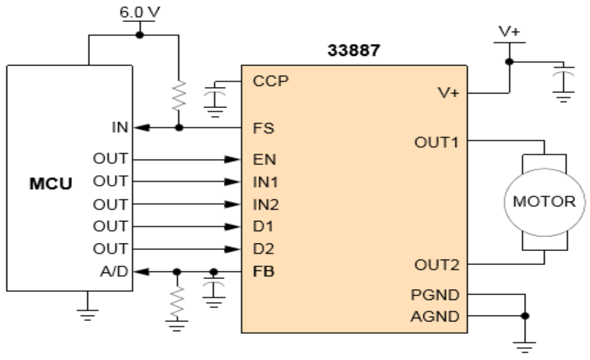
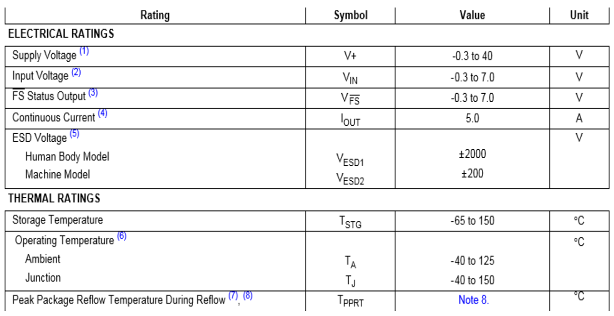


Figure 3.21: Scheme of MC 33887 driver

[5]Pulse-Width-Modulation

[6]Percentage of on and off pulsing signal



Rating	Symbol	Value	Unit
ELECTRICAL RATINGS			
Supply Voltage [1]	V+	-0.3 to 40	V
Input Voltage [2]	V_{IN}	-0.3 to 7.0	V
$\overline{FS}$ Status Output [3]	$V_{\overline{FS}}$	-0.3 to 7.0	V
Continuous Current [4]	I_{OUT}	5.0	A
ESD Voltage [5]			V
Human Body Model	V_{ESD1}	±2000	
Machine Model	V_{ESD2}	±200	
THERMAL RATINGS			
Storage Temperature	T_{STG}	-65 to 150	°C
Operating Temperature [6]			°C
Ambient	T_A	-40 to 125	
Junction	T_J	-40 to 150	
Peak Package Reflow Temperature During Reflow [7], [8]	T_{PPRT}	Note 8.	°C

Figure 3.22: Features of MC 33887 driver

By its configuration of H-bridge it receives a voltage reference modulated at PWM and gives to the DC motor the current needed to reach the wanted position or force. The voltage reference is computed by the PID schemes implemented in the controller of motors.

3.5.4 Communication protocol

The first idea of communication creation between the various members of haptic Interface was a bit complicated and saw a lot of hardware involved: a principal PC, on which Ros [7] programming environment can run, had to dialogue with a central controller called Arduino Uno which was the master distributer of information to single slave motors controlled by their Arduino Nano. The operator had to insert some data like operational mode and set points for the motors in a piece of Ros code which needed to be sent to central Arduino by a Ethernet way of communication using UDP protocol. Then this device interpreted the package and sort it to the corresponding motor by a I^2C transmission which is a serial one and allows to send only byte after byte.

Obviously in order to have a useful and complete supervision of the system work by an external operator, each single motor needed to give feedbacks of position and force to the central PC and also these informations followed the same path of transmission but in the contrary versus so firstly from Arduino Nano to Uno by I^2C and then from Uno to PC by UDP sockets.

The previously described setup and way of communication was good until a not minor detail had to take into account: the work speed of the central Arduino. As a matter of fact if we focus on the transmission of only one

[7]Robot-Operating-System

package to and from a single motor there could be not so many problems. The unique one can be the capability of sending and receive informations of the I^2C which works as a serial line and so it can be not so fast to handle both versus of communication if packages are sent from PC at several dozens of HZ and feedbacks are received at the same frequency. Just for this reason an idea of changing setup was already met. Moreover there is the real serious problem: how to handle by a central single controller the data mass of four motors at a high rate? .

For sure lots of lag would be present in the communication due to the fact that the master have to interpret each single packet and decide to which slave sending and last but not least the serial communication would be very slow.

Thinking about a possible solution to this problematic it has been selected the idea of eliminating the central controller and to address data packets directly to respective motors in order to avoid a transmission passage. At this point the UDP protocol on Ethernet takes on a role of primary importance because it is the unique responsible of making PC and motors talking one to each other.

For the sake of completeness it must be remembered that there exist two principal communication protocols: TCP and UDP.

They are quite similar from point of view of informations' transport but differ one from the other by means of two features. The first is the possibility for UDP to loose some informations because it doesn't have the feedback return to communicate that the whole transmission has been successful. The second consists in the transport velocity: UDP are very fast because they have to transmit small information quantity while TCP are slower due to higher quantity involved.

For the previously described aspects the choice was made on UDP communication.

In order to obtain the new setup some EthernetShield specific for Arduino Nano has been bought and mounted on each slave controller. Obviously a targeted communication opening must be created between computer and each Arduino but it will be discussed later.

In practise the final communication and control system in haptic Interface is made by the presence of the four slave motors commanded by relative Arduino Nano controllers which talk directly to central PC thanks to Ethernet communication.

This last one enables to exchange with PC the operational modes and set points in one direction while transmitting feedbacks in the other opposite direction.

In the next section it will be discussed in details how all this system can work and arrive to have a precise control of motors in order to realize the robot application and to simulate the force feeling on human body.

3.5.5 TSA firmware

For the thesis purpose Arduino Nano is used principally for three goals:

- Receiving set points from the user which wants the TSA to follow the desired reference trajectory
- Implementing the PID schemes in order to have a precise and robust control of the tendons and of consequence of the central structure of the haptic system.
- Sending feedback to PC to be visualized as video prints.

From the point of view of operational modes, these are basically represented each one by a number and allows to change the type of control which Arduino Nano has to apply on the tendons and of consequence to the bracelet of haptic structure.

The three possible operational mode are:

- 0: OFF Control
- 1: POSITION Control
- 2: FORCE Control

For a matter of completeness it is useful to underline that the most used control is the force one. This can be explained by the fact that the usual haptic Interface's goal is to realize the force perception and to arrive to target it is important to create vibrations on human body. The way to do this is controlling in force the motors which apply to central bracelet four tensions able to generate that feeling.

In order to detail better how the informations are managed by firmware code, it is useful to remember that Arduino can use the library *Thread.h*. So in order to divide the two main tasks of receiving packets while interpreting them and that of sending feedback, the firmware is split into two main threads running at different rates.

One thread is the main *loop*() and inside it the handling of packet interpretation to extract useful information is coded while for what concerns feedback sending part it is created a *threadCallback*() function which runs at a custom rate decided by user.

One note which has to be made is that instead at first sight controlling the device could result the highest priority task, in reality the feedback check by the operator should be more important to have a better supervision of what robot is sensing and facing up during its work. For this reason normally the feedback calculation and sending thread must be run with higher rate.

Starting to describe in detail the firmware code it is not possible to avoid to talk about *setup*() part. In this section the useful actions to initialize the whole project are computed. First of all there is the creation of two main UDP sockets: `UDP_Receiver` is dedicated to transmit packet from PC to Arduino while `UDP_Sender` is aimed to send feedback in the opposite direction so from Arduino to the user. The opening of this two sockets is possible by the specification of the IP address of the controller and the PC but principally the ports on which they have to listen on. These must be specified in some part of the code, usually at the top as it was made in this case.

The ports are the essential means by which to TCP and UDP protocols to manage multiple data streams through a single physical connection to the data transmission network like Ethernet.

By comparison with real life we imagine to send a letter to a friend. If the recipient lived in a single house and it was the only tenant on the envelope would simply indicate its address. This situation, however, is rather unusual because, more likely, he will live in a building with other tenants or share the house with other relatives. Each of these can receive mail at the same address and then to uniquely identify the recipient you will also need to use the full name which is composed by the address plus port. Likewise we will indicate on the envelope information about the sender in order to receive a reply.

A similar thing happens for network communications through the TCP / IP protocol. Each machine will be identified on the network by an IP address, but due to the fact that usually the same machine can provide different services, it is necessary to find a method for separating the individual data streams and direct them towards the correct management program.

The problem is solved through the ports, in comparison with the previous example, take the place of the name of the sender and recipient. Streams of separate data within the same machine are characterized from different ports.

After this UDP sockets' opening the setup provides also the physical connection of Arduino's pins to motor's connector in order to apply the PID control and read correctly the encoder. Moreover in setup the enabling of interrupts is made.

Now it's time to talk about the core of the firmware which is composed by the two previously cited threads. Going in detail of $loop()$ it sees the succession of :

1. Reception of the incoming packet
2. Interpretation of the packet
3. Use of some data for the PID law computation

The part of code devoted to reception is principally made by two standard library function of Arduino: $UDP.parsepacket()$ which checks for the presence of a UDP packet, and reports the size and $UDP.read()$ which saves the packet in a buffer of adequate dimension.

Packet dimension is 20 bytes and it is composed by:

- character $'M'$
- 1 byte for operational mode
- character $'C'$
- 4 bytes of command which is the set point and originally is of float type
- character $'G'$
- 4 bytes for each gain of PID so in total 12 bytes

Then this packet is passed inside a loop finalized to check each character. The loop take each byte of the packet and passes it as parameter to a function called $handleCommand()$ which, depending on the character $'M'$, $'C'$, $'G'$, run a custom FSM.

This finite state machine is coded inside a function called $handleData()$ which provides various states in which the useful bytes of packet are stored in an array of dimension 17 bytes called `data_array`. This array must have smaller dimensions with respect to the original one because some information like those of the characters are not useful to build up a control algorithm of the motor but they are only used to pass from one state to another inside the FSM.

Obviously the user can have built the package in a wrong way so inside the loop of check of bytes a sort of safety mechanism is present. In particular by the variable $isValidPacket$, set true every time a new packet arrives and false in a particular `State_Error`, it is possible to exit from the loop because if a character is wrong the state becomes that of error and in this last one the variable is set to false, condition for which the *while* condition of the loop is no more true so no other interpretation can be carried on.

Once the whole packet has been interpreted it is passed finally to a function called `update_mode()` where the magic happens. As a matter of fact thanks to the standard library function *memcpy* to which parameters of destination variable, source array and length of data are passed ,`update_mode()` is able to copy the values of different bytes length from the location pointed to by source, directly to the memory block pointed to by destination. In practise this tool is useful to take the interpreted array `data_array` and save some bytes as *setPoint* and Kp, Ki, Kd fields depending on which byte of *mode* it has been chosen by the user.

Finally those new variables are passed to a PID constructor which instantiate an object of PID type in order to apply the control to the motor.

Because of the PID class needs also the feedback output value of the motor to compute the error

$$e = setpoint - feedback \tag{5}$$

it is necessary to compute feedback. This is made in apart functions which have to run independently from the two threads.

This in not meant to have a third active thread but only a single functions which work as ISR [8] and so exploit the concept of interrupts. When one of them happens, the function performs the calculation implemented inside it. Then the value is converted in rounds of the shaft by using a term of resolution of encoder.

[8]Interrupt-Service-Routine

This final value is used by the PID object as term of feedback and so the constructor is complete and ready to compute the control law.

Previously a second thread called *threadCallback*() has been cited. It is used not to compute any calculation or interpretation of data but its goal is uniquely that of sending back feedbacks of position, force and current to the PC which is a fundamental task, too. In order to be able to do so, three other functions *memcpy* are used to pass to a buffer destination the three feedbacks stored in three variables: the position one `pos_feedback` computed by one ISR and the other two `force_feedback` and `curr_feedback`. Each of these is 4 bytes large so the total dimension of feedback il 13 bytes because there's the adding of a char character which identifies the number of motor to which feedback are related.

The final buffer is sent using the `UDP_Sender` instance which performs the successive standard functions *UDP.beginPacket*(), *UDP.write*() and *UDP.endPacket*.

Now a simple but basic question can be made: how the packet is prepared and sent by PC? The answer is in the next section.

3.6 System architecture: Workstation

3.6.1 What is Ros

Ros [9] is the principal helper for software developers which are able to create their own applications to be run on robots or in general on automated devices. Despite its name it doesn't provide the functionalities of a real and proper operating system but it can be better analysed as a programming environment for the creation of robotic custom control algorithm.

Ros provides to the user many helpful libraries and various features like hardware abstraction, device drivers, libraries, visualizers, message-passing, package management. For its possibility and flexibility of interfacing with all kind of robots it is for sure the most widespread environment in the field of robot programming.

The Ros' principle of usage is based on the presence of two main categories of tools: nodes and topics.

Nodes are the most various pieces of code where some robotic application are developed, while topics are a sort of particular channel through which

[9] Robot-Operating-System

nodes can share informations. Of consequence Ros can be seen as a peer-to-peer middleware package. As a matter of fact it acts as a linker of some nodes that want to establish a communication based on an exchange procedure of many different types of messages.

In the figure below it is represented a procedure of communication establishing and it must be underlined the presence of a special node called Ros Master which need to be launched before the other nodes. In fact it is responsible of managing the linking initial phase between programs.

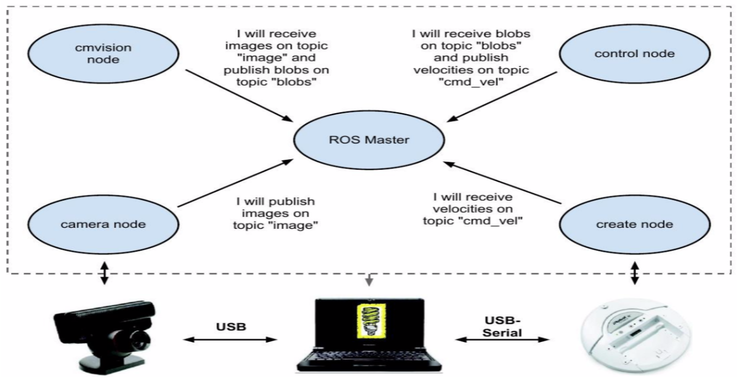

cmvision node
I will receive images on topic "image" and publish blobs on topic "blobs"
I will receive blobs on topic "blobs" and publish velocities on topic "cmd_vel"
control node
ROS Master
camera node
I will publish images on topic "image"
I will receive velocities on topic "cmd_vel"
create node
USB
USB-Serial

Figure 3.23: Establishing communication between nodes

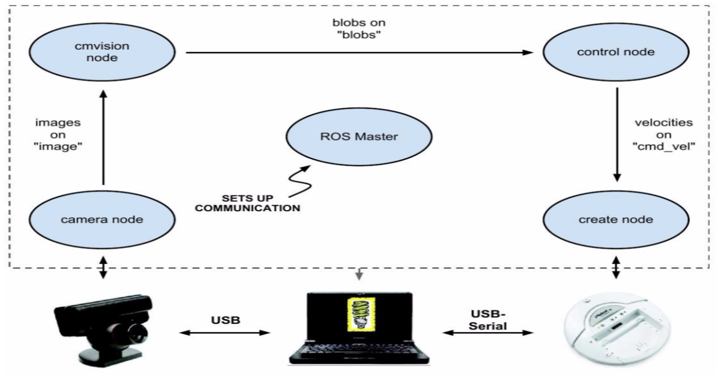


Figure 3.24: Established communication between nodes

Usually the total robot functionality is performed by using different nodes because each single smaller goal is developed in a different node. Of course they have to talk to each other and this is possible by the presence of topics channels but they remain separate entities. For example two distinct nodes can run on different computers but communicate over the same topic or over various topics. On the other hand nodes can run on the same terminal which talk to the controlled robot and this situation is really frequent in robotic field.

The dialogue provided by topics between different machines is possible thanks to TCP or UDP protocols. They are quite similar from information transport point of view but differ one from the other by means of two features. The first is the possibility for UDP to loose some information because it doesn't have the feedback return to communicate that the whole transmission has been successful. The second consists in the transport velocity: UDP are very fast because they have to transmit small information quantity while TCP are slower due to higher quantity involved.

As it was previously reported, the main aim of nodes inside Ros environment is that of performing some piece of global application by exchanging information with other nodes and providing computation. This work is made under a general scheme of publisher-subscriber relationship: usually a node is a publisher of some result of computations or service that it has brought to completion and the other nodes listen to it firstly waiting for informations and then sharing them. But it is not all because there are some examples in which many subscribers can listen to lots of publishers on a single topic.

But for sure one of the most used tool that Ros environment provides is Ros service. It is a specific publisher-subscriber mechanism in which a node (client) sends a request of operation to another node (server) and receives a response in return.

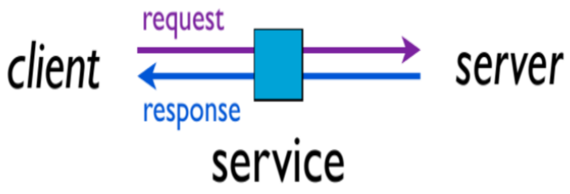


Figure 3.25: Client-Server mechanism

A service is called with a request structure and, in return, a response structure is created. This is quite similar to Remote Procedure Call (RPC).

Ros provides some standard services but a custom use of services is often required especially when working on personal robots.

3.6.2 Why Ros in haptic project

When the use of Arduino controller was presented, it was said that obviously in order to work by a feedback regulation, it needs some references on which basing to close the controlling loop. These set-points informations must come from the user and so from outside the Arduino device itself. So the idea was to exploit the idea of information managing and transmission of Ros based on nodes and topics to provide Arduino the adequate information it requires.

In practise the concept of nodes is particularly useful in case of the development of haptic interface. As a matter of fact it has been possible to create a package called `UBHaptic_ros_bridge` in which representing all the functionalities of the haptic system.

Inside the package some nodes are present and they dialogue one to each other.

The first node called `UBHaptic_ros_test` is the one with the goal of implementing the two principal activities of the system: sending commands and receiving feedbacks of force and, less important, of position. So it can be said that it regulates two main threads in which the overall goal is divided.

The second node represents the physical structure of haptic system and so it is implemented as a class called UBHaptic with some attributes and methods.

There are other three main nodes which are the attributes of the global class UBHaptic but in turn they are implemented as classes because they must provide their own methods and attributes.

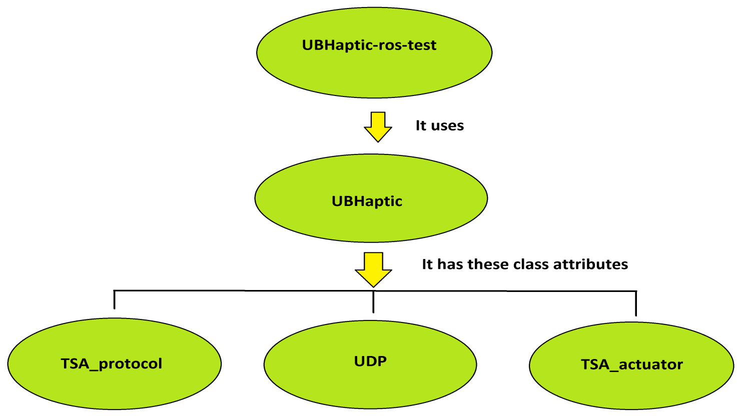


Figure 3.26: Haptic Ros Nodes

In conclusion, from the point of view of sending commands, the general structure of code is devoted to send informations through the use of one method of `UBHaptic_ros_test` and these commands will become the set-points for the Arduino controller basing on which it will compute the PID control scheme. From the point of view of feedback reception the second thread will be exploited.

3.6.3 Development of UBHaptic-ros-bridge

In this chapter the main features and details of the various haptic classes will be presented. In particular the various methods of each node will be specified.

The whole Ros code implementation has been developed thanks to the Netbeans IDE.

The upper level class of the whole chain for the reception of feedback and the command sending is the `UBHaptic_ros_test` class. It can be considered the father node because it handles the definition of commands to be sent to the system and manages the video print of the feedback information from the robot.

The code of `UBHaptic_ros_test` is mainly divided into two parts: the feedback reception function and the "*Main*". Going in further details it is important to be noticed that the $receivePublishFeedback$ is the responsible of taking information from the system by the methods of the child class UBHaptic and of printing them to video. This passage is really important because despite the command sending must be made correctly, the acquisition of returning feedback allows the operator to be aware of what robot is sensing and this is for sure the main goal of an haptic Interface. For this reason the `receive_rate` frequency of this function must be set in a way to have as many information as possible and it depends principally from the computation capacity of Arduino controller to elaborate data.

The second part of code of this class is the $Main$ and in this section there is the constructor of the object *haptic* from the class UBHaptic because it must be used by the `UBhaptic_ros_test` father class. The creation of the object is made by the code $haptic = newUBHaptic(ip, Port, IDs)$. The parameter passage is fundamental in order to let the object know to which motor sending commands and receiving information.

Then the parameters which have to compose the final packet sent from PC to Arduino are declared: *mode*, *cmd*, *Kp*, *Ki*, *Kd* are the necessary ones to drive the system. They are passed to *haptic* object by the function declaration

$haptic- > sendCmd(haptic- >$ `actuator_1` $- > ID, mode, cmd, Kp, Ki, Kd)$.
Detailing more each of them it can be seen that *mode* is an indication for the relative motor of which control algorithm applying: position, force, or current one. Due to the fact that the most important indication for an haptic user is the force feedback it is an obvious consequence that there will be a preference for the force control. The set-point value is sent through the variable *cmd* and indicates the quantity to which the regulator has to bring the motor. Depending on the value of field *mode* it can be the force to be apply or the position to be reached.

For what concerns the last three parameters to be set, they are the three gains which create the resulting control law computed by a specific algorithm inside the controller. They need to be passed to the $UBHaptic$ class in order to have an online setting of the control law by the operator directly from remote and not just in the Arduino firmware itself.

But the most fundamental function for the whole software project is for sure

`boost::thread thread_receive(receivePublishFeedback, rate_receive_thread)`.
This allows to spawn a second thread and this is responsible to receive at a certain rate the feedback information from the robot. As it has been already

said the idea of dividing the two aspects of sending commands and receiving information is very smart and useful because the operator can check what the robot is sensing at a different rate with respect to the rate of sending commands and moreover can check robot's data whenever he wants which is an incredibly advantage for a good haptic teleoperation.

Last important portion of code of this class is the check about the opening and closing of communication between Ros and system. Of consequence there is the presence of if condition made on the $getUdpsender$ and $getUdpreceiver$ methods of UBHaptic class.

The class that has the aim of describing the functionalities of the whole system is $UBhaptic$. What about the code to render the idea of the haptic actions? This is organized in three main parts.

The first section is devoted to the creation of the objects of protocol, actuators and Udp channel used. So the constructors of these objects are present. In particular it has to be underlined that these are all attributes of $UBHaptic$ class but they are all classes themselves and provide their own methods and attributes.

For what concerns the definition of the constructors the parts of code are presented below:

- $this- > \texttt{actuator\_n} = new\texttt{tsa\_Actuator}(actuatorsID[n-1])$ where $n = 1, 2, 3, 4$;
- $udpSender = newUDPNode(this- > IP, this- > Port); udpReceiver = newUDPNode(\texttt{LOCAL\_PORT})$;
- $this- > protocol = new\texttt{tsa\_protocol}$;

Obviously the class presents the two principal methods to send commands and receive feedbacks.

Analysing the part of the code aimed to send commands it is easy to understand that the function definition takes as arguments the ID of the motor to which the set-point is addressed, the mode of robot operation, the command value that represents the set point to be tracked and finally the three gains Kp Ki Kd by which building a PID control:

$sendcmd(id, mode, cmd, Kp, Ki, Kd)$. The passage of parameters is fundamental because it allows to dialogue with the father upper level class `UBHaptic_ros_test` which handles all the interaction with the user. These parameters are then passed to a method $buildTsaPackage$ of the `tsa_protocol` class which is one of the attribute of *UBHaptic*. The goal of that method is to build the real package to be sent to the robot in a manner that will be described later. For this reason one more parameter needs to be passed because it must be returned by the method of the `tsa_protocol` class and this parameter is called *pack* which is an array of dimension 22 bytes. In practise *pack* is built correctly and by a certain protocol rule in the previously cited class and it is returned to the *UBHaptic* class where it is sent via Ethernet to controller.

One more detail must be considered: the real package that is sent by the haptic Ros node is not *pack* but *PackToSend* which is another array of dimension 20 bytes. This differentiation is useful in order to make the communication easier and faster because the information package is lighter and it doesn't give any problem because the correct motor to which sending proper data is addressed not by the ID itself but only by the network IP address. So it is a matter of UDP socket which is opened by the node's code with a certain IP address corresponding to a precise EthernetShield card mounted on each motor.

Talking about the other main method of UBHaptic class, this is called *readFeedback*. Thinking about its name it is easy to understand that its goal is to receive data. It does its job by saving in a proper struct called *fbData* the fields of position, force and current feedback coming from the controlled system. In order to make this service, firstly it needs to call a method *decodeTsaFeedbackPacket*() of the attribute class `tsa-protocol`. In this piece of code the interpretation of the feedback from Arduino is present and returned to the UBHaptic class. This returning packet is saved in the previously cited data struct and so exploiting some small methods of UBHaptic class like *getID*, *getActualPosition*(), *getActualForce*() and *getActualCurrent*() it is possible to read the fields of number of motor, position, force and current feedback and to pass them to the upper class `UBHaptic_ros_test`. Here a video print is present and this is really useful in order obtain a clearer supervision of the situation by the operator.

For a matter of completeness the full description of the `tsa_protocol` is reported. It is one of the three attributes of the general class *UBHaptic* together with the *UDPNode* one and `tsa_actuator` but the description of those ones doesn't improve the comprehension of the whole process of the haptic system.

As a matter of fact in the actuator class there's only the association of the passed field *ID* to the variable *ID*.

For what concerns the class of UDP use, it presents in the first part of code the declarations of objects useful for UDP creation. In the second part methods like `receive(data,data_length)` and `send(data,data_length)` are present and they are those which allows the interaction with $UBHaptic$ class.

Detailing more the first cited attribute so the `tsa_protocol` one, this class is the lowest level one because it is responsible of the real work of data packing for sending thread and data unpacking for feedback reception.

The creation of sending packet is made by the method $buildTsaPackage(mode, cmd, Kp, Ki, Kd, pack)$ which takes the passed parameters from $UBHaptic$ class and saves them in the cells of array *pack*. This is made directly for data like *mode* which are integer type and so they can be easily saved, differently from the case of other data which are all floats and so they need to be saved by many for cycles, one for each data. This aspect is crucial because the array is of char type which has the same dimension of byte. Floats have a four times dimension with respect to char and the used protocol doesn't expect to work with floats but only with byte or char data types because of the communication via Ethernet. So with for cycles the float data are saved inside the cells of *pack* array.

Feedback reception and display to video work conversely. As a matter of fact the video printed information need to be numbers of float type but from Arduino a sequence of byte arrives. For this reason in the method $decodeTsaFeedbackPacket$ the use of *union* struct is necessary. Knowing that the feedback sequence from the each motor is composed by the first byte of ID, then four bytes of position feedback, then four bytes of force feedback and the last four bytes of current feedback, the sequence of passages is the following:

- by a for cycle the passed byte array is saved in three smaller arrays of dimension four bytes, each for one type of feedback
- each array is converted in the corresponding float value by the *union*
- each float value is saved in the corresponding field of a data struct called *data*

This struct *data* is returned to the class $UBHaptic$ and saved in turn to the struct *fbData* with the same fields of *data*. Finally by the previously cited *get*() methods each field of *fbData* is read and video printed in the father class `UBHaptic_ros_test`.

4 Chapter: Control of TSA module

4.1 General description

From the controller point of view, TSA actuators provide two main loops: position and force. As a matter of fact TSA has two main sensors basing on which closing the feedback schemes of control. The first sensor is an optical encoder with 200 frames, positioned at the bottom part of the device and it is useful to give measurements of TSA rotor's angle of rotation of in order to close the position loop.

The second sensor take trace of the traction force that the tendons is transmitting to the load and so its use is dedicated to close the force loop.

The two sensors need to be calibrated really precisely because the sensibility of the system must be very high and a fast control has to be applied due to the delicate tasks which can be performed.

The feedbacks are brought back to the Ardino dedicated controller which has to implement two different PID schemes depending on which type of control the user wants to be applied.

In practise the useful information that Arduino needs to manage in order to control each related motor are the set-point, the three PID gains and last but not least the operational mode. One more parameter is the feedback of position or force depending on the type of quantity which the operator wants to control.

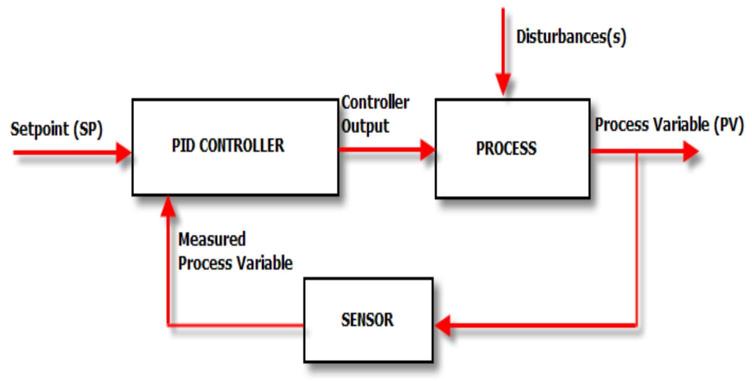


Figure 4.1: PID feedback control

How these parameters are kept together is a matter of the PID class which is present inside the standard libraries of Arduino.

The most important piece of code of this library is for sure the constructor of the objects PID. As a matter of fact the goal of creating a real PID algorithm is made possible by the call of the class PID which takes as arguments:

- Set-point of position or force from Ros
- Gains Kp, Ki, Kd for PID tuning from Ros
- Feedbacks of position or force from sensors mounted of motor block

The PID class comprehends some methods useful to compute correctly the control law. In particular they are able to process the gains data from the operator sending packet and to calculate the sum of the three proportional, integral and derivative terms.

Moreover the PID class allows to set the saturation of the controller in order not to have an overloaded work of the actuator.

Due to the fact that the main idea of the haptic interface is to have two types of controls firstly it is necessary to instantiate two objects:

- `PID_pos` to realize the position control
- `PID_force` to realize the force control

Obviously thanks to the inheritance principle, these objects have the same attributes and methods of the general PID class.

Two of the most used methods which are present in the Setup portion of code of Arduino firmware are $SetOutputLimits(min, max)$ and $SetMode(mode)$. These respectively establish the saturation limits of the output control law computed by the controller and switch on or off the computation algorithm. In particular talking about the second method it is important to underline that if the algorithm is switched on, there's the necessity of initialize internally all the processed variables to the 0 value a part from the gains. This aspect is useful because it allows to delete all the previously memorized values which can damage the right behaviour of the system.

$SetTunings(Kp, Ki, Kd)$ is the method of the PID class responsible to pass to the real control law computing algorithm the gains which are sent to Arduino by the external Ros node. The method is called in the *setup* because there the gains are put to 0 value for the first initialization. Then it is re-called each time in the principal loop due to the presence of online tuning of the PID control. As a matter of fact in the main loop the procedure of reading the incoming packet and its interpretation are present so also the gains must be read correctly and the processed. After that they are saved as new gains for the calculation of control law. In practise the operator is able to tune directly from Ros nodes the Arduino controller in order to have a good tracking.

But the magic happens in the *Compute*() method. In this part of code the final computation of control law is done. This last one is expressed in terms of PWM signal which creates a duty cycle able to drive the H-Bridge which creates the current flowing inside the motor windings. The PWM is one of the main modulations of signals which allows to generate continuously a different output depending on the frequency of the PWM wave so it is a modulation technique used to encode a message into a pulsing signal. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices because the global driver for an motors is always composed by electrical bridges able to take the voltages as inputs and to give some current as outputs. How to modulate the voltage and to change its value? PWM answers to this questions: the average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods (duty cycle), the higher the total power supplied to the load.

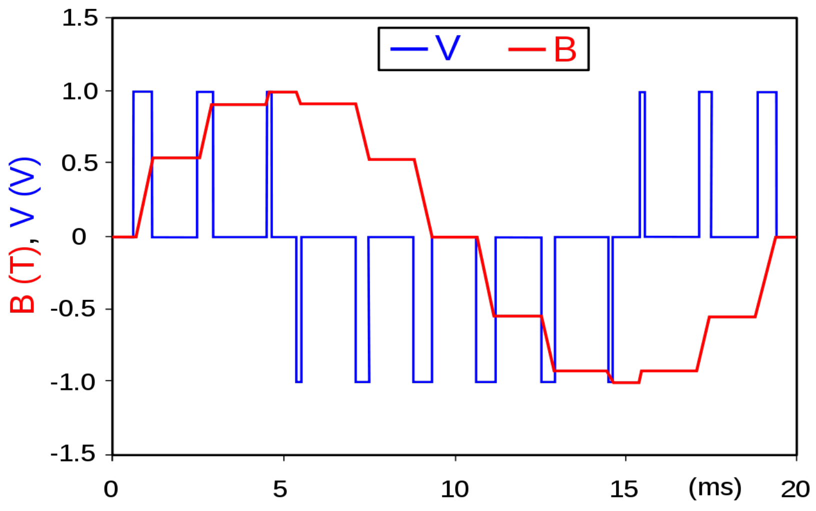


Figure 4.2: PWM example

The resulting output value in terms of PWM is computed inside the previously cited method but the final value written on the analogic output pin 6 by means of the function $analogWrite(PWMvalue, pin)$ needs one more treatment passage.

In fact the computed value can be either positive or negative but always bonded in two thresolds -255 and +255.

Depending on the fact that the result is positive or negative the direction of motor's rotation is set respectively to clockwise or anti-clockwise and this is another information to pass to driver by means of the pin 7 which is a digital output pin where a boolean value true or false is written.

But how to compute effectively the final control law?

Firstly the error expressed as difference between the set point and the feedback is built and then multiplied for Kp gain to create proportional term.

Then it is necessary to compute the integral term. This time the concept is quite more tricky. As a matter of fact in order to express a continuous integration in terms of discrete successive sums, there the necessity to sum a new value to the one computed in the previous cycle. So the code is $Iterm+ = Ki * error$.

For what concerns the derivative term, it needs to be built not in relation to the tracking error but to the difference between the actual feedback and that measured in the previous sample time. This procedure choice is mainly justified by a stability reason.

One last detail about the output law useful to control the motor deals with the function $SetPwmFrequency(pin, divisor)$ present if the Arduino firmware.

This is responsible to change the working frequency of the pin where the PWM law is written by the previously cited function $analogWrite()$.

This aspect is not of secondary importance because it allows to make the driver to work with a correct value of duty cycle and not an higher or lower one.

In general the Arduino Nano standard frequency for pin 6, where usually PWM is written, is 62KHz but in nominal condition of work it must be set to almost 1KHz and this is valid for most Arduino boards (those with the ATmega168 or ATmega328P).

For this reason the divisor is chosen at 64.

4.1.1 Timer

Timers are very useful tools in the programming field because of their property of temporizing certain pieces of code to which they are related and decide when the code must be performed.

They are created principally in two different ways.

The first sees the setting of some registers of the controller and at each clock cycle registers increases until they reach an overflow condition and put a flag to true value.

The second allows the operator to be more involved in some critical decisions like when the flag has to change value. As a matter in this second way of using timers, it is necessary to calculate the number of counts that the registers have to perform before triggering the flag change. This number results to be equal to

$$counts = (\texttt{function\_period/timer\_resolution}) - 1 \tag{6}$$

where timer resolution depends on the nominal clock frequency while the function period is the inverse of the frequency at which the operator wants to change the flag value.

But the most interesting fact in the use of a timer and the reason for which it is really widespread, is that if the flag changes a related ISR [10]is performed. When the flag is triggered, an interrupt is sent to the CPU which stops immediately all the computations that it was doing previously and starts to manage the particular piece of code. So in this way if the operator wants to perform a calculation at a certain frequency, he has to do nothing but not using the second method and setting this frequency for the parameter 1/`function_period`. Doing this the timers is learned to increase the registers of as many counts as said before and when the value is reached the flag changes value and the interrupt is sent to the CPU.

For what concerns the project of haptic Interface timers are used both for force sensor data acquisition and control loop timing. Detailing more about this last one, it has been decided to perform the ISR of control precisely at 1 ms. Obviously the control loop has the possibility of computing the PWM law from one of the two main PID objects which are linked respectively to a position or force control loop.

As a matter of fact the two constructors have the same output variable called *output* and so it takes as consequence that only one of them can be active at a time so only one $PID.Compute$ can work in order to avoid risks of overwriting the same variable.

4.1.2 Sample time

In order to calculate rightly the integral and derivative terms it is necessary to use the time informations. As a matter of fact the conditions to have an integral or derivative contributions is to modulate them by the sample time. This last one is taken into consideration obviously if the magnitudes are discrete in time while, if continuous integral and derivative terms are present, their computation is made considering infinitely small time ranges.

In case of projects where real controllers are involved, the presence of a central CPU which has to perform many computations being temporized by clock cycles, allows to build the integral and derivative terms considering the sample time interval $t - (t + T)$.

For what concerns the integral contribution, it is computed by multiplying the error for the sample time. This happens because theoretically an integral is the sum of the infinitely small rectangles in which the horizontal dimension is the magnitude with respect to which integrating and in this case time. As

[10]Interrupt-Service-Routine

said before in real projects the idea of infinite is not considerable because CPU thinks in discrete ranges.

Dealing with the derivative term, roughly speaking, it represents the opposite concept with respect to integral because it considers the values of variables at a precise instant of time. For this reason the calculation happens taking into account the error and dividing it by the sample time.

But now the question comes up spontaneously: what about the value of the sample time?

This is an important issue because it needs to be high enough to compute rightly the variables but it must be get as low as possible with the aim of making free the CPU fast to perform other calculations. After some tries the value has been fixed to 1 ms.

4.1.3 Anti-wind up technique

The last aspect which has to be treated about the PID class is the presence of the anti-wind up term in the *PID.Compute*() method. This contribution permits to circumscribe the final control law in good range without increasing a lot.

The wind-up phenomenon happens when the value of the control law can't increase due to the saturation of actuator but in the meanwhile the error is not zero. For this reason the integral term continues to increase "charging" itself but without producing an increment of the controller output value. Moreover it makes the controller working badly even the error decreases or becomes negative because the integral term needs to "uncharge" and so the control law can return to the linear part of actuator's characteristic curve.

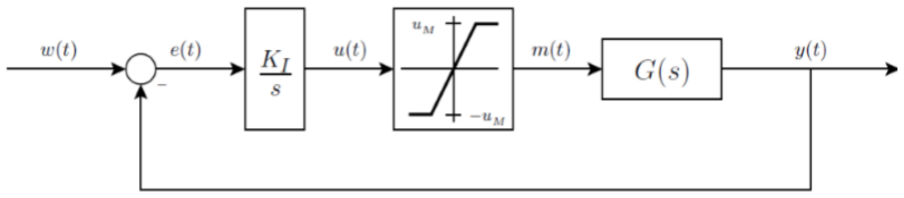


Figure 4.3: Wind-up phenomenon

The anti-wind up techniques could be mainly two: the first is called Internal Model Principle [11] which is based on the hypothesis of process knowledge (which is the actuator in this particular application) and the copy of its dynamic inside the controller.

The second technique consists in the cancellation of the integral term if the PWM law becomes higher than the saturation values admitted by controller hardware. In the context of the project this solution has been chosen for its simplicity of codification and for the simple idea behind.

4.2 Position control of TSA module

4.2.1 Position feedback

The role of feedback in a closed loop control is for sure determinant and of consequence the treatment of this particular signal must be accurately done.

In case of TSA control algorithm one of the two feedbacks is that of position provided by an optical relative rotary encoder.

A general optical transducers like this uses a light shining onto a photodiode through slits in a metal or glass disc. In particular the optical encoder's disc is made of glass or plastic with transparent and opaque areas. A light source and photo detector array reads the optical pattern that results from the disc's position at any one time. This code can be read by a controlling device, such as Arduino micro-controller to determine the angle of the shaft.

The difference and main aspect of disadvantage of this technology is that differently from the absolute one an "incremental" encoder accurately records changes in position, but does not power up with a fixed relation between encoder state and physical position. So devices controlled by incremental encoders may have to "go home" [12] to a fixed reference point to initialize the position measurement or in case of TSA actuators they have to take as 0 position the start one when the firmware begin to run.

For the haptic project there's one main exploited feature of this encoders' type: an incremental encoder works by providing an A and a B pulse outputs that create no usable count information in their own right. Rather, the counting is done in the external electronics and in this case in Arduino firmware.

In particular to sense the motor's shaft motion it is sufficient to set as input the pin in which B pulse is generated and this is done in the initial

[11]commonly called IMP in literature

[12]Homing procedure

setup function of Arduino's firmware.

But the most important setting deals with an "attachment" of the software interrupts signals of Arduino to one special function called *count*() dedicated to perform the interpretation of the previously cited pattern of lights-shadows.

To detail more this aspect it must be said that in practise the function *count*() makes an `enc_count` variable increasing or decreasing at each interrupt signal. This interrupt happens every rising edge of A pulse and the increment or decrement depends respectively on the coincidence of rising edge or descent front of B pulse with respect A.

In reality the calculation's output is given in "encoder tics" unit of measure which is very difficult to be treated for a successive control. The necessity is to pass from this type of variable to a more convenient one expressed in motor's shaft roots.

To reach the result the *resolution* of the encoder is used and this is set at 256. So finally the useful position variable for a precise control is computed as:

$$\texttt{pos\_feedback} = \texttt{enc\_count}/resolution \tag{7}$$

4.2.2 Position controller

Now that also the position feedback has been calculated with the use of interrupts tool, the position control can be performed.

In particular a PID class dedicated to the computation of PWM law is created and this takes as arguments the `pos_feedback` variable, the *output* derived from PID calculation and finally the `setpoint_pos` variables and Kp, Ki, Kd gains sent from Ros central PC:

```
PID_pos(pos_feedback, output, setpoint_pos, Kp, Ki, Kd)
```

The *output* final result is the modulated signal to pass to H-bridge driver which gives power to motor.

4.3 Force control of TSA module

4.3.1 Force sensor

In order to perform a force control the Arduino Nano needs a feedback to close the loop. The measure of force given back to the algorithm is created by an application of a certain load to the tendon and the sensing by a proper force sensor which has the features of an optoelectronic transducer.

This is positioned and fixed in the upper part of the module thanks to a simple cavity housing derived from a processing of ABS box material.

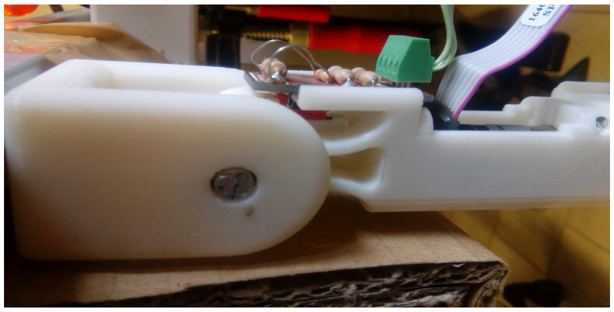


Figure 4.4: Real force sensor housing

As said before, this transducer is an optoelectronic device and so it takes as input the light coming from the external environment and converts the measure in the relative output applied force.

In order to let a different quantity of light hitting the forces sensor and so a variable force measurement, there is the necessity to open a window of various size in which the light can enter. To do this the elastic module have a fundamental role because it is linked to a component which allows to modify the previously cited window.

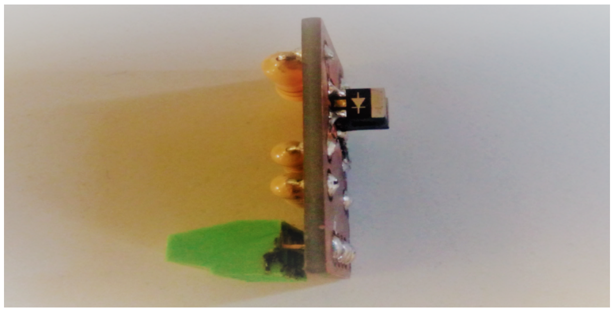


Figure 4.5: Force sensor

But now the question is how the component can move to let the window change its size.

To answer it is sufficient to think about the particular material of which the module is made.

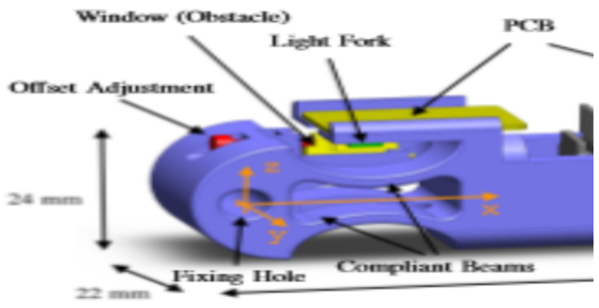


Figure 4.6: Module force sensor detail

As a matter of fact, thanks to the plastic material of which it is made, the module provides some compliant beams in the bottom part and they allow to create the elasticity feature fundamental to move the component.

So when the tendon is put in traction by a load, the overall module deforms itself and let the component moving so that the light can enter more in the window.

4.3.2 Calibration setup

One of the most practical issues that the project has to face up is the calibration of the force sensor. This is used to measure the traction forces applied on the twisted strings in order to close the control loop and to make the motor rotating differently depending on the tension quantity to apply on the each tendon.

Usually a good force sensor calibration is performed by exploiting the combination of a linear motor and a load cell. The motor keeps in traction the load and the load cell returns a certain real value of force. In the meanwhile also the force sensor has returned a force quantity and so the next logical step is to verify which mathematical relationship there's between the two measurements.

In the haptic project a simpler approach has been used but in order to understand the overall procedure it is necessary to describe the calibration setup.

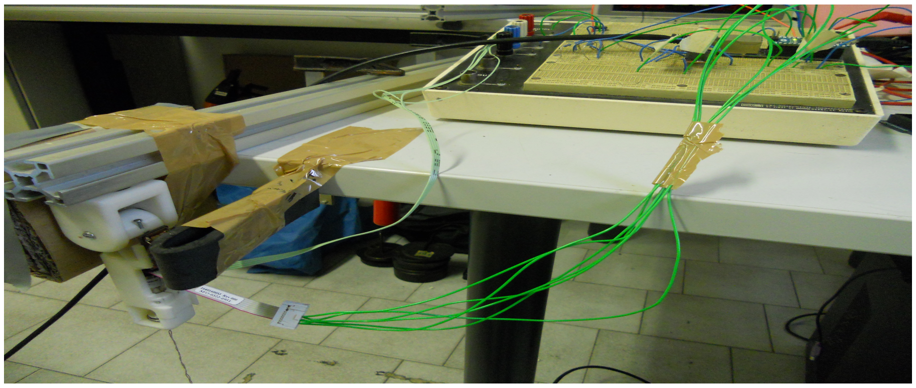


Figure 4.7: Overall Calibration setup

First of all the a TSA module has been fixed to a metallic frame which is robust and nondeformable under the weights that are needed in calibration procedure. If the metallic bar would be flexible it had introduced a noise component in the measurement due to high vibrations.

Than the pins of motor are linked to the breadboard where Arduino controller is positioned in order to realize all the necessary electrical circuits.

As a matter of fact the breadboard is a practical construction base for prototyping of electronics. This makes it easy to use for creating temporary prototypes and experimenting with circuit design which is useful for didactic purposes or research projects.

The breadboard instrument provides a lot of holes which are linked one to each other in two different ways: those one for devices' power supply are linked in horizontal way while the biggest central area where a large number of components can be housed to form a circuit are linked vertically.

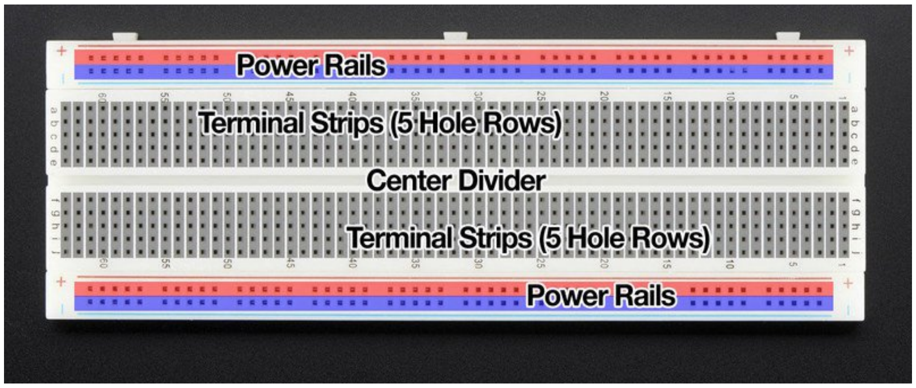


Figure 4.8: Breadboard

Finally the most simple-to-understand but fundamental part of the setup is the motor's module put in vertical with the twisted string flying in the air. This is due to the successive step of attaching to it various weights which represent the load that a linear motor can simulate.

The overall module is surrounded by two masks, as shown in Figure 4.7 to avoid the light changes to affect a lot the measurements.

The already described setup is very useful to implement a good calibration which is precise enough for a successive force control.

In reality the most obvious issues that a setup like this can generate are the presence of noise in the measurement due to two main factors: the first is for sure represented by the flying wires which can create electrical problems. Moreover a little percentage of electromagnetic influence between the wires is present.

The second most important issue is the light hitting the force optoelectronic sensor. As a matter of fact this type of sensor takes as input light and converts it in voltage output signal. Light can be a problem if there are lot of changes in the environment and many shadows are present. Despite the mask created to protect the sensor works good, the problem remains and risks to damage the measurements. The real solution to this problem will be explained in the next section where calibration procedure will be described.

4.3.3 Calibration procedure

The calibration procedure is based mainly on the necessity to create a sort of mapping: as a matter of fact the actual quantities which are returned by the sensor are expressed in Volt while the real forces are evaluated in Newton unity of measurement. Because of the operator works with this last one type, the calibration has the aim to provide to the control algorithm the right feedback.

This mapping is made possible by building the special setup previously described in which there's only the motor plastic box which is put in vertical and let the tendon being suspended in the air with the aim of hanging up a metallic base. This is useful to charge the tendon with a series of half a kilo weights.

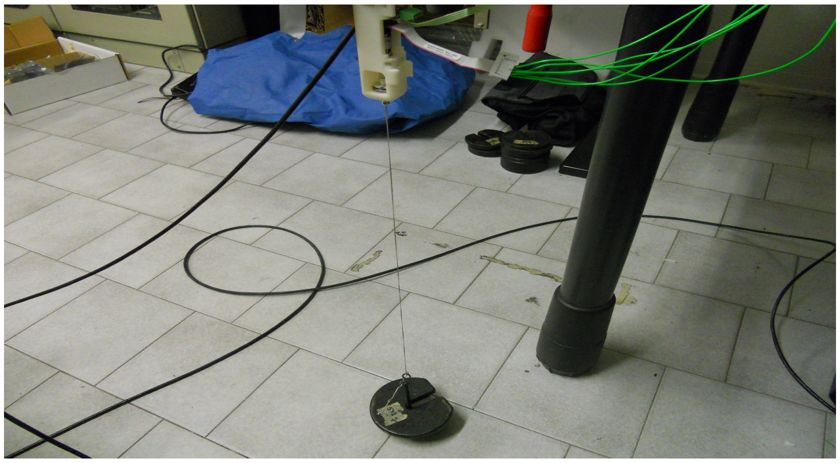


Figure 4.9: Motor's module for calibration

As said before, obviously the final unity of measurement is Newton but the calibration is thought in terms of kilos. The passage between these types of quantities is easy because there's only the gravity acceleration parameter in the middle:

$$Newton = Kilos * \frac{m}{s^2} \tag{8}$$

Exploiting the serial prints tool of the Arduino controller , it can be possible to check the variation of the voltages quantity depending on the increasing and decreasing of the weight charge on the tendon. The obtained result in this project is really interesting because it comes out that there's a constant decreasing of measurement of 0.02 Volts for each half a kilo added.

For a matter of clearness of data measurement, during the experiment it has been used a particular mathematical tool called mobile window filter.

This is a peculiarity of the calibration process because it permits to have more stable measurements thanks to its property of computing the medium value between a series of acquisitions (in this case the window is over 110 values). This fact is important because during the process of adding weights there has been a small quantity of noise but large enough to affect the calibration. The noise is present because of the force sensor is an optoelectronic one so also very little variations of light in the ambient can damage the measurements. In this particular project there are not huge variation of voltages involved so that also a noise of small entity can damage the measure, and a 0.02 Volt variation corresponding to half a kilo added is small enough not to be safe from noise. Of consequence a good filter like the mobile window one can help the calibration.

The concept under the implementation of the filter is very simple and exploits the array tool. As a matter of fact the first thing is to fill each cell of the array and to compute the average between all the terms. After that the acquisition of data measurements continues in a way to subscribe the oldest value in the previous series with the new data obtained by the sensor. Than another average is computed and this procedure of updating and average computation is repeated continuously.

The introduction of the mobile window medium filter creates some problems in terms of timing. In fact depending on the number of cells in the array the availability of a reliable force value changes. This is due to the fact that if the real force value in terms of Kilograms, obtained in a certain acquisition moment, differs from the previous one , it is inserted in the array's series of values but the medium value results to be affected more by the other values. In order to have a significant change in the measurement all the cells need to be filled by values similar to the new one and this happens after a quantity of acquisitions equal to the array cells' number.

Using this kind of filter a good procedure of calibration can be achieved: increasing and decreasing the number of weights hanged on the tendon it can be possible to save in an Excel LUT [13] the comparison between the voltage measurement and the real Kilograms force value. Taking these data the second passage is to exploit the standard function $REGR.LIN()$ in order to find the best linear function that could approximate them. In practise considering as x-axis values the Kilograms real forces and as y-axis values the voltages, it has been found a linear function of slope 0.0490.

[13] Look-Up-Table

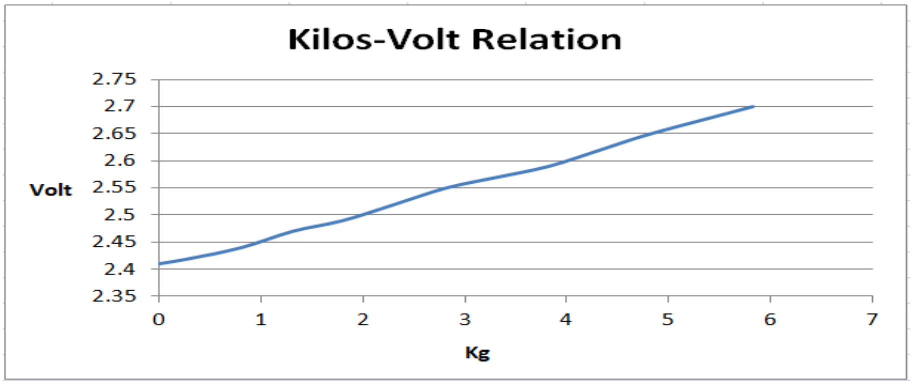


Figure 4.10: Force sensor calibration

This value is the fundamental one in terms of calibration because it permits to give to the force control loop the right mapped feedback to be put in error with the set point of Newton force decided by the operator.

For the sake of completeness it must be said that this just described procedure was not implemented in the principal firmware of Arduino but in a specific program.

4.3.4 Force controller

For what concerns the general firmware to be used in the project, the measurement of feedback from force sensor is coded thanks to the possibility of inserting a second timer which sets different registers with respect to the control loop timer. The force data measurement is made at regular intervals of 1 ms. When time elapses an ISR is triggered in the way described in Section 4.1.1.

The function dedicated to compute the feedback useful for control algorithm shows the same mobile window filter and thanks to the set frequency of the timer handler and the number of cells present in the mobile window filter's array, the value of the delay introduced by the filter can be computed. As a matter of fact it is sufficient to multiply the number of cells for the period of the timer because it is the time necessary to fill all the cells with coherent values and to perform an adequate average to represent the real force change on the tendon.

In this project case the filter's window is 110 cells large so that the delay is about ten microseconds which is an acceptable value which doesn't affect a lot the performances of the final force control loop.

The first acquisitions without the intervention of the filter give these results

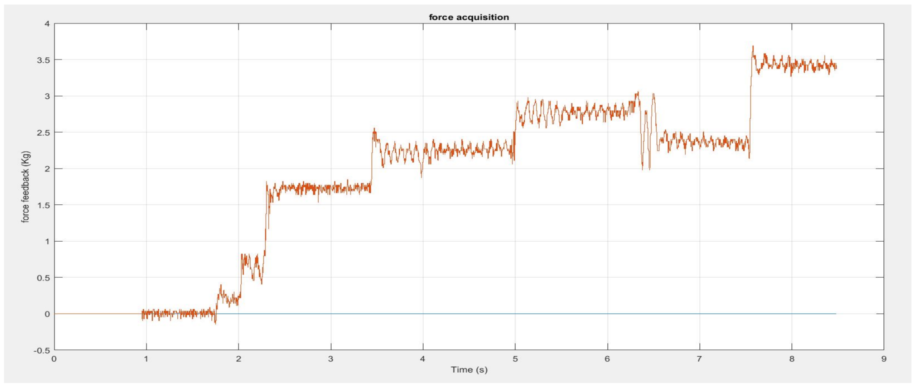


Figure 4.11: Force acquisition

The succession of the increasing weight has been chosen randomly and shows a series of 0 Kg, 0.3 Kg, 0.8 Kg, 1.8 Kg, 2.3 Kg, 2.8 Kg

Going more in details of the graph and zooming it can be seen that there is a ripple of about one hectogram and at a really high frequency which leads to a incorrect and inaccurate control algorithm.

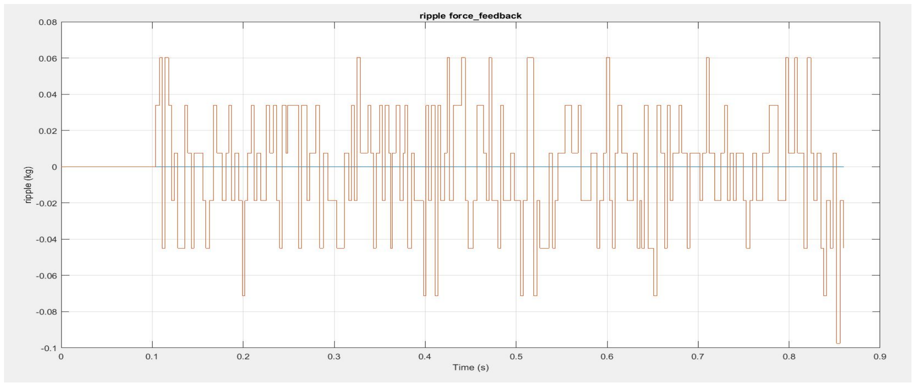


Figure 4.12: Ripple

The solution to filter these data is made by computing the medium value over 110 acquisitions and to save it as the new force feedback.

For a matter of simplicity it is reported a feedback measurement obtained by a reduced series of weights addition.

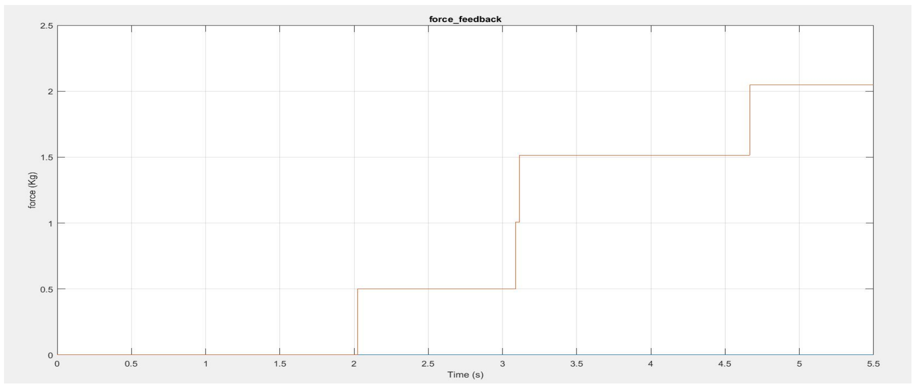


Figure 4.13: Final force feedback

During the force feedback measurements it has been taken into consideration the possibility to insert also a sort of forcing of value to the previous saved one if the new acquired differed less than 0.1 Kilos. This acts as a sort of thresold and could be important if the system is mounted wrongly. As a matter of fact an additive noise with respect to the unique light-related one could be present if the tendon doesn't move only in axial direction but also in an horizontal one.

The introduction also of this technique to improve the measurement results wrong for two main reasons: the first deals exactly with the correct structure building of the haptic system because the motors' disposition permits the traction of the weights only in the axial direction. The second reason deals with the own concept of cutting. In fact the possibility to force the feedback to a value that is decided by a thresold doesn't guarantee to have a continue smooth signal but takes to one with sudden changes so made by a series of steps.

Obviously this fact can't take to a good control law because of the necessity to compute it to react to these peaks and so there's no possibility to have a correct PWM law. This is why it has been decided not to use also this technique to measure force feedbacks.

One last important fact needs to be clarified: the passage between the sensor acquisition in volt and the translation in Kilos.

Exploiting the previously computed value of 0.0490 which is the slope of regression linear function, it can be possible to compute the logical passage but it is not sufficient. In fact if we think about a linear function, its formula is

$$y = m * x + q \tag{9}$$

In this case the y value is the force expressed in volt as the sensor naturally returns, while the x is the force expressed in terms of Kilos. The aim is to compute the x quantity given the related y one. To do this the m slope is already available while the q offset has to be calculated.

The terms q is computed in the *setup*() function of the firmware doing a series of 1000 acquisitions and making the medium value over them. This is also called the *bias* term.

At this point the inverse formula to have the force feedback in Kg is:

$$x = (y - q)/m \tag{10}$$

which in terms of code results to be:

$$\texttt{force\_feedback=(outvolt\_average-intercept)/slope} \tag{11}$$

It is useful to remember that the value `outvolt_average` is the medium value computed by the mobile window filter while the one used by the final control algorithm is called `force_feedback`.

After the tricky work to have an almost clean feedback from the force sensor, this signal is given in entrance to the PID class dedicated to the force control.

As said in previous chapters other parameters that the class needs to have to compute a correct and performing PWM law are the external reference to be tracked and the gains of PID scheme:

5 Chapter: Experimental results

5.1 Matlab-Simulink environment and its connection with Arduino Nano

Matlab-Simulink is a fundamental instrument in the engineering field because it allows the user to perform tasks like system modelling and data analysis. For what concerns the haptic project, it has been used principally for a reason of plotting data which is possible thanks to its Ethernet tools present in Simulink environment. These allows to receive packets of informations from other devices but this is possible exploiting special IPs addresses.

These are the broadcasting IP addresses through which there's the possibility to send contemporarily informations from one device to all the other devices which are communicating on the same LAN network and this is done between Arduino controller, central PC and an other PC in which Matlab runs. Arduino has to send feedback informations to central PC in which Ros has to run while, from its side, Ros has the aim to send packages with mode of operation and set points. All these data can be plotted into a Simulink environment to have a more intuitive and precise evaluation of what is happening inside the system.

Talking more in detail, the principal block which permits the reception of information transmitted by the network is the UDP-Receive block.

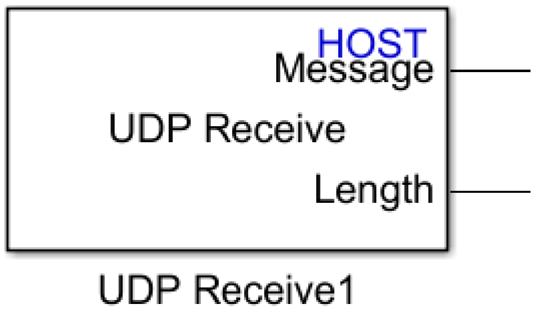
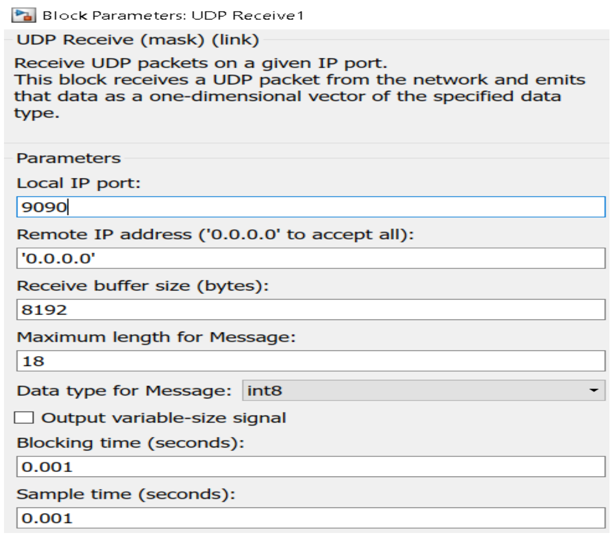


Figure 5.1: Udp Block

Thanks to this block the series of char data transmitted by Ros or by Arduino can be taken and then plotted. It is important to set well the block's parameters because the correct plotting of the data depends on this aspect.


Block Parameters: UDP Receive1

UDP Receive (mask) (link)

Receive UDP packets on a given IP port.
This block receives a UDP packet from the network and emits that data as a one-dimensional vector of the specified data type.

Parameters

Local IP port: 9090

Remote IP address ('0.0.0.0' to accept all): '0.0.0.0'

Receive buffer size (bytes): 8192

Maximum length for Message: 18

Data type for Message: int8

☐ Output variable-size signal

Blocking time (seconds): 0.001

Sample time (seconds): 0.001

Figure 5.2: Udp Block Parameters

The most important parameters to be set are the IP address and port from which the data are sent, the number of data to be received and their types. In this particular project the fundamental information's type transmitted through the LAN network is the char or byte one depending on the presence of sign. As can be seen from the previous figure, 0.0.0.0 is the IP dedicated to the reception of packages from every connected device so that the specification of the port on which listening to is very important to distinguish the devices one from each other.

From the sender side so the Arduino and Ros one, the broadcasting IP is 255.255.255.255.

One last consideration about the use of Matlab-Simulink environment is that in reality the original type of information which are transmitted is $float$ a part from the operational mode that is *byte*. The $float$ type is composed by four chars so in order to plot the correct value it is necessary to put together the four chars in which the original float was divided in, and to re-create the signal.

This is possible by the use of a Matlab function called *typecast*.

The whole Simulink scheme useful to plot all the information is reported below

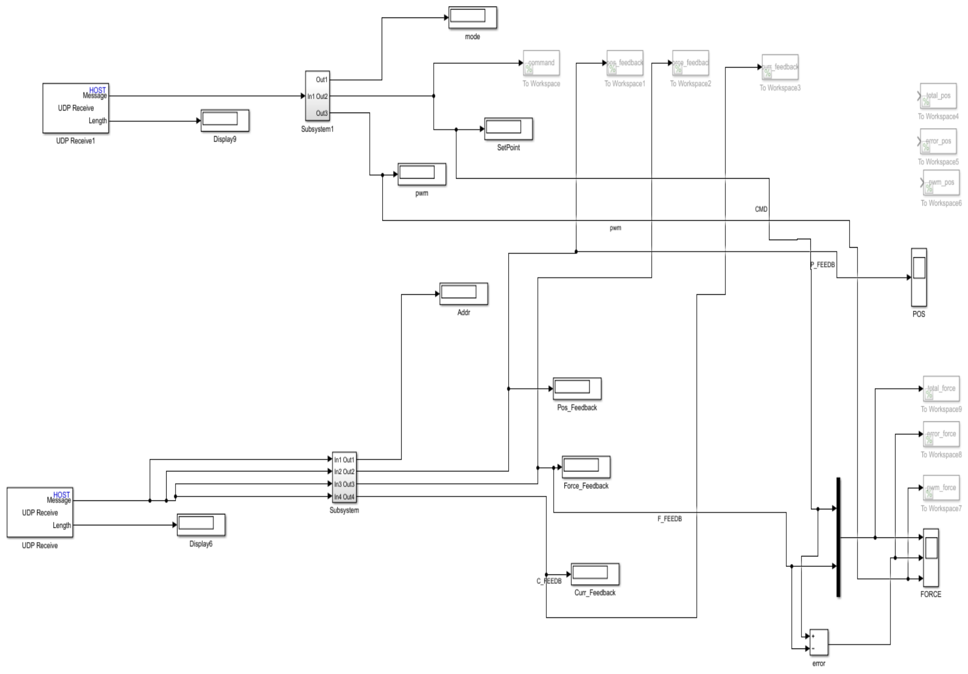


Figure 5.3: Simulink scheme

In which the subsystem where the outputs of UDP-Receiver block goes, can explode in a scheme with the previously cited Matlab functions. There are three Matlab functions and so three *typecast* because feedbacks are of three types and for each of them the reparcelling of bytes into float must be provided.

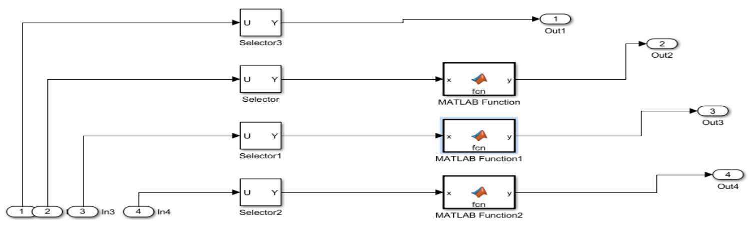


Figure 5.4: Subsystem

The parameters of the simulation are set in order to have an ODE23t variable step solver. The relevance tolerance is imposed to 0.01.

Due to the maximum and minimum step size present in the simulation, this runs with time indication of 1 simulation second equal to 22 real seconds.

5.2 Position control results

Exploiting the acquisition possibilities given by the UDP tools in Simulink environment, it is possible to obtain and plot some important results of the position algorithm implemented inside the Arduino firmware.

The position control loop has the aim of taking the motor's output in terms of rotations of the shaft to be equal to a set point given by the operator. This fact is important to position in a correct manner the central bracelets with its linked gimbals.

One of the main uses of this particular control is for sure the possibility to take the bracelet in an "homing" position in order to initialise the whole system.

After that procedure the nominal working phase can start with the goal of transmitting to the operator the force sensations.

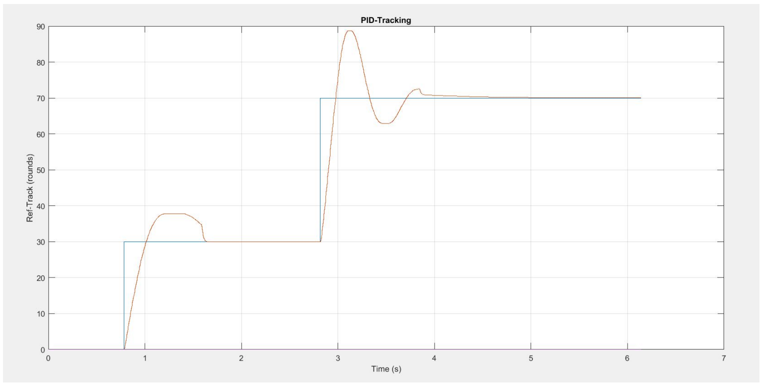


Figure 5.5: Position track with online tuning

As it can be seen by the previous plot the results of reference tracking are really successful because the steady state error is approximately equal to zero but this fact is reached after a long interval of time.

This happens because the tracking performances are obtained by an online tuning of the PID controller. As a matter of fact the first approach is to choose an high proportional gain which take the system to have an oscillating response as it can be seen until the simulation second 1.45 or in the case of set point change until second 3.45.

After those times the tuning of the controller's parameters change because the integral and derivative terms are added. The response changes instantaneously and the curve tends to the set point in a really fast way. Moreover, as said before, the error tends to zero which guarantees an extreme precision.

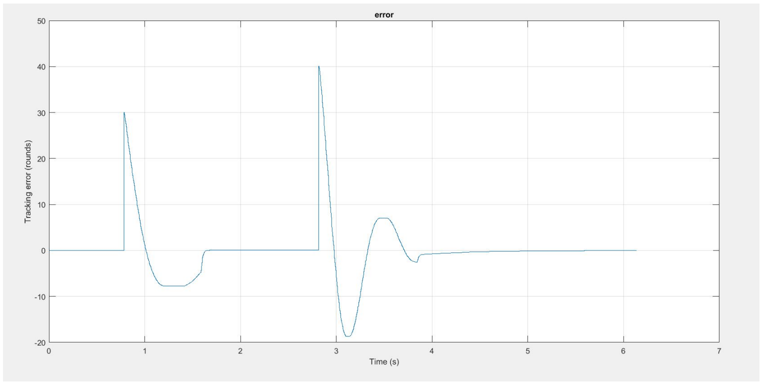


Figure 5.6: Tracking error

The PWM law is reported below

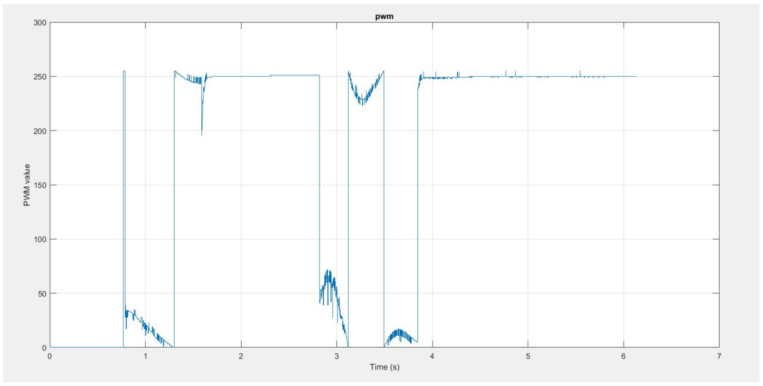


Figure 5.7: PWM law

In order to prove that the chosen parameters permit a successful control both in terms of steady state reaching time and precision of positioning, an "ad hoc" node in Ros has been implemented. In this program a series of set points is generated and published in the topic where the `UBHaptic_Ros_test` node can read them and build the packet to be sent to Arduino. Those references are a succession of steps of three units.

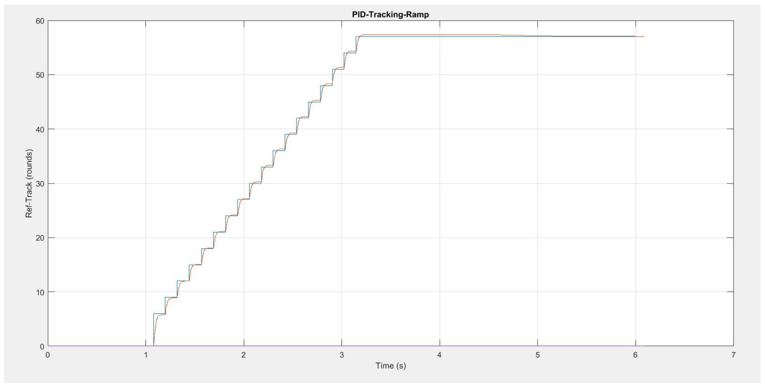


Figure 5.8: Series of references tracking

From the results the validation of the chosen gains is done and the controller performances are really satisfying.

5.3 Force control results

For what concerns the control algorithm dedicated to the force reference tracking, this is a similar problem with respect to the position loop but it hides some "dangerous" issues.

The loop of control is implemented in the same ISR where the position control law is calculated but obviously the force one is triggered only when the mode coincides with the number $MODE2$.

One of the note that has to be immediately done talking about this kind of control is that the references to be tracked must be all positive because in nature there couldn't exist negative forces.

The second point to be touched consists in the possibility to perform the experiment in a approximated setup, built for the sake of force sensor calibration and controller tuning.

As a matter of fact the setup described in the previous chapter of calibration topic really damages the accuracy of measurements and introduces some noise. The continuous oscillations of the feedback signal is not a good thing for both calibration and for sure control.

The noise is caused by the electrical signals inside the breadboard which acts as an antenna and the flying connection between pins of Arduino controller and those of motors. The problem has been solved quite at all by the introduction of an efficient filter during the calibration phase but in the control one some spurious peaks of noise remain and they still has a significant relevance.

The remaining oscillations are present because when the tendons pull the attached load, some vibrations are induced in the setup and so some electrical connections born.

This aspect is evident in the plots taken with the help of Matlab-Simulink tools

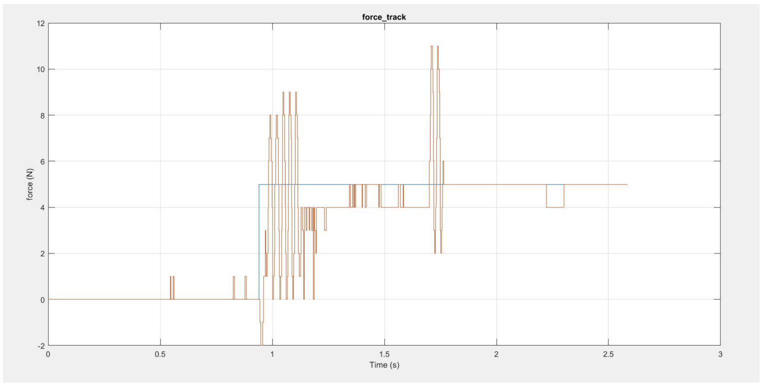


Figure 5.9: P controller performance

In this case the control is realized by choosing only a P controller with an high gain. Obviously this is not a good choice for two main reasons: the first is the absence of the possibility to track reference with zero error. The second is that the control results really nervous because it reacts immediately to error but doing so it introduces the noise components previously cited.

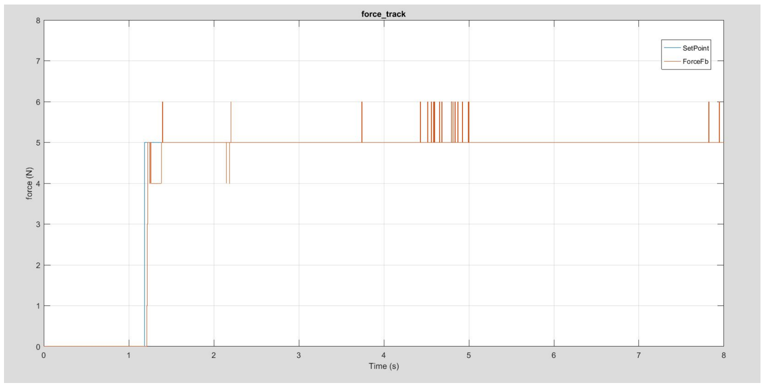


Figure 5.10: PI controller performance

This second plot shows a better controller made by a PI type which results to be less nervous, more precise but it doesn't make zero all the noise. These components of disturbance, even if sporadic, are really bad for a precise control and take the motor to a condition of instability because it has to react to these peaks.

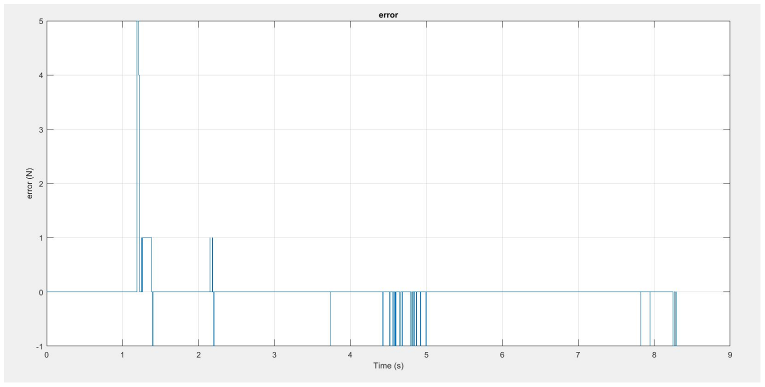


Figure 5.11: Tracking error

The error of tracking is shown in the above figure and basing on this plot it is possible to appreciate more the instability of behaviour.

Obviously also the PWM law becomes wrong and really nervous, thing that can damage the hardware itself in terms of motor's driver. One explicit moment of work in which it could be appreciated that the PWM law becomes "crazy" is between the fourth and fifth second of simulation which corresponds to one minute and half of real work by the motor.

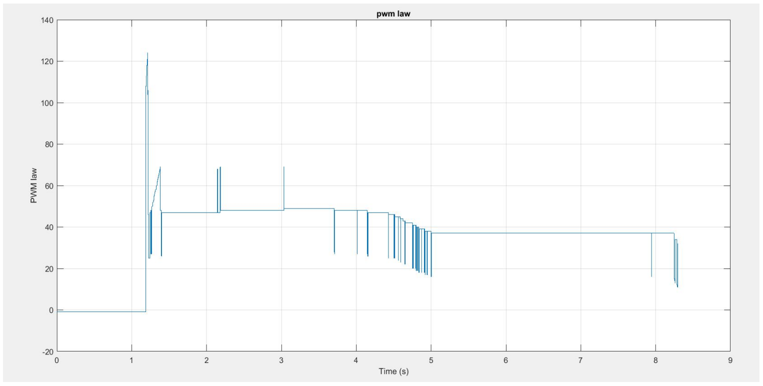


Figure 5.12: PWM law

In practise it could be said that there is a really high improvement in the response of force control when a proportional control passes to a more complex structure like a proportional - integral one. The improvement is obviously evident in terms of precision at steady state and initial oscillations like important overshoot. But the most interesting aspect is related to the minor presence of noise components even if not totally absent.

For sure better results can be obtained with a final setup which encapsulates the motor and the controller embedded on it. This lets to avoid the electrical issue due to the presence of the breadboard, which is only an experimental board, and the flying wires that connect Arduino controller and the relative motor's pins.

6 Chapter: Implementation of kinematic model

6.1 Rviz graphical Ros environment

One last but very important tool which is used in thesis work but in general in many project that requires also graphical part, is Rviz. Its name is pretty explanatory and, in fact, it is the Ros visualization environment.

This is based mainly on the message sharing from generic nodes to the Rviz node through the topic tool. Thanks to this last one, the graphical ambient is able to receive some standard Ros messages in terms of Cartesian coordinates of objects created in user defined nodes and to plot them.

In particular the most used messages of Rviz are the so called TF and Marker.

The first one is a detailed description of the coordinates transformation between two frames in the space. A better explanation of what are this object messages and which are their parameters will be proposed in the Section 6.2.

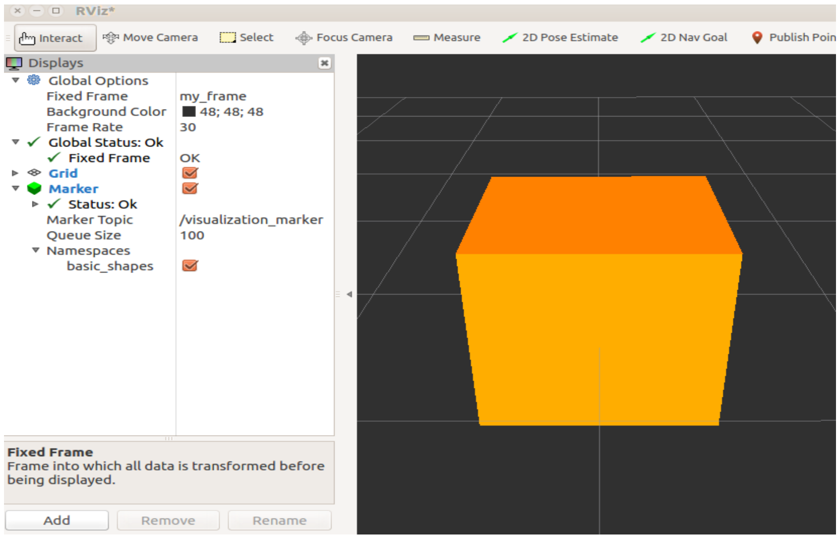
The second message is properly an object with dimensions, colour and geometric scale.

It is a $visualization :: Msg$ type and depending on the number of markers that the user needs to graphic there's the necessity of creating many objects of that type.

If, for example, the user wants to visualize a cube object in Rviz the parameters to be set are:

- The name of parent frame to which referring the coordinates of the object: `header.frame_id`
- *shape* which will be set to $CUBE$
- the Cartesian coordinates $positions.x, position.y, position.z$ where it must be positioned
- orientation in terms of axis-angle $orientation.x$, $orientation.y$, $orientation.z$, $orientation.w$
- the $Scale$ so the dimensions of each side.

Other useful parameters to be set to complete the object definition are for sure its duration of life in the graphical environment and finally the colour in terms of RGB gradation.



Global Options	
Fixed Frame	my_frame
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Ok	
Fixed Frame	OK
Grid	✓
Marker	✓
Status: Ok	
Marker Topic	/visualization_marker
Queue Size	100
Namespaces	
basic_shapes	✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Figure 6.1: Rviz cube

The really interesting thing is that the two objects, TF and Marker can be associated because in the definition of the marker's parameters it is possible to set x,y,z coordinates with respect to the TF's `frame_id` and so, if they have all 0 values, the object coincides with the origin of TF and moves with the same Cartesian coordinates of this last one.

6.2 Vicon technology

Vicon technology is one of the most widespread tools for the aim of knowing the rigid bodies' position and orientation in the space. It tracks them with a precision of tenth of millimeter.

In haptic project it is used to track the Cartesian coordinates of crucial points with respect to a World reference frame.

A substitute method would have been that of computing the spatial coordinates of the haptic's objects by exploiting the relation between the motors' rotations and the shortening of the various tendons. This for sure is a lacking approach in terms of precision.

Obviously this last aspect is fundamental for haptic algorithm and bad measurements happen if encoder and TSA model are used.

For this reason the introduction of a technology like the Vicon tracking is required.

It is substantially a complex set of cameras displaced in various strategic angles of an environment and that can record some data about system that they are framing. In particular there exist three main types of motion capture:

- Optical-Passive: this technique uses retroreflective markers that are tracked by infrared cameras. It is the most flexible and common method used in the industry.
- Optical-Active: this technique uses LED markers connected by wires to the motion capture suit. A battery or charger pack must also be worn by the subject.
- Video/Markerless: this technique does not require markers to be worn and instead relies on software to track the subjects' movement. Varying tracking methods yield different results, but real-time and final data error ranges tend to be larger than marker-based solutions.

For the specific project's purpose the choice has been relapsed on the first method mainly due to the flexibility feature and so there has been the necessity to have a set of markers positioned on the objects to be tracked.

The cameras are infrared ones

Figure 6.2: Vicon camera

and thanks to the particular material of the markers, cameras can send their light beam against them and receive as response beam a reflected light to be taken and transformed in a position information.

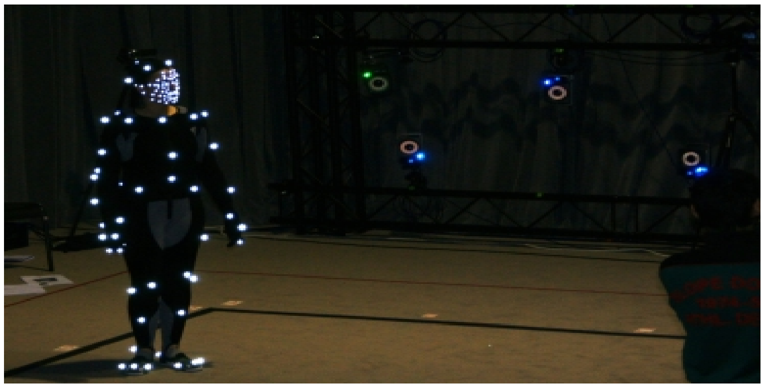


Figure 6.3: Reflection of infrared to markers

Obviously the images that Vicon can process are in 3D stereo mode and so all the spatial informations are taken accurately. One delicate aspect is the resolution of the cameras which ensures a better quality. As a matter of fact all motion capture systems' fundamental accuracy starts with the resolution and quality of the camera sensor. It is what 'describes' the markers to the system and essentially the higher the resolution of the sensor, the more detail you obtain from the marker. This is particularly important in haptic interface application that requires precision tracking of small markers or markers that are close together.

Until now it has been described the hardware part of Vicon technology and what it is necessary to obtain some accurate motion tracking. But there's the necessity to understand how the data of markers reflection are useful.

The strategic position of the cameras in the room or general environment where the object to be tracked is present, allows to create a sort of World Cartesian frame with respect to which acquiring position of the object. On the other hand the markers are positioned on the object itself in a way to describe another reference frame integral to the object itself which can be called Mobile Cartesian frame.

And so the game is done.

Basing on the light' reflection of the markers to the cameras it can be possible to evaluate the position and orientation of the Mobile frame with respect to the World frame and so points originally expressed in the first one are then mapped in the second one. The transformation between the two frames is given by the so called Homogeneous Transformation Matrix.

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 6.4: Homogeneous transformation matrix

in which the first 3x3 square matrix represents the orientation of each axes of Mobile frame with respect to the World one while the last column vector 3x1 represents the displacement of the origin of first frame with respect to the World one.

The motion data are taken by camera, elaborated and then sent to a central PC in order to be published on a specific topic of Ros environment and this communication is make available by an Ethernet transmission.

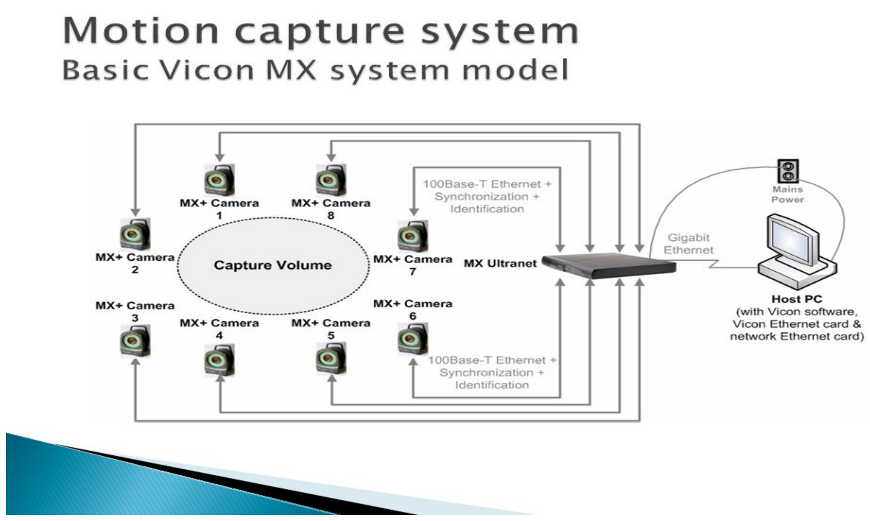
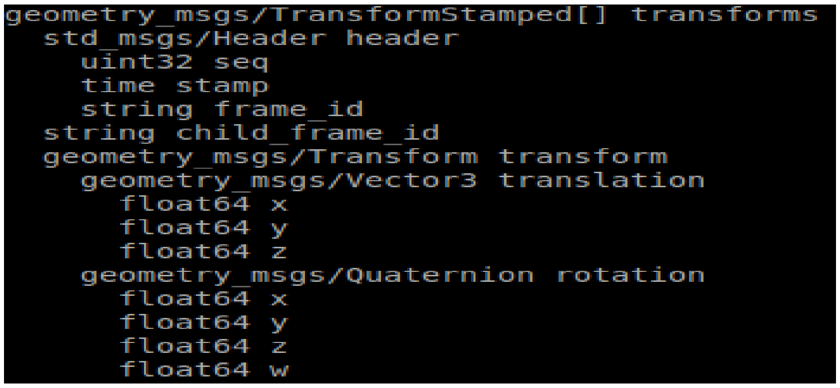

Motion capture system
Basic Vicon MX system model
MX+ Camera 1
MX+ Camera 8
MX+ Camera 2
MX+ Camera 7
Capture Volume
MX Ultranet
MX+ Camera 3
MX+ Camera 6
MX+ Camera 4
MX+ Camera 5
100Base-T Ethernet + Synchronization + Identification
100Base-T Ethernet + Synchronization + Identification
Gigabit Ethernet
Mains Power
Host PC (with Vicon software, Vicon Ethernet card & network Ethernet card)

Figure 6.5: Global Vicon system

When the transformation between the frames is computed and shared with the central computer, it must be encapsulated in a standard message type of Ros in order to be published on $/TF$ topic.

The message is composed by many fields like:

- Parent frame name and child frame name: this reveals to which couple of frames is related the transformation
- Cartesian coordinates transformation in terms of position
- Coordinates transformation in terms of orientation



```
geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  string child_frame_id
  geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion rotation
      float64 x
      float64 y
      float64 z
      float64 w
```

Figure 6.6: TF message fields

For the sake of completeness it must be said that the orientation information is not the same of a 3x3 matrix line in an homogeneous matrix but it is changed in an axis-angle computation.

The TF message is useful both for the computation of the Cartesian position of the various elements which compose haptic interface and the their graphical visualization.

6.2.1 Role of Vicon in haptic interface

All the features provided by Vicon are really useful when haptic interface project is taken into consideration. As a matter of fact the position of the crucial points like the motors' modules `a_i` points and the `c_i` anchor ones need to be known in space to compute the transpose Jacobian.

This last one is an useful mathematical tool to evaluate forces involved in robotic systems and in case of haptic project it is used to understand the relationship between workspace forces and tendons' tensions so the set points for the motors.

One first passage to obtain this result is to fix some markers to the central bracelet. They represent the Mobile frame to be related to a World frame.

In particular this last one is a workspace reference frame Fw defined with the origin placed at the midpoint of the segment connecting the TSA modules `a_0` and `a_1` and its x-y plane including the three TSA modules `a_0`, `a_1` and `a_2` placed in the vertices of the tetrahedron's base (Figure 3.8).

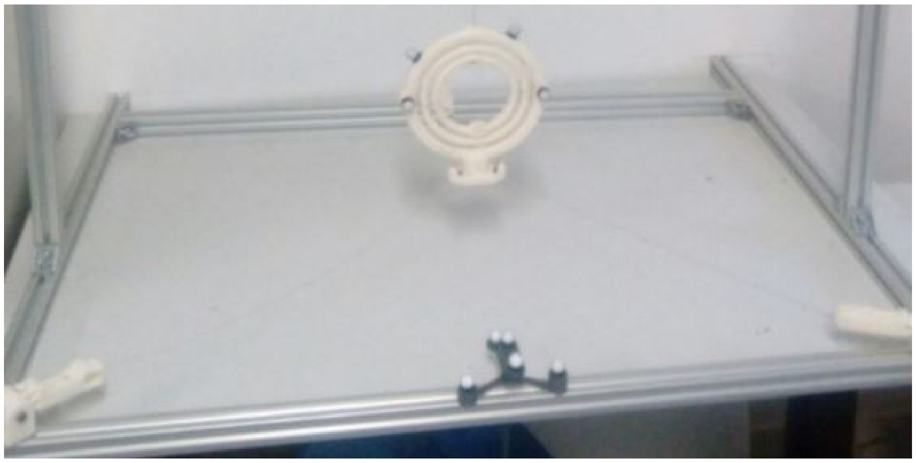


Figure 6.7: Couple of haptic markers in crucial points

For the sake of completeness it must be said the final haptic structure is not already available and so a temporary set of markers representing the World frame is dispose in the previously cited position as it can be seen in the figure above.

The visualization of the two frames in Vicon's environment is reported in the figure below.

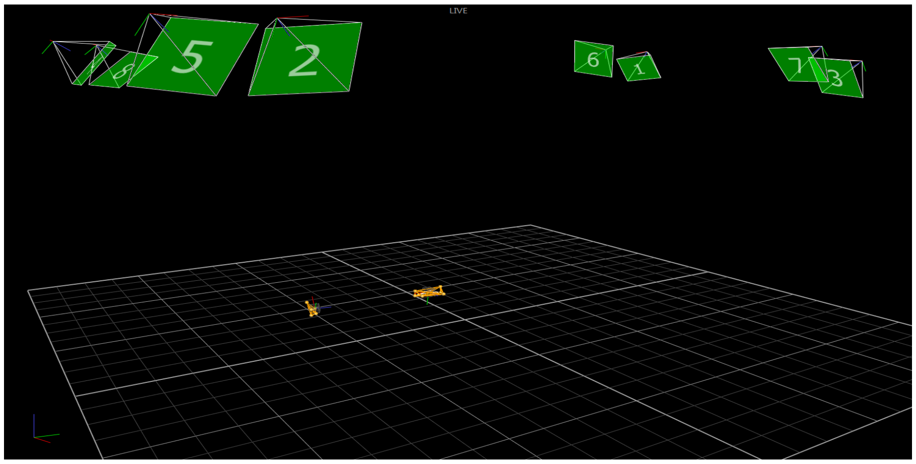


Figure 6.8: Vicon image of tracked object

In order to compute the strings' lengths the informations about the motors' crucial points and those about the anchor points have to be related to the World frame.

In particular the first ones are known directly by structure building while the second ones are known only with respect to the Mobile frame by the bracelet design.

How to refer to the World frame the anchor points' coordinates expressed in the Mobile frame, is a matter of the Vicon transformation as said in the previous section.

The two frames are figure out in the Rviz environment which is the one dedicated to the graphical part of Ros. In particular there will be proposed three couples of images related to different positions of the central bracelet in the system.

The various configurations are reached by means of operator intervention which inserts its wrist inside the bracelet and moves it around.

First position: bracelet at the center of haptic system

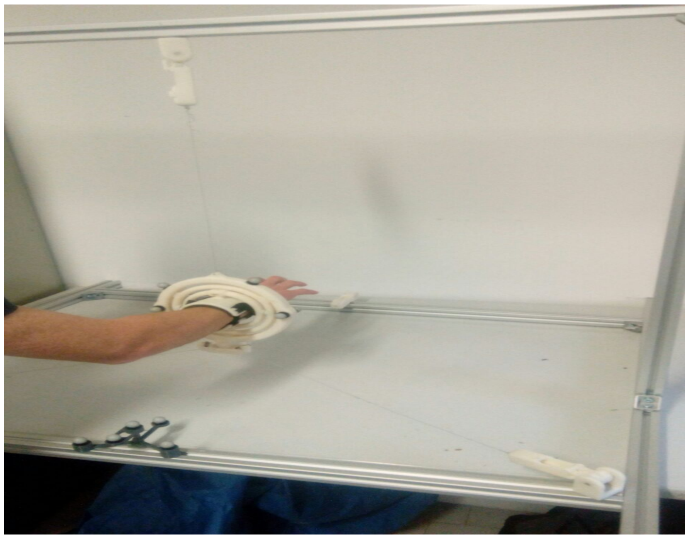


Figure 6.9: Configuration 1

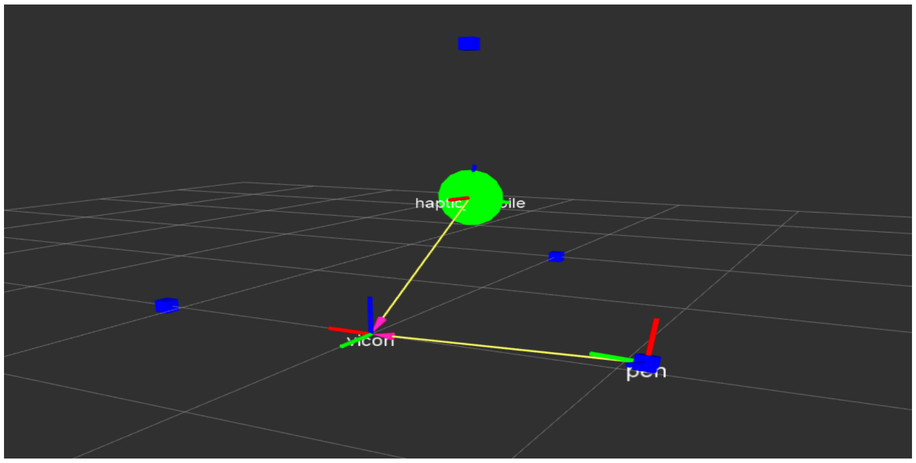


Figure 6.10: Rviz Configuration 1

Second position: bracelet far from motor's module 0

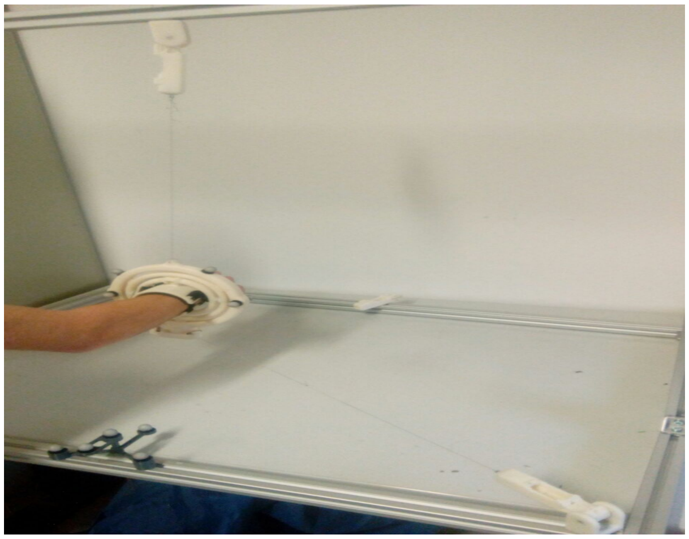


Figure 6.11: Configuration 2

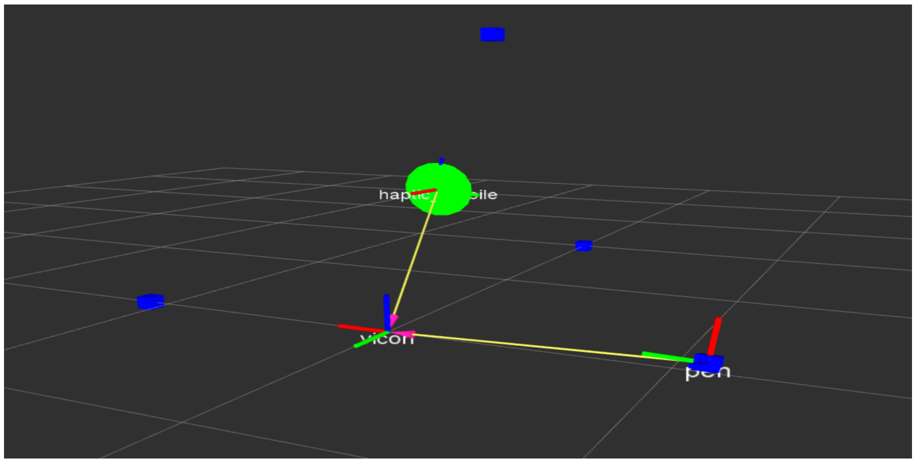


Figure 6.12: Rviz Configuration 2

Third position: bracelet near to the motor's module 0

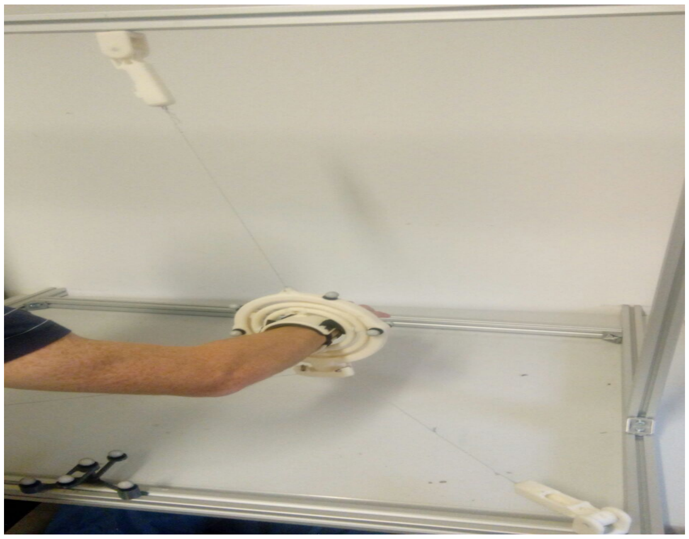


Figure 6.13: Configuration 3

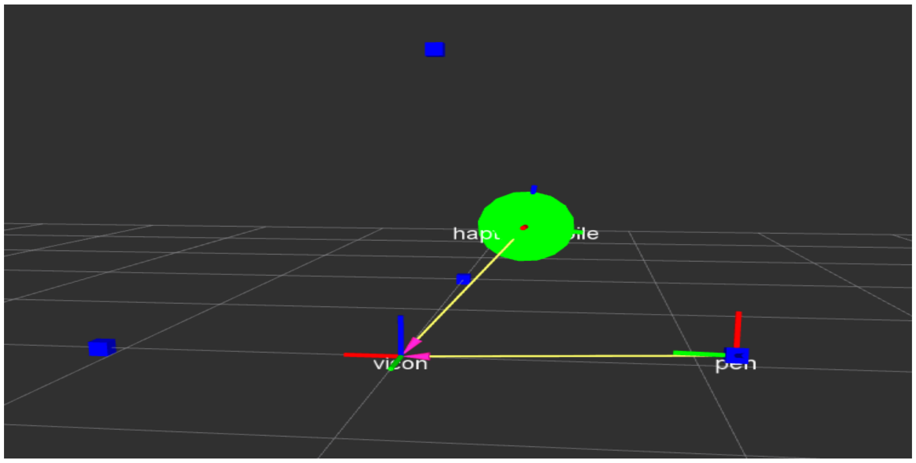


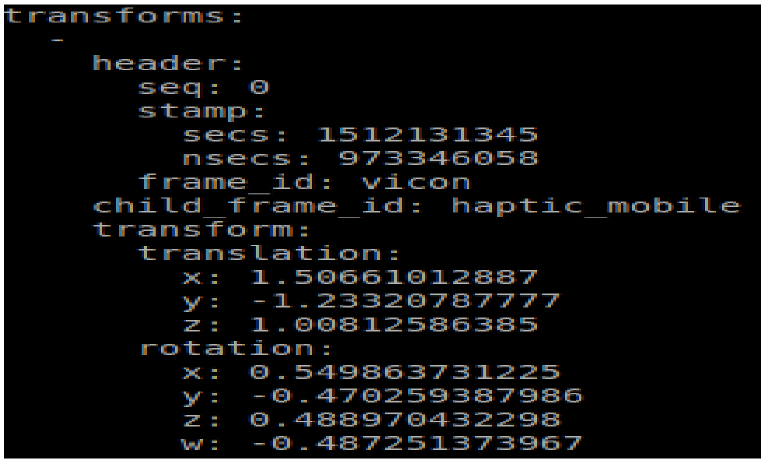
Figure 6.14: Rviz Configuration 3

In all the Rviz images the frames have a green x axis, red y axis and blue z axis for what concerns the Mobile frame while the World reference frame shows a red x axis, green y axis and blue z one.

The Mobile frame is called `haptic_mobile` while the World reference frame positioned at the middle point of the ideal line between motors 0 and

1 is called `vicon`. `pen` is only a visual indication to know where motor 0 is positioned in the structure.

The produced Vicon data in terms of transformation between frames are stored in a TF message composed by the field explained in the previous section. In the figure below it is represented the TF associated to `haptic_mobile` with respect to `vicon` one.



```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 1512131345
        nsecs: 973346058
      frame_id: vicon
    child_frame_id: haptic_mobile
    transform:
      translation:
        x: 1.50661012887
        y: -1.23320787777
        z: 1.00812586385
      rotation:
        x: 0.549863731225
        y: -0.470259387986
        z: 0.488970432298
        w: -0.487251373967
```

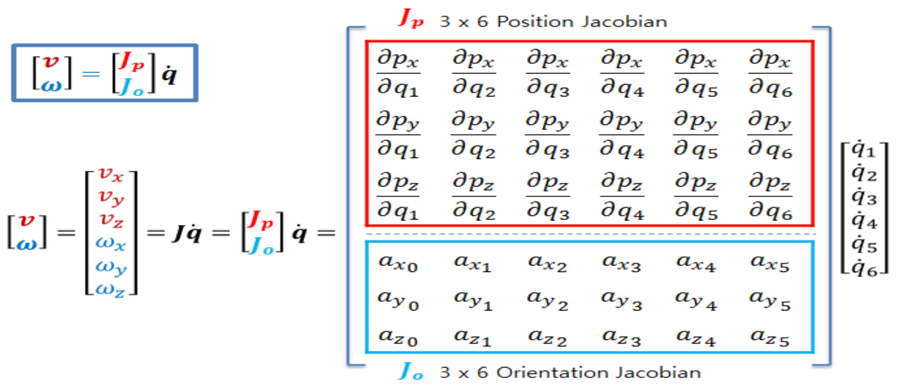
Figure 6.15: TF message example

6.3 Theoretical implementation of transpose jacobian matrix

A Jacobian matrix is a fundamental tool in robotic field. As a matter of fact it permits to have a linear mapping between the workspace and the jointspace's velocities and forces of a general robot and easily pass from one to the others. In case of haptic interface it has a determinant role because it allows to compute the tendons' forces(joint world) given the desired ones(workspace) which the operator wants to have on the central bracelet.

Basing on the kinematic model, the Jacobian matrix can be extract. In fact it is nothing but no a series of partial derivatives of each component of end effector vector with respect to each joint variation. In practise this particular matrix says how the variation of end effector quantities is linked to the jointspace variation.

For this reason the dimension of the Jacobian matrix is n rows equal to the number of workspace coordinates and m columns equal to the number of joints present on the particular robot for which the computation is performed.



$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} J_p \\ J_o \end{bmatrix} \dot{q}$
J_p 3 x 6 Position Jacobian
J_o 3 x 6 Orientation Jacobian

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = J\dot{q} = \begin{bmatrix} J_p \\ J_o \end{bmatrix} \dot{q} = \begin{bmatrix} \frac{\partial p_x}{\partial q_1} & \frac{\partial p_x}{\partial q_2} & \frac{\partial p_x}{\partial q_3} & \frac{\partial p_x}{\partial q_4} & \frac{\partial p_x}{\partial q_5} & \frac{\partial p_x}{\partial q_6} \\ \frac{\partial p_y}{\partial q_1} & \frac{\partial p_y}{\partial q_2} & \frac{\partial p_y}{\partial q_3} & \frac{\partial p_y}{\partial q_4} & \frac{\partial p_y}{\partial q_5} & \frac{\partial p_y}{\partial q_6} \\ \frac{\partial p_z}{\partial q_1} & \frac{\partial p_z}{\partial q_2} & \frac{\partial p_z}{\partial q_3} & \frac{\partial p_z}{\partial q_4} & \frac{\partial p_z}{\partial q_5} & \frac{\partial p_z}{\partial q_6} \\ a_{x0} & a_{x1} & a_{x2} & a_{x3} & a_{x4} & a_{x5} \\ a_{y0} & a_{y1} & a_{y2} & a_{y3} & a_{y4} & a_{y5} \\ a_{z0} & a_{z1} & a_{z2} & a_{z3} & a_{z4} & a_{z5} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix}$$

Figure 6.16: Jacobian structure for a generic 6 Dof robot

As said before when the ambient is the velocity one, the created matrix helps to pass from jointspace to workspace in direct analysis while the inverse analysis is performed by computing the inverse of the Jacobian matrix.

But the problem is dual when forces are concerned. As a matter of fact certain wrenches can be applied by the end effector due to some joint space forces, so the torques applied to joints by the motorizations present on them, and this mapping is obtained using not the Jacobian itself but only its inverse. Otherwise, considering the inverse problem, which consists in finding those joints' forces to reach some user defined wrenches, the transpose matrix is used.

Dealing the actual problem of haptic interface, this system can be seen as a real robot because it shows all the fundamental components of a normal robot. In particular the end effector is the bracelet inserted in gimbals while the various joints are the TSA motors. Starting from their motions the central end effector can be positioned where ever the operator wants.

This means that also for a complex system like it, a Jacobian matrix can be computed but starting from the kinematic model already derived in Section 3.3. The matrix can be computed relating to the lengths of the tendons which are responsible to move the load.

The computation of the transpose Jacobian matrix passes through that of transpose pseudoinverse Jacobian. This is straightforward if the unit vectors associated with each of the four string are considered:

$$\widehat{v_i} = \frac{c_i - a_i}{l_i} \tag{12}$$

where i=0..3 because associated to the considered motor's number.

The pseudoinverse transpose Jacobian matrix which can be obtained by putting as column vectors those previously found unit vectors is the one needed for the direct approach so that of finding the workspace wrenches given the joints' forces.

But the real aim of this project part is to obtain a tool useful to compute tendons' forces given a desired set of Cartesian forces to apply to the bracelet. In order to have it, there's the necessity to invert the previously found matrix and this is not a simple task because in this particular case the matrix results to be non-square.

In fact the n rows are three due to the fact that the vectors composing the matrix are all position column vectors and so they have three Cartesian components. On the other hand the m columns are four because there are four motors(joints) in the system and so it is straightforward to know the number of crucial points a and c.

The approach to invert a rectangular matrix consists in computing its Moore-Penrose pseudoinverse form but it is necessary to choose between a right or a left form and this depends on the structure of the matrix.

In this case the matrix to be inverted results to be an NxM with $N \leq M$ and $rank(J) = N$ so a right inverse form is chosen

$$J+ = J^T * \frac{1}{J * J^T} \tag{13}$$

The pseudoinverse of the originally found pseudoinverse transpose Jacobian is proper the Jacobian transpose which is useful to do the inverse force approach.

In conclusion, when this computation is performed, it is all ready to compute the tendon's forces knowing which wrenches to apply to bracelet.

$$t = J^T * f \tag{14}$$

where t is the vector of tendons' forces and f the vector of workspace wrenches.

An ultimate adding is taken into consideration to have all mapping positive solutions because basing on physics it is a non-sense to obtain negative forces. So a quantity depending on the base of null space of the already computed matrix is added to the previous solution. This gives a proper thresold to the solutions and the quantity is set to 1N.

6.4 Practical implementation of transpose jacobian matrix

All these theoretical passages have been implemented in the `UBHaptic_ros_bridge` packet by coding various functions in its nodes. An important note that has to be done is the choice of Client-Server approach in order to compute the tendons' forces only when there is the necessity.

As a matter of fact the Service is a practical and very widespread tool in informatics because of its flexibility. The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. In practise a server acts as a resources tank for the client which have to create a link with the server and to request for a certain application.

Client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. It only has to understand the response based on the well-known application protocol.

They talk one to the other through messages and both client and server are processes running on the same system but not necessarily on the same machine. In fact an interprocess-communication like this is usually exploited to obtain results in different hardware machines.

So the client can run on a machine while the server which provides the required functionalities may run in another machine. This is the case of the well known World Wide Web which is an example of a communication through a computer network. If a user wants to have access to a certain web page defined over a predefined web address, the web browser(client) queries a web server which returns the cited address and so the link to this is made possible.

The possibility of having a Service over two or more different machines is not exploited in this project because both Client and Service are present on the same hardware where the haptic Bridge runs.

To understand what is the goal of the Service it must be remembered that the results of the Jacobian use is that of computing the tendons' forces given the vector of Cartesian wrenches that the bracelet has to feel in the workspace. So the Client will send to Server the Cartesian forces and receives the TSA set points as response.

The Client is an apart node present in the `UBHaptic_ros_bridge` packet and it is structured in two main parts.

The first is the proper requesting part in which the Client links to the service(srv) `j_forces` of *Jacobian* type defined in the packet. To do this a portion of Client initialization and definition of the various arguments to pass to Server is implemented:

$ros :: ServiceClientclient = n.serviceClient < \texttt{UBHaptic\_ros\_bridge} :: Jacobian > ("\texttt{j\_forces}")$

The passed arguments are:

- $srv.request.Fx = atoll(argv[1])$
- $srv.request.Fy = atoll(argv[2])$
- $srv.request.Fz = atoll(argv[3])$

To conclude the first code portion the real initialization of Service communication is made by the call $if(client.call(srv))$

The second part of the Client node code consists in saving the Server response in terms of the four tendons' forces and in publishing them on a certain topic called $JointsReferences$ in the form of a message containing also operational mode information and gains to be passed to Arduino firmware by the help of `UBHaptic_ros_test` node.

Now the Server has to be analysed. It is implemented in the global project coordinator node `UBHaptic_ros_test` and its call is provided in the main section:

$ros :: ServiceServerservice = n.advertiseService("\texttt{j\_forces}", \texttt{JacobianT\_compute})$.

It is important to notice that the name of the Service object `j_forces` is the same of Client's one.

The function `JacobianT_compute()` is the one which creates the magic. As a matter of fact inside this there's the initialization of two requests and responses vectors. Obviously in the first vector the Cartesian forces sent by the Client node are stored for a successive manipulation while in the second vector will store the tendons' forces obtained after a series of computation. The two vectors are called respectively `f_work` and `f_joints`.

They are passed to a function `jointsforces_compute` implemented in the UBHaptic class which is the real responsible of the calculation to arrive to final tendons' forces. In practise the call present in the `JacobianT_compute` is $haptic- > $ `jointforces_compute(f_work,f_joints)`.

Of consequence now there's the necessity to talk about the methods for Jacobian computation and final jointspace force vector present in the UB-Haptic class.

In this class there are four main methods of which the previously cited one is the more external one while the others are recalled inside it. To have a better comprehension of what is implemented inside those functions, the various theoretical passages in order to obtain the transpose Jacobian and so the tendons' forces must be reminded.

The first method to be recalled is `acquire_positions()` in which the storage of the various crucial points' position coordinates in the haptic structure is present. This is made possible by exploiting a special instrument like Vicon motion tracking technology in a manner that will be explained in the next chapter.

The second method is called `tendonslength_update()` and its goal is to compute the lengths of the various tendons by means of the previously defined positions and exploiting the Equation (1).

The third method called `jacobianT_compute` (not to be confused with the Service call present in the upper node) works a lot because it has to take the previously computed lengths and fill the various cells of the pseudoinverse transpose Jacobian which solves the direct force approach. But, as it has been explained in the previous sections, the haptic project needs the inverse approach to pass from workspace wrenches to joint space forces. For this reason in the same method the pseudoinverse of the already found matrix is performed.

Finally, using the Eigen library for Ros which provides to the user all the functions for matrix operations, it is possible to arrive to

$$jacobianT = jacobian.transpose()*(jacobian*jacobian.transpose()).inverse() \tag{15}$$

where jacobian is the original pseuodoinverse transpose jacobian.

Looking better to this form it results the same thing of Equation (13).

It is important to precise better that all the variables that the various methods can work on, are attributes of the UBHaptic class and so all the functions can see every change.

For this reason the last but most important method `jointsforces_compute` can perform the final calculation computing

```
f_joints=jacobianT*f_work+coefficient          (16)
```

where the term coefficient is required to have all the positive tendons to fulfill the coherence with physics laws.

The so computed `f_joints` are passed back to the `JacobianT_compute` Service call in the `UBHaptic_ros_test` node and from this the Server gives back to Client the four tendons as response.

6.4.1 Code generation for pseudoinverse of jacobian

One final note has to be made relatively to the Jacobian issue. In the practical implementation it has been said that the method `jacobianT_compute` exploits the Eigen library functions in order to compute the pseudoinverse of the matrix. This is for sure the best way to approach to the problem because of the simplicity and clearness of code are reached easily. Calling functions like *.transpose*() or *.inverse*() is a really efficient code implementation of calculations.

But this result is a final one and it has been obtained after a first try with the Code Generation approach.

What is Code Generation?

This is a practical tool provided by Matlab-Simulink environment which allows to create a Simulink model and then to code it in terms of C or C++ language in order to be run on the most favourite platform.

The transformation between a Simulink model and a sequence of code lines is possible after having set some parameters. One of them is for sure the solver of the simulation which can think in terms of continuous time or of discrete steps. One standard choice that was taken into consideration also for haptic project is ode3, fixed-step.

Other parameters for a correct code generation are the machine's operating system which will run the code and the maximum dimension available for the new code. In case of the project those parameters was set in Linux(x64) and 2e20 bytes.

The model used to implement the pseudoinverse of a matrix is

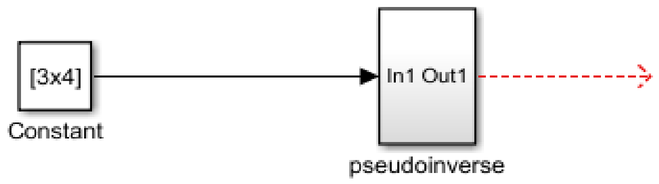


Figure 6.17: Overall pseudoinverse Simulink scheme

At first sight it seems to be a really simple model but when exploded it shows a lot of matrix computations.

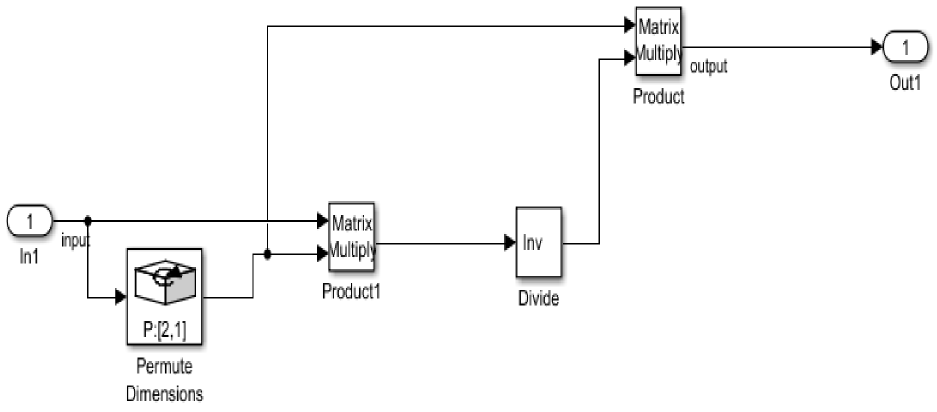


Figure 6.18: Detail of pseudoinverse Simulink scheme

The blocks are the standard ones for the inverse, transpose and multiplication matrix operations and realize the right Moore-Penrose pseudoinverse as required by the dimensions of the Jacobian matrix to be inverted.

One more detail about the previous scheme is that in order to create some input and output global variables to be used in the code, the in and out pins of the overall Simulink pseudoinverse block have been renamed.

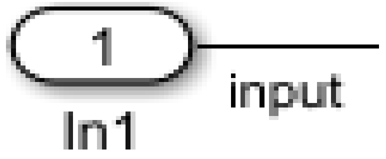


Figure 6.19: Naming of the "in" pin

When the programmer presses the button "*BuildModel*" a set of files is generated. The most important one is for sure the file .cpp in which the various functions are reported in code implementation but there are other files .h where the definitions of these functions and the variables to work with are present.

To have a working code the header files have been included in the `UBHaptic_ros_bridge` packet and the functions in the file .cpp exported in the UBHaptic class.

For the sake of simplicity here it is not reported the code to perform the pseudoinverse because it is a only a series of operations between the various cells of the input matrix.

The unique important thing to remember is that the global variables input and output generated by the code generation are not matrices but column vectors and so when the original pseudoinverse transpose jacobian for the direct force problem has to be inverted it must be saved in a proper array due to the passage from a 2D matrix to a 1D vector. This consideration is valid also for output inverted matrix.

As said at the beginning part of code generation section, this approach allows to arrive at the same result of the Eigen library exploiting method and it is quite practical. On the contrary, it is for sure not very efficient and reusable and this last one has been the major factor that has acted as weighing needle for the decision of the approach to be followed.

Having a reusable code like a pattern to be inserted in various project is determinant for a good project because if something has to be changed inside the code, usually the global architecture doesn't need to be revised and this may save lot of precious time.

6.5 Tendons' forces results

The overall procedure of taking position data from the Vicon system and the successive computation of the transpose Jacobian drives to the computation of the tendons' lengths and set point forces.

For the sake of clearness the presented results will be those related to the three configurations shown in the previous chapter: bracelet in central position, bracelet far from the motor's module 0, bracelet near the motor's module 0.

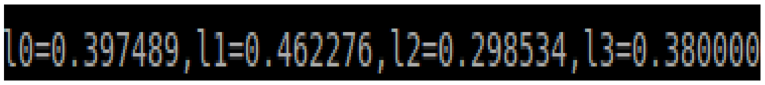
The results are obtained for all the couples `c_i-a_i`. This is possible by initializing the coordinates of these fixed points directly in the World frame while the anchor points are referred to Mobile frame and mapped into World one by exploiting Vicon.

In order to perform the computations there's the necessity to run both the Client node and the Server one which is implemented in the `UBHaptic_ros_bridge` itself.

It must be underlined that for the thesis project's purpose both the results of tendons' length and joint forces are plotted a video but, in reality, during a normal working procedure only the computed tendons' forces are important to be known. This is due to the fact that the operator needs to know what are the motors' set points and verify their coherence.

One last thing to remember is the numeration of the motors and of consequence of related tendons: 0 is associated to the motor at right of World frame, 1 to the motor at its left side, 2 to the motor in the back part of haptic structure and 3 to the upper one.

Talking about the first position of the end effector it allows to have this series of tendons' lengths



```
l0=0.397489,l1=0.462276,l2=0.298534,l3=0.380000
```

Figure 6.20: First configuration lengths

In order to prove that the simulation results in terms of lengths are the real one, a real measurement has been necessary even if of the unique tendon 0.

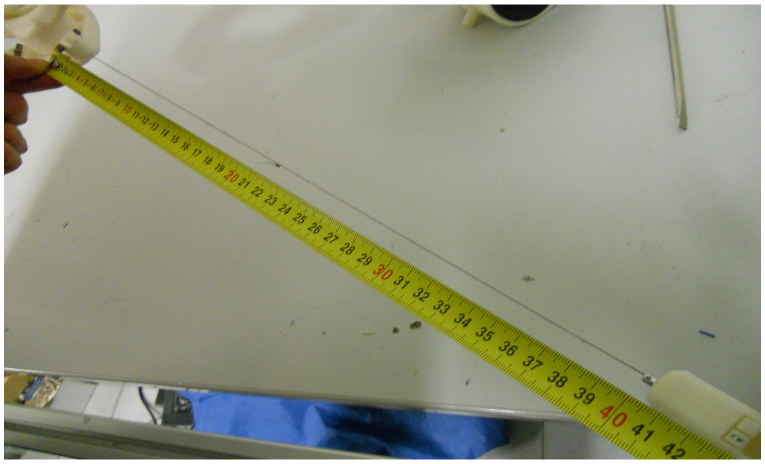
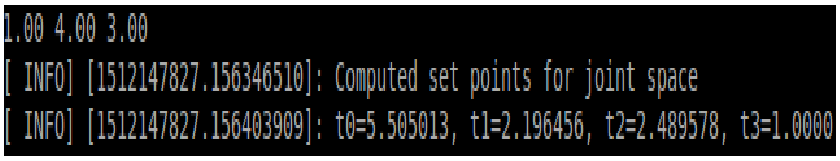


Figure 6.21: Validation by a real tendon's length measurement

By these numbers the transpose Jacobian can be created and of consequence be used to to pass from Cartesian workspace user defined forces to those of joint space.

The next result will be obtained for a choice of workspace forces of 1 Newton along x axis, 4 Newton along y axis and 3 Newton along z axis.

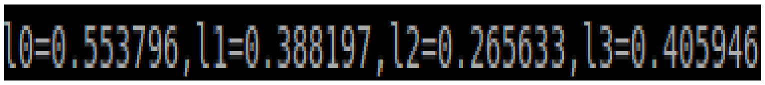


```
1.00 4.00 3.00
[ INFO] [1512147827.156346510]: Computed set points for joint space
[ INFO] [1512147827.156403909]: t0=5.505013, t1=2.196456, t2=2.489578, t3=1.0000
```

Figure 6.22: First configuration tendons' forces

The results in other configurations are reported below

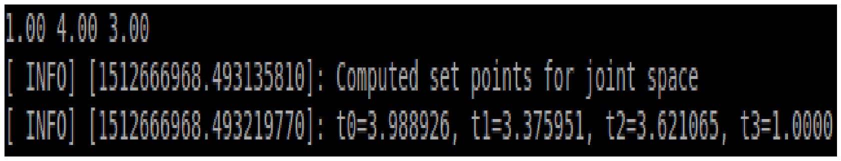
In case of bracelet far from the motor 0 it is obtained



```
l0=0.553796,l1=0.388197,l2=0.265633,l3=0.405946
```

Figure 6.23: Second configuration lengths

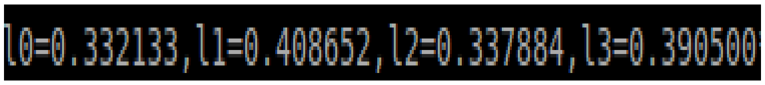
with the subsequent forces



```
1.00 4.00 3.00
[ INFO] [1512666968.493135810]: Computed set points for joint space
[ INFO] [1512666968.493219770]: t0=3.988926, t1=3.375951, t2=3.621065, t3=1.0000
```

Figure 6.24: Second configuration tendons' forces

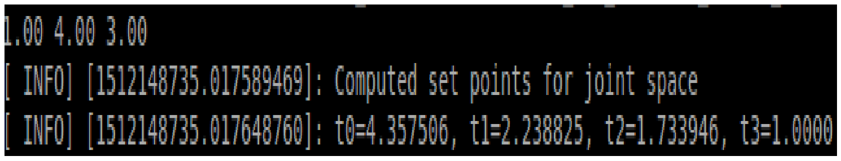
In case of bracelet near from the motor 0 it is obtained



```
l0=0.332133,l1=0.408652,l2=0.337884,l3=0.390500
```

Figure 6.25: Third configuration lengths

with the subsequent forces



```
1.00 4.00 3.00
[ INFO] [1512148735.017589469]: Computed set points for joint space
[ INFO] [1512148735.017648760]: t0=4.357506, t1=2.238825, t2=1.733946, t3=1.0000
```

Figure 6.26: Third configuration lengths

6.6 Singular configuration in haptic interface

Generally, in robotic field, is common to take into consideration some special configuration of robot which can create some problems to its motion. They are the singularity configurations.

They happen in certain determinate conditions when joints are in "bad" positions.

For example in a manipulator with 2 revolute joints a singular configuration is created when the links are aligned and so in terms of angular displacements the second joint has 0 or π degrees

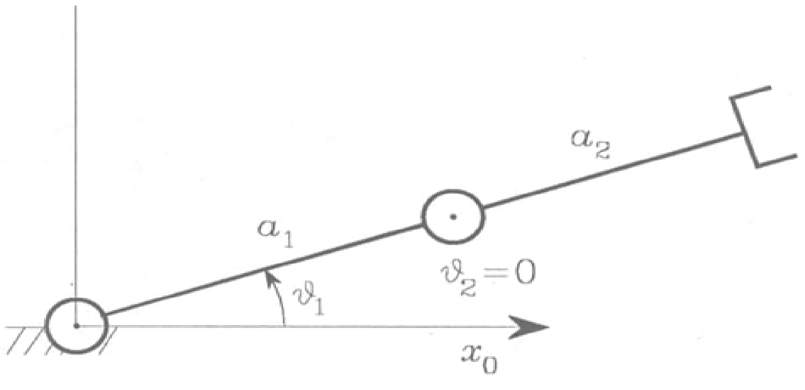


Figure 6.27: Two joints manipulator

One more common example is the proper human body. As a matter of fact it can be seen as a series of links (the bust and the legs) which are connected by special joints(the knees).

The normal condition of standing up is a singular configuration for humans!

In practise there is a simple classification of the singularity configurations:

- at the borders of the reachable workspace: they are created when the manipulator is all extended or all turned over on itself. In general they don't represent a serious problem because it is possible from the programming phase to avoid those configurations

- inside the reachable workspace: they are typically generated by a two or more motion axes alignment or in correspondence of particular spatial postures. These represent an objective issue because being in the workspace they can be reached even without the explicit intention.

But now the question is simple: why these singularities are so problematic?

The answer deals with the possibilities of motion of the structure. In fact, when in that condition, the robot looses a grade of mobility and a generic trajectory cannot be followed by the end effector.

From a mathematical point of view a singularity born when the jacobian matrix associated to that particular state of the robot becomes non-invertible and this happens when its rows or columns are dependent. So the rank decreases and this means that one degree of freedom is lost.

Moreover there's the problem of the inverse dynamic which has infinite solution in this case. This means that the system is not controllable at all and some problems may occur.

Last but not least, in case of singularity also velocities and forces are involved.

For what concerns velocities it can be noted that if a singular configuration happens, slow motion of end effector can be related to very high(even near to infinite) velocities of joints.

Talking about forces two things happen:

- discharge of forces: if a mechanical structure is in singularity with the end effector facing the ground and it is sustaining a load, the torques of joints are zero because this is a natural configuration of balancing the gravity force. To understand better the situation just think about the singularity position of human standing up which requires no fatigue
- control possibilities: when a manipulator is all stretched the control of forces applied by end effector on the environment is problematic. As usual just think of a precision of human forearm when it has to write something on a paper far from the body

The analysed haptic interface is a proper robotic structure where the tendons act as joints and the central bracelet as the end effector. So also for this particular device it is necessary to talk about the singularity configurations.

They are reached when the bracelet touches one of the three lateral faces and in particular when even one unique anchor point enters in the virtual faces. These ones are outlined by three motors as vertices so the one at the top of the tetrahedral structure and two in the basis.

If a singular configuration is reached the inverse dynamic can't be computed due to the fact that the jacobian matrix is not-invertible, so a control problem happens.

Mathematically speaking there are problems when J^T becomes singular due to the orthogonality between the lower strings and the desired force.

To dimension the whole structure in order to avoid singularity configurations some geometric considerations have to be done.

Firstly, it is necessary to build a virtual sphere of radius R1 with origin in the tetrahedron's centroid which represents the desired workspace where the bracelet's centre must freely move avoiding singular configurations. The radius of R1 is chosen of 20 cm. Moreover a maximum sphere of radius R3 inscribed in the tetrahedron and tangent to its faces can be then computed.

Its radius is obtained as the sum of R1 one and the radius of R2 which is the minimum sphere which circumscribes the bracelet comprehending the anchor points.

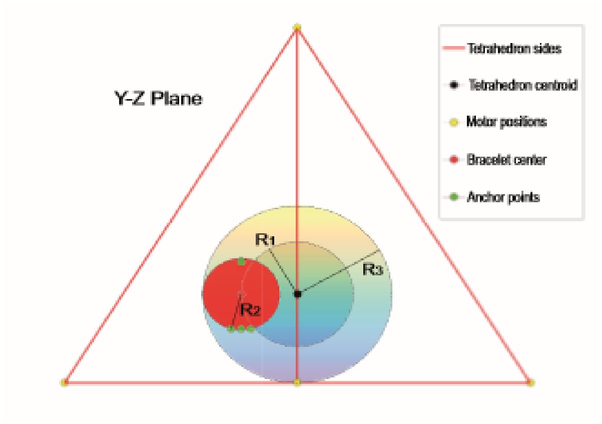


Figure 6.28: Virtual spheres to evaluate singularities and reachable workspace

Thanks to the knowledge of R3 the length of each tetrahedron's side in order to avoid singularities can be obtained.

$$L = \frac{12}{\sqrt{6}} * R3 \tag{17}$$

In order to let the operator knowing when he is approaching to a singular configuration the Rviz visualization tool can be used.

As a matter of fact it can be possible to implement a node which allows to publish a marker describing the bracelet which gradually changes colour if this is touching the external faces.

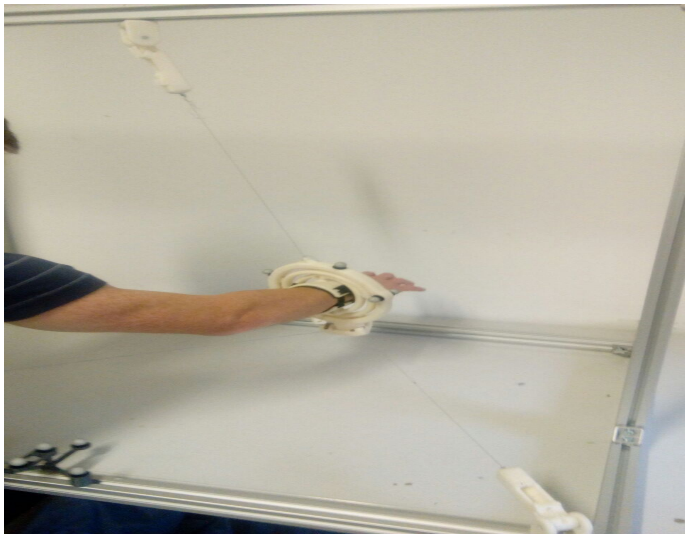


Figure 6.29: Real singular configuration in haptic interface

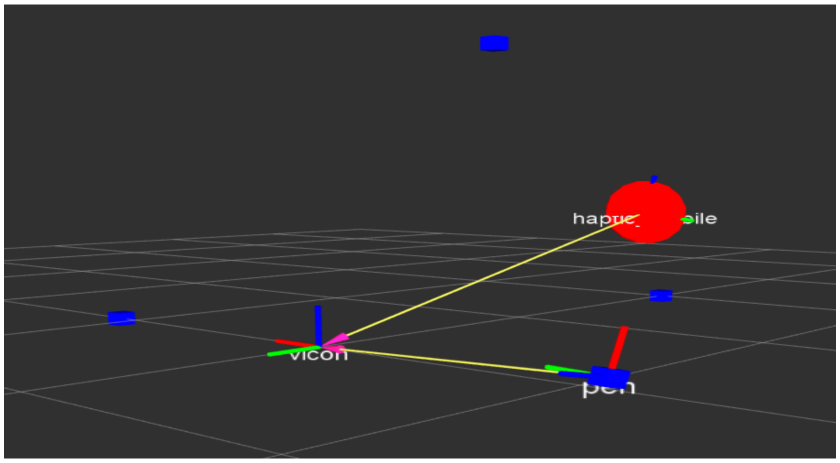


Figure 6.30: Rviz singular configuration in haptic interface

Another choice that can be made is to let the faces colouring gradually instead of the bracelet.

7 Chapter: Conclusions

7.1 Summary of thesis activity

In order to summarize all the aspects met during the thesis activity, it can be said that the core of the project deals with the construction of a properly working motor's module in which both the mechanical and software part are taken into consideration. By the use of four working module of the same type mounted on the haptic structure, the operator is able to control 6 Dof of position and orientation of a remote device while sensing 3 Cartesian Dof forces. In particular the thesis work threats more in detail the software part based on the implementation of both a position control and a force one.

To reach the results an initial overview of the already existing haptic systems and a successive focus on the cable-driven one with TSA motors has been done.

The central part of thesis work deals with the analysis of the mechanics of each component of the haptic structure because the implementation of a code able to realize a satisfying control in terms of speed, precision and steady state error starts from this aspect.

Of consequence the core Chapters are the fourth and fifth one where the used programming techniques are reported and the code implementation of both the control algorithms is described deeply together with the plot results.

Some other important aspects of the thesis are shown in Chapter 6 where the reader can have a global vision of the entire passage from the definition of Cartesian set points by the operator and their transformations into joint space references for each motor.

The importance to have a good communication between the various parts and a precise control enable to avoid problem of stability and permits to reach the required precision in terms of motion and manipulation tasks of the teleoperated device.

7.2 Future developments

Despite the haptic system has been taken to a quite advanced step from the software point of view, there's a lot of remaining job to be carried on in order to conclude the project. The developed parts represent the first programming phase but ensure a base for the future full software.

The first passage to finish the project is to build the whole structure with all the motors inside their proper modules and to dispose smartly the markers on the central bracelet.

Another aspect to face up to is the precise calibration of all the four force sensors.

Finally the validation of the firmware on the four controllers has to be done and some experimental results in terms of position and force response need to be taken out.

List of Figures

References

[1] https://www.researchgate.net/file.PostFileLoader.html

[2] https://pure.tue.nl/ws/files/4419568/656592.pdf

[3] http://hackaday.com/2015/01/17/twisted-string-actuators

[4] https://en.wikipedia.org/wiki/Telerobotics

[5] http://www.dexmart.eu/fileadmin/dexmart/public-website/downloads/presentations/

[6] http://www.pelicanrope.com/pdfs/DyneemaSK75-Tech-Sheet.pdf

[7] http://www.cim.mcgill.ca/ haptic/pub/VH-ET-AL-SR-04.pdf

[8] G.Palli, C. Natale, C. May, C. Melchiorri, and T. Wurtz, Modeling and control of the twisted string actuation system, IEEE/ASME Trans.on Mechatronics, vol. 18, no. 2, pp. 664673, 2013.

[9] A.Pepe, M.Hosseini, U.Scarcia, G.Palli and C.Melchiorri, Development of an Haptic Interface Based on Twisted String Actuators, Advanced Intelligent Mechatronics, 2017

[10] G.Palli, Slides Corso Automation Software and Design Patterns, Unibo, a.a 2016-2017

[11] https://en.wikipedia.org/wiki/Pulse-width-modulation

[12] https://en.wikipedia.org/wiki/Client/server-model

[13] http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/pro/pro10-b5-navigazionerobot.pdf

[14] https://www.vicon.com/what-is-motion-capture

[15] https://en.wikipedia.org/wiki/Rotary-encoder

[16] https://www.arduino.cc/reference

[17] www.miro.ing.unitn.it/.../Robotica/Cinematica