Universitat de Girona
Escola Politècnica Superior

Incoming Exchange Student - Master Thesis

Erasmus ■ Techno □ Other (specify):

Title of master course: VIBOT

Title of master thesis: Gesture Based HMI for Intervention Underwater Robots

Document: Master Thesis

Student (Name & Surname)**:** Daenen Ruben

EPS Advisor: Pere Ridao Rodrìguez & Josep Bosch Alay
Department: Arquitectura i Tecnologia de Computadors

Delivered on (month/year): June 2015

GESTURE BASED HMI FOR INTERVENTION UNDERWATER ROBOTS

By
Ruben Daenen

Department: Industrial Engineering in Energy

Defended on
June 15, 2015

Internal Promoters:
Johan Baeten (KULeuven, Belgium)
Leo Rutten (KULeuven, Belgium)

External Promoters:
Pere Ridao Rodriguez (UDG, Spain)
Josep Bosh Alay (UDG, Spain)

Abstract

Recently there has been an increasing interest for underwater robotics. Great efforts have been made in finding techniques to control underwater robots. In this master thesis, a new technique is introduced. This technique uses hand gestures detected by a Leap Motion device for piloting a Remotely Operated Vehicle (ROV). To use this hand gestures a literature review has been done in this master thesis to know which gestures can be used for each command. The technique for controlling a ROV with the Leap Motion device is more intuitive than controlling the robots with already known techniques like a joystick or a keyboard. This master thesis also explores how hand gestures can be used for specifying targets and grasping operations in a very intuitive way for Intervention Autonomous Underwater Vehicles (I-AUV). For testing the hand gestures on the robots simulators have been used. After the simulation the new technique is tested on a real robot, called the Girona500.

Acknowledgements

This master thesis could not have been performed if not for the assistance and support of many individuals.

Foremost, I would like to express my sincere gratitude to my advisor Josep Bosh Alay for the continuous support of my master thesis. In addition, a thank you to Prof. Pere Ridao Rodriguez, who introduced me to Robotics and the CIRS research center of Girona. Their guidance helped me in all the time of this master thesis.

Besides my external promoters, I would like to thank the people in the CIRS research center of Girona for their assistance.

I am also grateful to my internal promoters, Prof. Johan Baeten and Prof. Leo Rutten, for their guidance in my master thesis.

Last but not least, I am extremely grateful for the opportunity to be able to go in Erasmus in Girona, Spain. I would like to thank therefore all the people who made this possible for me.

Table Of Contents

1 Introduction

The ocean covers 71 percent of the Earth's surface and has a big impact on the future existence of the human beings. If the attention is focused on land and atmospheric issues, the ocean will be overlooked. Because of this the humans are not been able to explore the full depths of the ocean and the resources which are stored in the ocean. The interest for knowledge of the ocean gets larger and larger. Because working underwater is dangerous and difficult for humans, underwater robots can help us to understand better marine and other environmental issues, protect the ocean resources of the earth from pollution, and efficiently utilize them for human welfare. The underwater robot which is used in this master thesis is the Girona500 [1], [2].

The Girona500 is an Autonomous Underwater Vehicle (AUV) which is reconfigurable, i.e. it can be equipped with different sensors or actuators depending on the purpose of the mission where it is used. The robot is composed by three torpedo-shaped hulls that offer a good hydrodynamic performance and a large space for housing the equipments while maintaining a compact size allowing to operate the vehicle from small boats. The vehicle can be used in “intervention”-mode when a 4 degrees of freedom arm is integrated in it, as seen in figure 1 [3], [4].

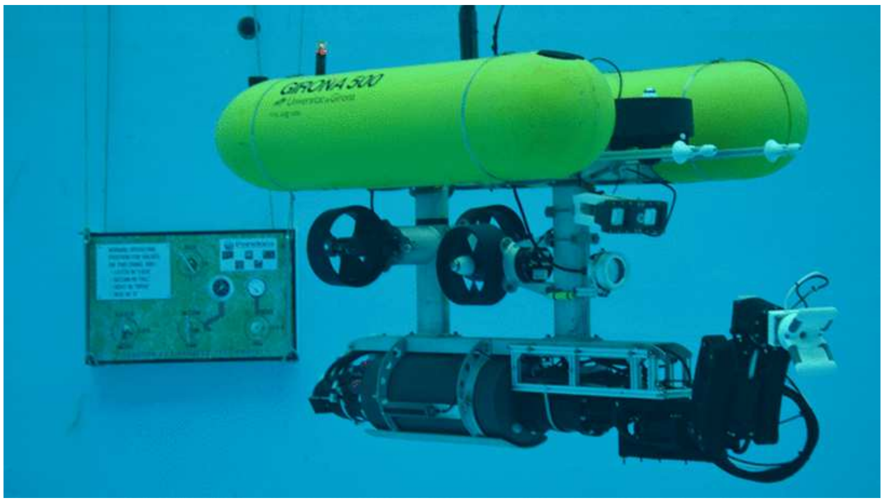


Figure 1: The Girona500 in the water [13].

Nowadays the GIRONA500 and the robotic arm of this robot can be controlled with a joystick or with a keyboard when the vehicle is used in ROV-mode. The purpose of this master thesis is to explore the different capabilities of the use of a Leap Motion device for intervention autonomous robots (I-AUV). The Leap Motion device is a device that tracks the movements of the user hands and fingers as input. The device sends an upwards infrared light. When the light rays bump against the hands, the rays will reflect back to the device and the device knows the information of the hands and fingers. The Leap Motion device is shown is figure 2 [5].

Figure 2: The Leap Motion device [5].

In the first part of this master thesis it will be studied which set of gestures may be used to efficiently guide the vehicle with the pilot hands. The system for controlling the underwater-robot and the robot-hand will be prototyped and tested using UWsim, an AUV simulator. After this test the system will be executed on the real underwater-robot. In order to demonstrate the flexibility and the modularity of the control system built, the same module will be used for piloting a Turtlebot, a land robot for educational robotics. In the second part of the master thesis, the Leap Motion device will be used for defining and sending an intervention task to the robot in a very intuitive way for the final user.

The control system of the Girona500 was first tested in a simulator, called UWSim and after that, it was executed a pool, located in the CIRS research center of Girona. The Turtlebot and the intervention task were only tested in the simulator due time and resources constraints.

The remainder of this master thesis is organized as follows. Section 1 is the literature review which is done for the set of gestures that may be used for guiding the vehicle with the Leap Motion device. Section 2 presents the equipments which are used for executing the tests. These are the Girona500, the Turtlebot and the Leap Motion device. Section 3 describes the software (ROS and Cola2) which are used for controlling the Girona500 and the Turtlebot. Sections 4 and 5 describe the 2 parts of the project and how they are implemented in the software architecture of the 2 robots. Section 6 presents the results which have been obtained by the tests. The last section presents the conclusions of this master thesis.

2 Literature review

One of the main objectives of this thesis is to control an underwater robot using hand gestures. Therefore it has been necessary to explore which set of gestures is the most appropriate for this task.

At first, an extensive review of the implementation of the Leap Motion device in the application “The Parrot AR Drone” is presented due to the big similarities between Unmanned Aerial Vehicles (UAVs) and underwater vehicles. Later on, other applications that use the Leap Motion device in robotics platform are presented.

2.1 Application: the Parrot AR drone

The Parrot AR drone is an UAV (Unmanned Aired Vehicle), this is an unmanned aircraft which can receive signals from another device. The B.S. University of Central Florida did some research about devices to sent commands to The Parrot AR drone. The devices that the university tested are the Microsoft Kinect device and the Leap Motion device. The AR drone is shown in figure 3 [6], [7].

Figure 3: The Parrot AR Drone [52].

2.1.1 Upper Body Interaction Techniques

The Parrot AR drone was tested for controlling with the Microsoft Kinect device for upper-body interaction technique development. The device can detect all the movements of the upper-part of the user's body. To control the drone some techniques were introduced so the drone knows on which gesture it has to react. The following 5 techniques were created in order to select the most comfortable and the most natural one. These are the 5 techniques [6], [7].

- First Person
- Game Controller
- The Throne
- The Proxy Manipulation
- The Seated Proxy Manipulation

2.1.1.1 First Person

This technique, shown in figure 4, is based on the movements of an airplane. The user has to pretend that he/she is an airplane and his/hers arms will be the wings of that airplane. When the gestures are executed the drone will go to the direction the user leans to [6], [7].

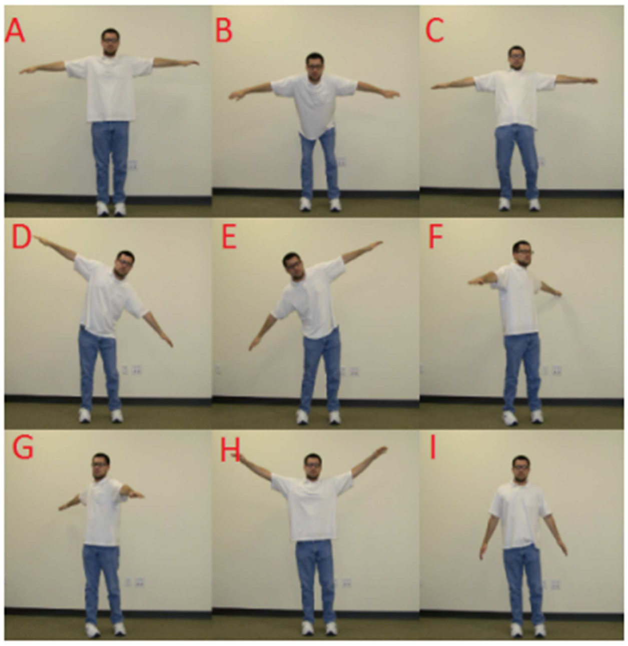


Figure 4: The First Person gestural commands. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

The movements are the following:

A. This is the base pose. The airplane has his wings horizontal. The drone will not move.
B. For going forward, the user needs to bend over, so that his body is in front of his legs. The hands stay horizontal.
C. The user will lean back for letting the drone go backwards. Also here the user's arms stays horizontal.
D. To sway to the left the user will lean to the left. Here his hands aren't horizontal anymore.
E. To sway to the right the user will now lean to the right.
F. The user will turn his body to the left if he wants that the drone turns to the left.
G. For turning to the right the user will turn to the right.
H. To let the drone ascend, the hands of the user will be lifted.
I. The hands of the user will lower if the drone needs to descend [6], [7].

Because the metaphor is natural, the expectation of this gesture was very positive. Because of the simple commands this technique is practical for using. It was also possible to combine the commands so that the user can lean forward and lean his body to a side, so a forward sway movement will be executed by the drone. The disadvantage is that those combined commands are uncomfortable [6], [7].

2.1.1.2 Game Controller

This technique, shown in figure 5, is developed to create a game controller with the arms of the user. A game controller has 2 joysticks. In a shooter game the joysticks are used for the following commands [6], [7].

- The left joystick controls the translation of the character
- The right joystick controls the rotation of the character

For controlling the drone the arms of the user will be the joysticks. The left and right arm will be respectively the left and the right joystick. The neutral position is that the arms are leaning on the arms of a chair in that way that the shoulders are relaxed. For executing the commands the elbow always stays on the same place but the direction of the lower arm will be the same as moving the joystick of the game controller in the same direction [6], [7].

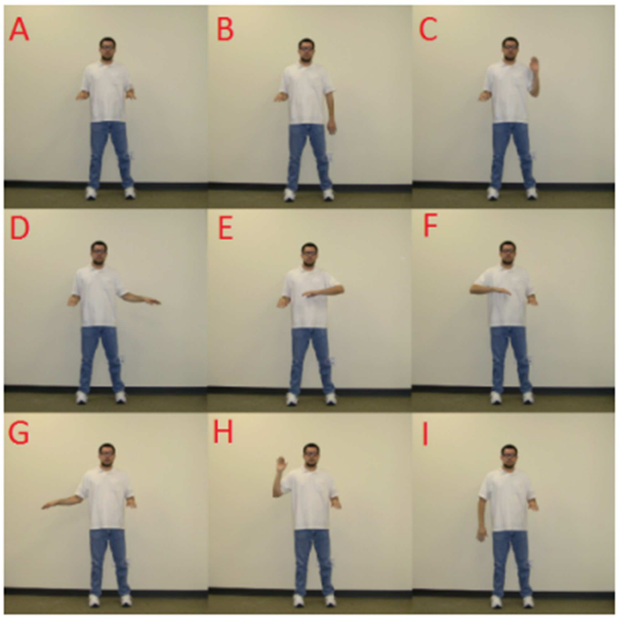


Figure 5: The Game Controller gestural commands. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

In comparison with the first person technique the arms are in a more comfortable position. Because the arms are independent of each other, the combination of translation and rotation is very easy and more comfortable to do than the first person technique [6], [7].

2.1.1.3 The Throne

This technique, shown in figure 6, is based on the orders of the king/queen who is sitting on his/her throne. The reasoning of this technique is that the king/queen gives commands with one arm to his/her workers, so that the commander uses the minimal amount of energy [6], [7].

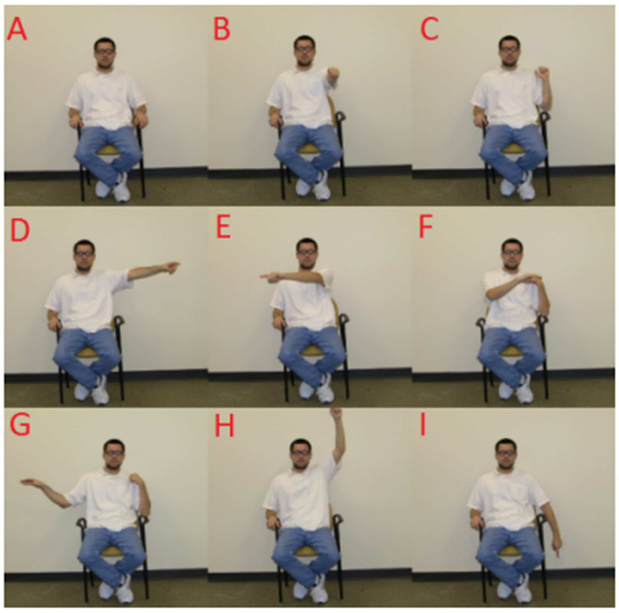


Figure 6: The Throne's gestural commands. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

For this technique the hand movements are very simple and it is a very comfortable technique because the user can sit and can control the drone with one hand. If the user wants to combine the commands it may need 2 arms [6], [7].

2.1.1.4 The Proxy Manipulation

The idea behind this technique is that the user has the drone in his hands and move it in the direction where he wants the drone to go to. The movements of the imaginary device in the user's hands will be transferred to the real device. The technique is shown in figure 7 [6], [7].

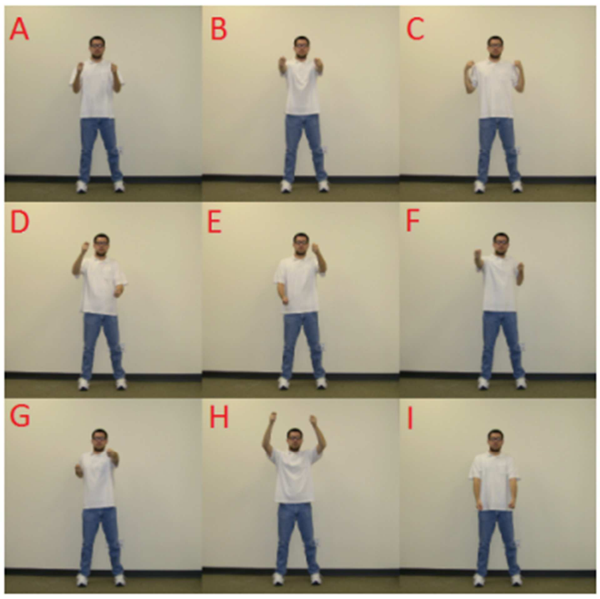


Figure 7: The Standing Proxy gestural commands. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

This technique seems very easy because we have the imaginary drone in our hands. However the complex commands could be lost because all the commands need the 2 hands. For example: turning while moving forward is difficult because in one case the both arms are forward and in the other one arm is not forward [6], [7].

2.1.1.5 The Seated Proxy

This technique, shown in figure 8, is based on the proxy manipulation. It is approximately the same but with the difference that the user now is sitting on a chair what makes it more comfortable, because the legs are relaxed [6], [7].

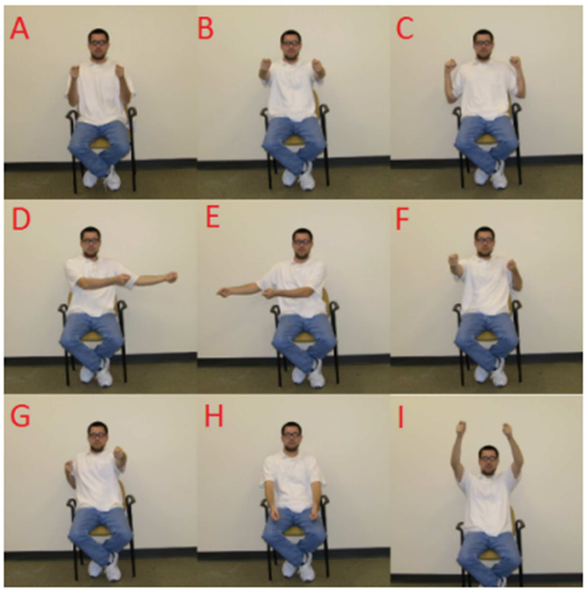


Figure 8: The Seated Proxy gestural commands. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

The commands of the seated proxy are the same as the manipulation proxy. Only the sway-commands are different. This technique is in case of comfort better than the proxy manipulation because the user can relax his legs. Even tough, doing the complex commands stays also the same as the proxy manipulation [6], [7].

2.1.1.6 Result

Now that all the techniques were introduced, the creators of these techniques did a research to see which technique is the best one. To know this a user study was performed, every user tried every technique and gave feedback for this. The results were the following ones [6], [7].

- The users appreciate the techniques based on easy, understandable metaphors. The technique Proxy Manipulation technique was regardless of the posture the most logical technique to follow and to understand.
- The other techniques (First Person, Game Controller, The Throne) were regarded as fun or perceived as natural.
- Because there are different results in factors of comfort, naturalness and perception, the conclusion is that there is a correct usage for each of the techniques when applied in a proper domain.
- For non-recreational use, the Proxy techniques are the most suitable.
- For recreational use, the First Person technique is the most suitable [6], [7].

The conclusion is that the technique which provides the best user experience are the ones with the metaphors which are closely associated with real life [6], [7].

2.1.2 Hand-and-Finger Interaction Techniques

For the drone-application also the Leap Motion device is implemented. The Leap Motion is a device that tracks movements of the user's hands. Therefore the techniques which were created for the Microsoft Kinect device have to be translated into hand and fingers movements. In total only 3 of the 5 techniques are useful for controlling the drone with the Leap Motion device [6], [7].

- The First Person
- The Throne
- The Scaled Proxy

It is not possible to translate “The Game Controller” into hand and fingers gestures because with the Microsoft Kinect the whole arms could be detected. Moving the arms in every direction, as shown in figure 5, is possible. But when only the hand, and not the entire body, can be detected and the hand has to turn for example 180° degrees and more. This is almost impossible for the user. Therefore this gesture will be deleted for the Leap Motion device [6], [7].

The other proxy technique is not useful to implement with the Leap Motion device because the device cannot detect if the user sits or not [6], [7].

2.1.2.1 First Person

The technique, shown in figure 9, stays the same: the drone will go to the direction which the user leans to. For the Leap Motion only the hands and fingers are taken into account and therefore they will do all the movements. For this technique only the right hand needs to be used. In this technique it seems that the hand pretends to be the drone. The top of the fingers are the ones which accord to the front of the drone and the wrist accords to the back of the drone [6], [7].

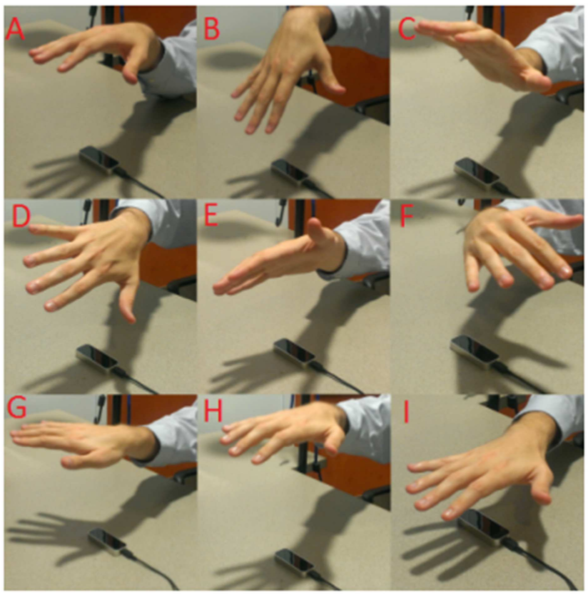


Figure 9: The First Person gestural commands - hand and fingers. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

A. This is the neutral position. The right hand is above the Leap Motion device at a height of more or less 12 centimeters.
B. For going forward, the hand needs to lean forward so the fingertips will be lower than the wrist.
C. The wrist of the right hand of the user must be lower than the fingertips for letting the drone go backwards.
D. To sway to the left the user's right hand will turn so the little finger will be higher than the thumb.
E. To sway to the right the right hand needs to turn so the thumb will be higher than the little finger.
F. The right hand has to turn to the left for turning the drone to the left.
G. For turning to the right the user's right hand has to turn to the right.
H. To let the drone ascend, the right hand of the user will be lifted.
I. The right hand of the user must be lower if the drone needs to descend [6], [7].

All the commands can be done with only rotating the hand, except for the vertical movement. The rotation of the hand is enough to move the drone horizontally. This technique seems easy to understand [6], [7].

2.1.2.2 The Throne

With the Throne-technique the drone can be controlled by using of only 1 hand, what is the same as the First Person technique. The technique is shown in figure 10 [6], [7].

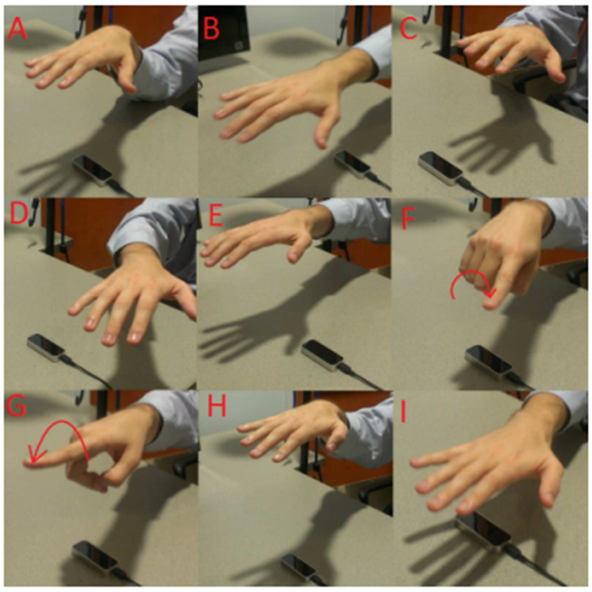


Figure 10: The Throne's gestural commands - hand and fingers. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

A. This is the base pose. The right hand is above the Leap Motion device at a height of more or less 12 centimeters.
B. For going forward, the right hand of the user must be further than the Leap Motion device.
C. The right hand of the user must be closer to the body than the Leap Motion device.
D. To sway to the left the user's right hand must be at the left side of the device.
E. For swaying to the right the user needs to put his right hand at the right side of the device.
F. The user must make a circle gesture with his index-finger of his right hand to turn the drone to the left. The circle gesture is counter-clockwise.
G. For turning to the right the user uses his index-finger of his right hand to make a clockwise circle gesture.
H. To let the drone ascend, the right hand of the user will be lifted.
I. The right hand of the user must be lower if the drone needs to descend [6], [7].

This technique also uses one hand, so it seems very easy to understand and to use. It is possible to give complex commands to the drone. The circle gesture that is used for the turning of the drone is a good proposal for this command but it will take a lot of attention. Therefore this is maybe not the best option [6], [7].

2.1.2.3 The Scaled Proxy

This technique, shown in figure 11, is a scaled version of the upper body Proxy technique. For the Scaled Proxy technique the 2 hands are required to sent commands to the drone. This is called the scaled version because the arms still do the same movements but now it is only above the Leap Motion device. For this technique the hands are always making fists [6], [7].

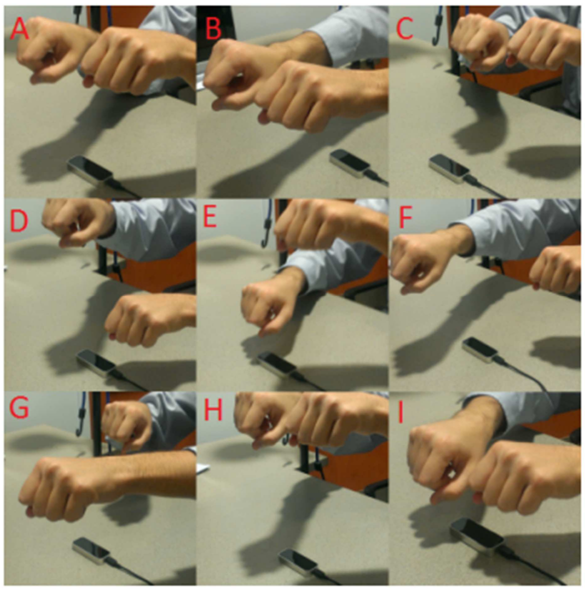


Figure 11: The Scaled Proxy gestural commands - hand and fingers. A: Base pose. B: Move Forward. C: Move Backward. D: Sway Left. E: Sway Right. F: Turn Left. G: Turn Right. H: Move Up. I: Move Down [7].

A. This is the neutral position. The hands of the user are at more or less 12 centimeters above the Leap Motion device.
B. For going forward, the user needs to push the drone forwards, so his hands will be further than the Leap Motion device.
C. The hands of the user will be closer to the user's body than the Leap Motion device. So the drone will go backwards.
D. To sway to the left the user will make sure that his right hand is higher than his left hand.
E. For sway to the right the user needs to put his left hand higher than the right hand.
F. The user could move right hand forwards and his left hand back for turning to the left. So the user takes the drone and the right wing will be pushed forward and the left wing will be pulled backwards so the drone makes a turn to the left.
G. For turning to the right the user moves his left hand forwards and his right hand backwards.
H. To let the drone ascend, the hands of the user will be lifted. The user will lift the drone.
I. The left hand of the user will lower if the drone needs to descend. The user puts the drone to a lower position [6], [7].

This method is easy to understand because it looks like you have the robot in your arms and take the robot to where you want it [6], [7].

2.1.2.4 Result

First person:
This technique only uses one hand and the rotation is enough for controlling the drone. Only the vertical movements are without a rotation. The hand will always be above the Leap Motion device. After a while the user's hand will get tired and it will be difficult to keep the hand in the air. For saving the energy of the user, the user can put his elbow on the table. Then the Leap Motion device can see this as a forward-gesture because the natural respond of the hand will make the forward-gesture. Also the turning in some directions is limited, so it will be hard to control the drone if the technique is not well tuned [6], [7].

Conclusion First Person:
The technique is easy to understand but it will be energy consuming to use and there are some limits for the angles. This technique is not recommended for using [6], [7].

The Throne:
This was the worst technique to use for the upper-body gestures. The Throne technique is a natural technique and hands do not need to rotate. Therefore there are no limits of the angles. The biggest obstacle is the circle-gesture for turning the drone. The user needs to do constantly the circle-gesture. If the Leap Motion device do not detect the circle-gesture, the drone can do some unexpected movements, because the hand will move a little, so the drone will also move [6], [7].

Conclusion The Throne:
This technique is more suitable and comfortable than the First Person technique, but for the turning-gestures there is still improvement possible [6], [7].

The Scaled Proxy:
This technique was the most successful for the upper-body gestures. For this technique there are 2 hands required. The rotation and the swaying are similar commands but they differ in the axis where they will be moved in [6], [7].

Conclusion The Scaled Proxy:
This technique is very successful because the Leap Motion device only offers a small place of movement. It is possible that the techniques with 1 hand sometimes give a wrong command. For this technique, which requires 2 hands, this chance is much smaller. That is the reason why the scaled proxy it the most valuable technique for controlling the drone with a Leap Motion device [6], [7].

2.2 Other Applications

2.2.1 Helicopter

Elephant Seven [8] built an innovative helicopter remote control. The helicopter can be controlled by a Leap Motion device. For controlling the helicopter Elephant Seven uses the First Person Technique. For a Helicopter there is no sway-commands possible. Therefore for turning to the left and to the right, Elephant Seven uses the sway-commands. The turning-commands are shown in figure 12 [8].

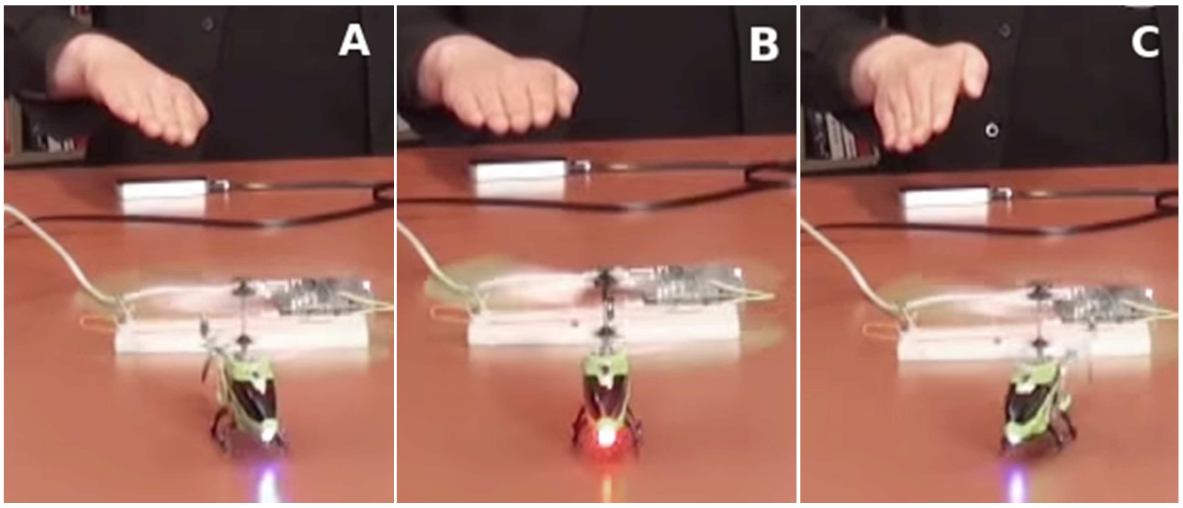


Figure 12: The First Person Changed commands for Helicopter Application. A: Turn Left. B: Base pose. C: Turn Right [54].

The remaining commands stay the same as the First Person Technique.

2.2.2 Arduino Car [9]

Nagrenda Babu Donthi Raju is a person who is specialized in Embedded Systems, Software Developing and Web Developing. The control of a self-made model car, the Arduino car, using the hand movements detected by a Leap Motion device. An Arduino Car is an autonomous vehicle and is shown in figure 13 [9], [10].

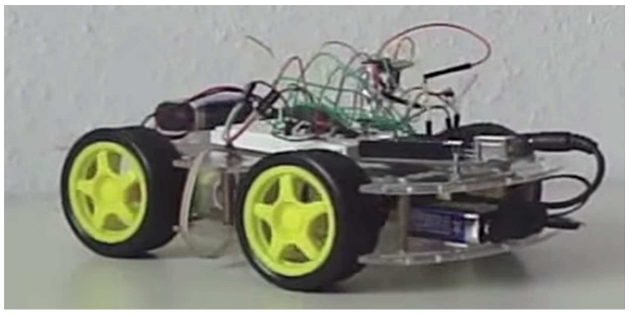


Figure 13: The Arduino Car of Negrenda Babu Donthi Raju [10].

For controlling the Arduino Car with the Leap Motion device, the First Person technique is used. Also here are the gestures of the turning changed in the gestures of swaying because with an Arduino Car the sway-movements are also not possible. The commands are shown in figure 14.

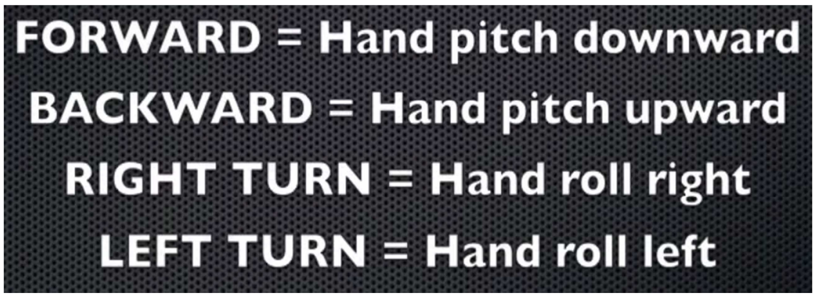

FORWARD = Hand pitch downward
BACKWARD = Hand pitch upward
RIGHT TURN = Hand roll right
LEFT TURN = Hand roll left

Figure 14: Instructions for the Arduino car [10].

2.2.3 Land Robot

The Mercer Engineering University started in 2014 a study for checking if a land robot can be controlled with a Leap Motion device. Participants tried to control this land robot with the Leap Motion device and they were been observed on how well they were able to control a simple robot. The commands of the robot with the Leap Motion device were the same as the Throne-technique. Also like all the other applications the sway-movements are not possible for this robot, so therefore again the turning-commands were changed into the sway-commands. This is shown in figure 15 [11].

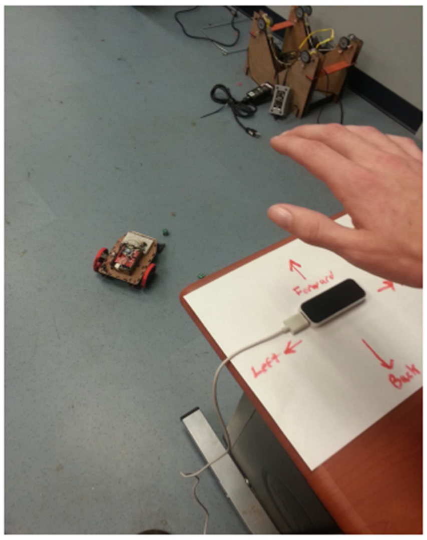


Figure 15: Controlling the robot with the Leap Motion Sensor [11].

2.2.4 Turtlebot

The Turtlebot is a land robot aimed for educational robotics with open-source software. This robot can drive around in flat environments and reconstruct the scene in 3D. The technical University of Munich [12] implemented a teleoperation control of the Turtlebot with the Leap Motion device. For this implementation also the First Person technique is used. Also for the Turtlebot the sway-movements are not possible. Therefore the sway-gestures will be changed to the turning-gestures. We get the next commands, shown in figure 16, for the Turtlebot [12].

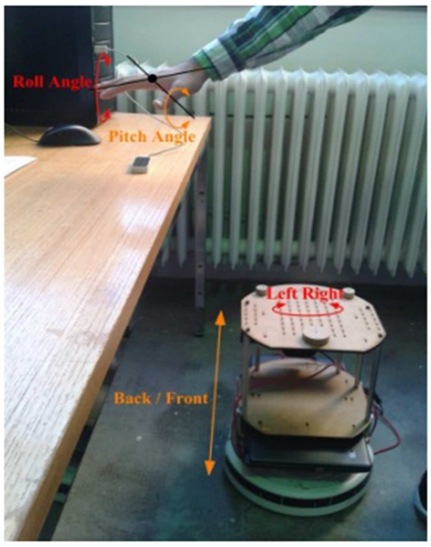


Figure 16: Roll/Pitch – Rotate/Move Translation Scheme [12].

2.2.5 Conclusion Other Applications

The most other applications use the First Person technique. Probably it is because it is an easy gesture to understand. For most of the applications the sway-movements are not used. Because the gestures for the turning in the most techniques is the most difficult command, the turning-gestures are changed into the sway-gestures.

2.3 Conclusion literature review

The technique which will be used for controlling the Girona500 is an adapted version of the First Person technique. The First Person technique is the most used technique. After implementing the 3 different techniques for controlling the Girona500, the First Person technique seems to be the easiest one, for controlling that vehicle.

3 Used Equipment

In this section, the equipment which is used will be introduced and described in detail. The used equipments are the Girona500, the Turtlebot and the Leap Motion device.

3.1 Girona500

The Girona500 is an AUV (Autonomous Underwater Vehicle) which is developed at the Underwater Robotics Laboratory of the University of Girona (Spain). This vehicle is designed to do some research for many applications. Examples of these applications are: classical sonar and video imaging surveys, challenging autonomous intervention tasks, … The Girona500 can operate in depths of 500m below the sea level. This is the reason why this vehicle is called Girona500 [3], [4].

3.1.1 Mechanical design

The Girona500 is based on an aluminium frame which supports 3 hulls. These hulls have a torpedo-shape with a diameter of 0,3m and a length of 1,5m. This means that there is a large space for storing the equipments. Apart of that, a compact size is maintained and the design offers a good hydrodynamic performance. The vehicle has a weight less than 200kg and the dimensions of the vehicle are the following ones:

- height = 1m
- width = 1m
- length = 1,5m [3], [4].

The vehicle consists of a main frame that is the base of the vehicle. There are 2 T-shaped aluminium pillars which are attached to 3 U-shaped profiles. These U-shaped profiles serve as backbone to the 3 hulls and work as ducts to convey the wet cables for power and communications. For protecting the sensible equipment and to reduce the drag of the vehicle, the 3 hulls are covered with a skin. This skin is composed by a thermoformed ABS plastic. The Girona500 without this skin can be seen in figure 17 [4].

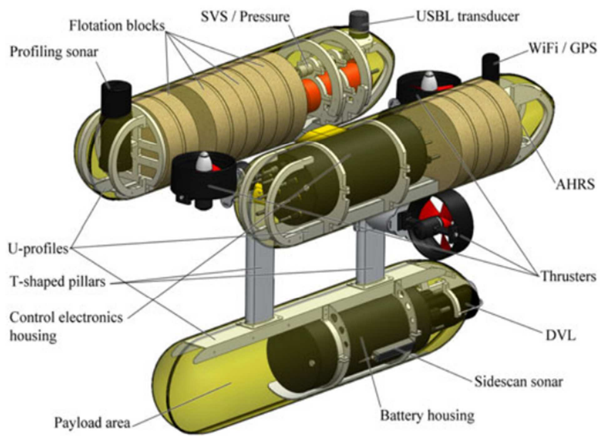


Figure 17: Girona500 AUV internals [4].

The Girona500 has 3 hulls: 2 upper hulls and 1 lower one. 1 upper hull contains flotation modules which are made of an epoxy composite foam. These modules make the vehicle neutrally buoyant. The other upper hull contains a cylindrical pressure housing which contains the control electronics. This housing is also positively buoyant. The lower hull also has a cylindrical pressure housing which includes the battery cluster, some power electronics, the payload and other heavy elements. It is clear that the light elements are located in the upper hulls and the heavy elements in the lower hull. This is for creating a very stable platform in roll and pitch if the vertical separation between the center of gravity and the center of buoyancy increases. The distance between these centers will depend on the configuration of the vehicle [4].

A robotic arm with 4DoF (Degrees of Freedom) can be added to the lowest hull. The manipulator is installed on the frontal vehicle frame to allow turning a valve on vertical panels. The manipulator is only able to control 4 DoFs: the Cartesian positions and roll. For making sure that the manipulator can reach all the places between his limits, the manipulator consists out of 4 joints. At the end of the manipulator there is a customized end-effector. This end-effector has been developed to manipulate a T-bar handle valve and consists out of 3 modules [13], [14]:

- A passive gripper: the gripper has a V-shape. This helps to guide the handle valve to the center of the end-effector.
- A camera in hand: A small camera films the environment in front of the end-effector. This provides the operator visual feedback of what the end-effector is manipulating. This camera is at the current state only used by the operator during a demonstration.
- A F/T- sensor: During the manipulation of the valve, this sensor provides information about the quality of the grasping and the necessary torque to turn the valve. This sensor is used to control if the end-effector is touching the handle valve [13].

This vehicle equipped with the arm can be seen in figure 18.

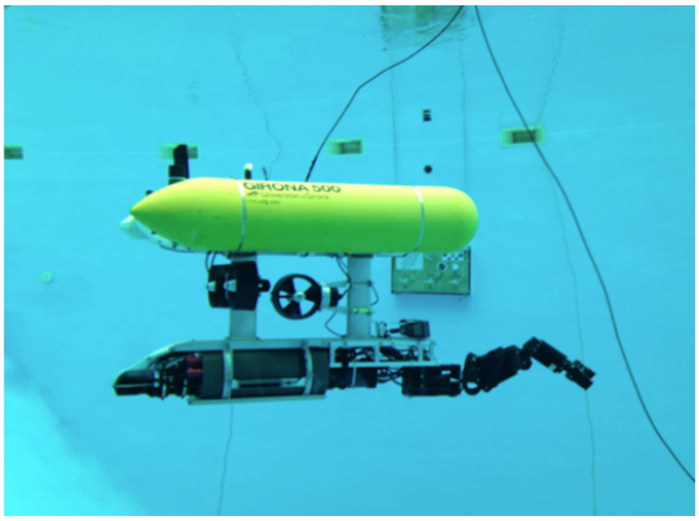


Figure 18: The Girona500 with the robotic arm AUV in the pool of CIRS (Girona) [14].

3.1.2 Proportional system

To satisfy the demands of the different applications of the Girona500, the propulsion system has been designed to allow different thrust configurations. This is ranging from the redundant vectored thrust typical of intervention ROVs (Remotely Operated underwater Vehicle) to more lightweight and efficient arrangements preferred for long endurance survey tasks. With reconfigurable mechanical parts and a junction box, these configurations can be easily implemented [4].

Figure 19 shows the 4 configurations:

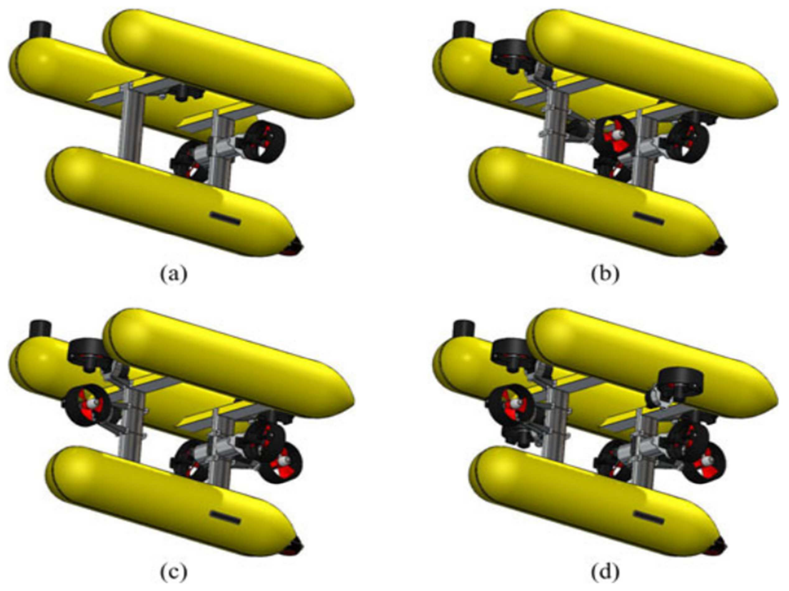


Figure 19: Some thruster configurations for the Girona500 propulsion system.
(a) 3 Thrusters, 3 DOF. (b) 5 Thrusters, 5 DOF. (c) 6 Thrusters, 5 DOF. (d) 8 Thrusters, 6DOF [4].

The first one (a) is equipped with 3 thrusters. 2 thrusters provide the horizontal movements (going forward and yaw-movement) and the other one makes sure that the robot rises or descends. So for controlling the robot there is a 3DOF (Degrees Of Freedom). In this configuration it is necessary to balance the vehicle carefully when a different payload is equipped [4].

The second configuration (b) is the standard Girona500 and has 4 thrusters. The added thruster makes it possible to do a sway-movement and a pitch-movement. In case there is a damaged thruster, the structural parts are such that these can be rearranged between the previous configuration (a) and this configuration (b). As such, the vehicle is rapidly back in operation despite the damage. The power-to-force ratio behaves linearly to the whole operating range. Because of that, the consumption of a single-thruster is more or less equivalent to the combined consumption of 2 thrusters, which produce half of the single-thruster. This makes sure that the addition of a thruster does not mean that the power consumption will increase [4].

Because it is possible that the water, where the vehicle is released, moves, it is possible to add 2 new thrusters (c). These thrusters are the bow and stern thrusters, which results that the vehicle gains control of the sway motion and redundancy in the horizontal plane. A lateral movement can be realized by installing a single-thruster in the middle of the 2 T-shaped pillars. Although it will lose the redundancy [4].

The last configuration (d) is the addition of 2 vertical thrusters. These thrusters add the roll DOF which makes sure that the vehicle is fully actuated. This configuration will only be used in applications where a high lifting thrust or a precise control is required. This is because of the passive stability of the vehicle [4].

3.1.3 Power System

The power source of the vehicle is a battery cluster which is composed of 24 small battery packs. These battery packs have a capacity of 95 Wh each. The battery packs have an output of 14,4 V and have its own integrated safety circuit. For the responsibility of the safety aspects by the charging and the discharging of the system, there are 3 high current controllers installed. Each controller has a capacity to manage up to 8 battery packs. The battery cluster, which contains the battery packs, can be charged by a fixed 20V-50A which comes from a AC-DC power supply. It takes 4h to charge this cluster completely. In the hull where the cluster is located, there are also 3 high-intensity DC-DC converters located. The converters makes sure that the output of the battery cluster is up to 48V and that the maximum power of it is 720W [4].

The results of these converters, which are connected in parallel, indicates that the power system can supply 2 different outputs: The first output is the one directly from the battery cluster (14,4V). This output provides the control and power of the upper part of the vehicle. The second output is the one after the converters and supply power to the propulsion system. Apart of this power supply there are also 2 connectors which charge the batteries with an external power supply and a serial communication system (RS232) for communicate with the upper hulls and to control the operation of the battery system. The other part of the power management takes place in the upper hulls. In that hull there are 9 different DC-DC converters which receive the unregulated 14,4V and they provide regular power to the subsystems. These subsystems are the two embedded computers, the LED lighting system and 7 other small DC-DC converters which supply power to primarily sensors. For having a more efficient power management during the execution, there are relay circuits placed before each DC-DC converters. These relays makes it possible to independently switch each one of the systems ON and OFF [4].

For bringing two different power supplies to the payload area, there is a cable which is connected from the upper hulls to the lower one. The first power supply provides the power for the sonars and the other underwater sensors. The other one is the power from the battery cluster which will be transported from the one to the other hull [4].

3.1.4 Simulator

There are different possibilities to experiment with underwater robots, but there are a lot of resources needed to experiment with this underwater vehicles. A water tank is one possibility, but this tank has to be deep enough for the systems before testing the underwater vehicle. This requires a lot of space and maintenance. Another possibility is to test the underwater robot in lakes or in the sea, where there is a lot of space but this increases the cost a lot and this requires special transport. Furthermore there is also the nature of the underwater environment where it is very difficult to observe the evolution of the running system. All this arguments make it very hard to experiment with real underwater robots. Therefore a good simulator is required.

A simulator is very useful when developing new control algorithms for the robot, where you can try very easily and quickly the new code without putting in risk the robots due to bugs in the code. Once the code had been tested in simulation it can be very easily transferred to the real robot due that the software architecture run in simulation is exactly the same than in the real robot. This simulation approach is called Hardware in the Loop (HIL). The simulation used for the Girona500 vehicle is called UWSim [15].

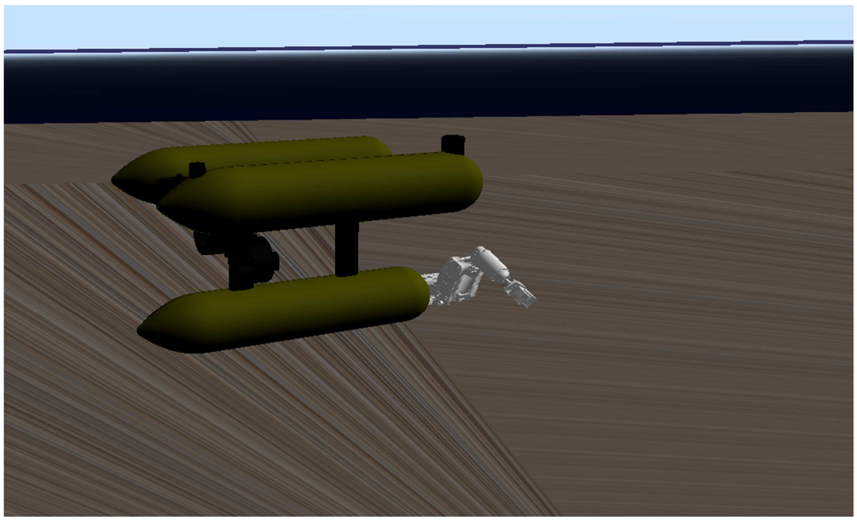


Figure 20: The Girona500 in the UWSim

With the UWSim it is possible to control the Girona500 and its robotic arm. In the simulator there is a panel installed. Attached to this panel there are different valves. This valve cannot be turned in simulation but currently it can be used to define a grasping position and orientation.

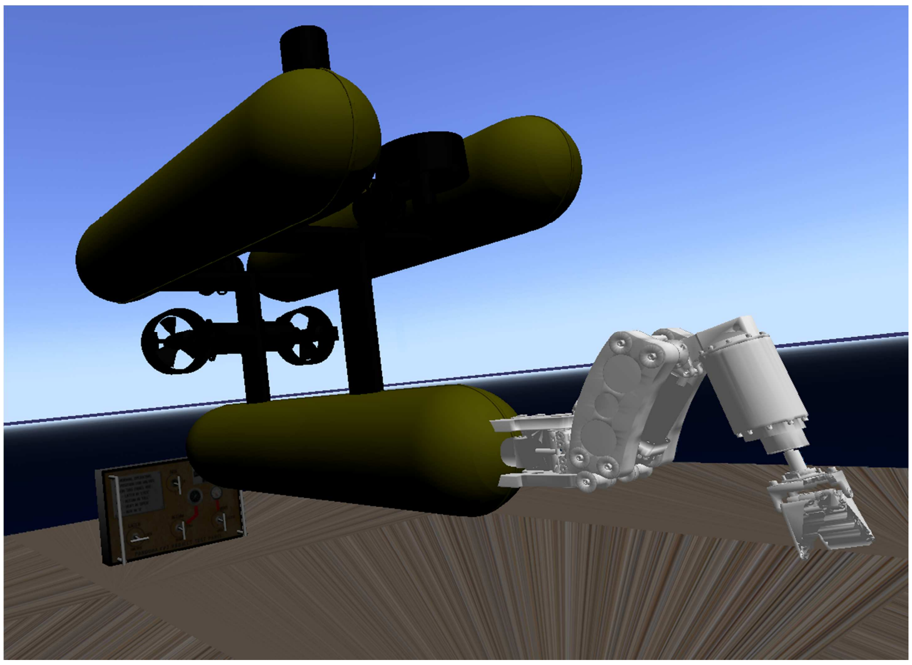


Figure 21: The Girona500 and the panel with the valves in the UWSim

3.2 Turtlebot

A Turtlebot, shown in figure 22, is a low-cost, personal robotic kit. This kit has an open-source software. The Turtlebot is used for programmers who are starting to use robots and for robotic research. The Turtlebot is designed by Willow Garage. This company develops hardware and open-source software for personal robotics applications [16], [17] [18], [19].

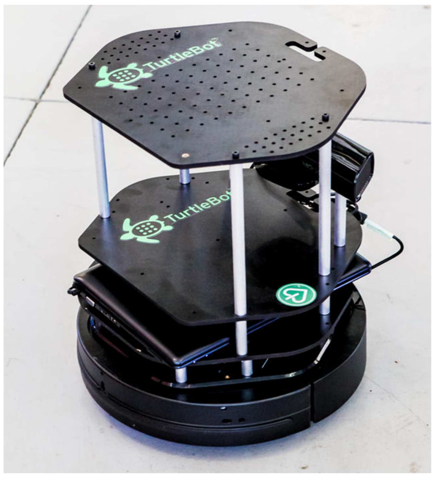


Figure 22: The Turtlebot [51].

3.2.1 Mechanical design

The Turtlebot is built on a iClebo Kuboki robot. This is a robot development kit. This development kit is the base of the Turtlebot and has a couple of inputs and outputs which the Turtlebot uses. At one input the Microsoft XBOX Kinect is connected. This device is a RGB-d sensor. A RGB-d sensor is a visual sensor which can detect the colors and can see the visible area in 3D. The Microsoft XBOX Kinect looks continuously to the environment of the robot. The commands which will be sent to the iRobot are coming from the laptop. The user sends the commands wirelessly to this laptop, so the robot will do the required commands. This wireless communication occurs via wifi. The different parts are shown in figure 23 [16], [17], [19], [20] [21], [22].

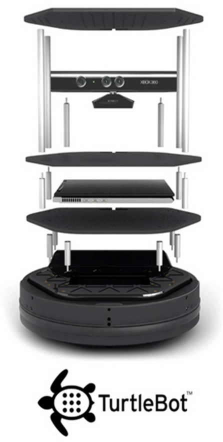


Figure 23: The Mechanical Design of the Turtlebot [20].

3.2.1.1 iClebo Kuboki Robot

iClebo Kobuki is a mobile research base , which can be seen in figure 24. This robot is specially designed for education and research on state of art robotics. Kobuki has a higly accurate odometry and long battery hours. The battery of the Kobuki provides also power supplies for the external laptop of the Turtlebot as well as the additional sensors (Kinect) and actuators [20], [21].

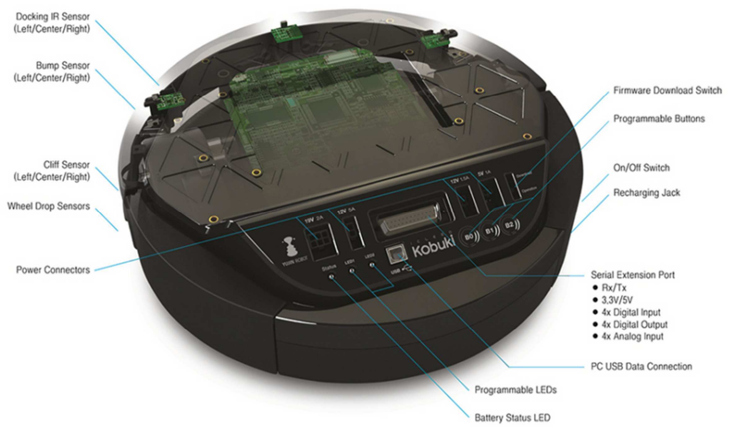


Figure 24: The iClebo Kobuki robot [20].

The specifications of the Kobuki are:

- The maximum speed = 0,7 m/s and the maximum turning speed = 180 °/s
- The odometry is built in 3-axis gyrometer and has a high-resolution wheel encoder
- The basic battery permits the robot to operate for 3 hours
- The Kobuki supports additional power to the sensors and actuators through 5V/1A , 12V/1,5A and 12V/5A power sockets [20], [21]

3.2.1.2 Microsoft XBOX Kinect

The Microsoft XBOX Kinect, shown in figure 25, is a computer vision-based system. This system uses an array of sensors to detect the environment. By combining the inputs from the array of cameras, the Kinect can see all the objects in the environment in 3 dimensions. This is possible because there are different cameras from other view-points. The cameras transmit infrared light and measure the "time of flight" after the light beam reflects off the objects. Because of this light is infrared, the problem with the visible light is also solved because the sensors only detect infrared light [7], [23].

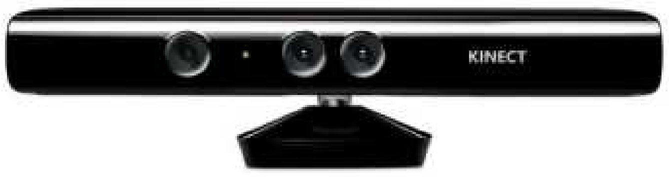


Figure 25: The Microsoft XBOX Kinect [7]

The Kinect is very cheap and a big advantage is that there are none physical devices needed to be used in order to interact with a system. A disadvantage is that if some object are behind other objects, the object which is the most far away is not visible for the device [7], [23].

3.2.1.3 Laptop

The laptop contains Linux as operating system. The laptop receives via wifi the commands from the user's computer and sent those commands with ROS (Robot Operating System) to the iRobot Create. A detailed description about ROS.

3.2.2 Simulation

The simulator of the Turtlebot is a visual application that allows the user to simulate the real Turtlebot. The user can sent his commands or programming code to the simulator. Because it can be tested in the simulator, the user does not have to take risks with the real Turtlebot. The name of this simulator is Gazebo and is shown in figure 26 [24].

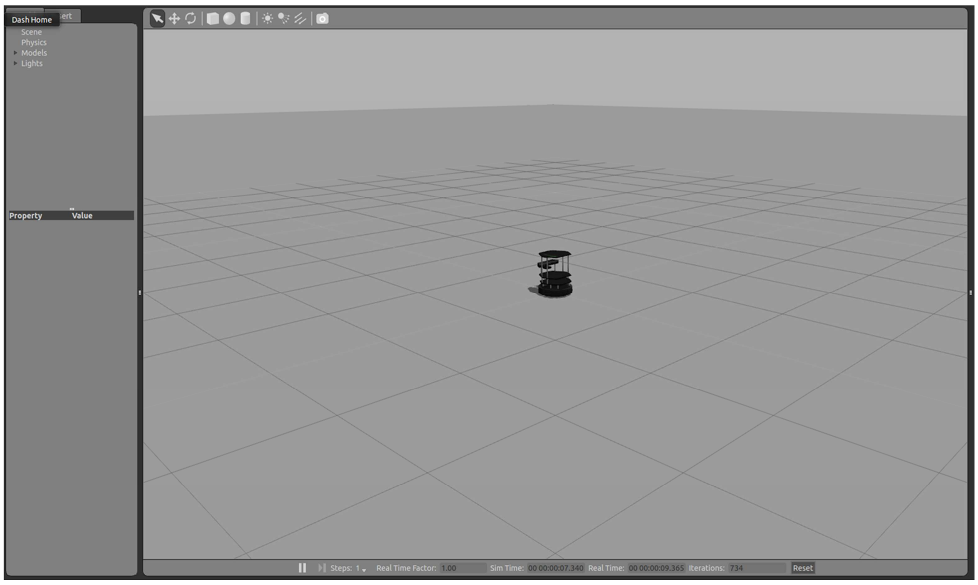


Figure 26: The Turtlebot in the Gazebo-simulator

Gazebo is designed to reproduce the dynamic environments accurately which the Turtlebot may encounter. Gazebo creates a 3D dynamic environment for the Turtlebot and this simulator is capable of recreating complex worlds. Gazebo is more than only a simulator of the robot, it also can do data visualization, simulation of remote environments and even reverse engineering of blackbox systems. The simulator makes it also possible to test different algorithms, design robots and perform regression testing using realistic scenarios. All simulated objects have mass, velocity, friction and a lot of other attributes. These attributes give a good image of the reality, because the behavior in the simulator is well imitated as the reality when the robot will hit an object [24].

3.3 Leap Motion

The Leap Motion controller (figure 27) is a device with sensors that translates hand movements into computer commands. The hand movements have to be above the device in the zone where the device can detect the hands. The Leap Motion device can measure the position of some parts of the hand and can detect the hand orientation. The device also knows some gestures [25], [26], [27]:

- Circle-gesture: a circle movement with a finger
- Key-tap-gesture: making a straight line with the hand and with stretched fingers
- Screen-tap-gesture: a forward movement to the screen with a finger
- Swipe-gesture: a downward movement with a finger [25], [26], [27]

Figure 27: The Leap Motion device [52].

Strengths of the Leap Motion device:

- It has a high accuracy
- The device is high sensitive for small movements
- Different parts of the hand (the fingertips, the wrist, …) can be detected [25], [26], [27]

Weaknesses of the Leap Motion device:

- If there are some parts of the hands which are not visible for the device, the accuracy decreases.
- If all the fingers are above each other, the device does not see where the upper fingers are located.
- If the fingers touch each other the device does not know which finger is of which hand
- If the fingers are folded, the same problem is happening [25], [26], [27].

Studies shows that the Leap Motion device has a high accuracy when the hands do not move. When the hands do some movements the accuracy of the device decreases [25], [26], [27].

The coordinate-system of the Leap Motion device is shown in figure 28:

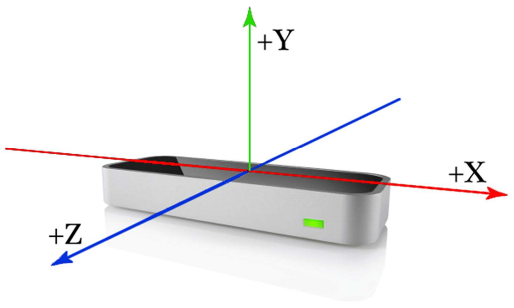


Figure 28: The Leap Motion Controller uses a right-handed Coordinate System [26].

The origin of the coordinate-system located at the top and on the center of the device. The device will be placed so that the user is in line with the positive side of the z-axis. The monitor screen is also in line with the z-axis, but it is placed behind the device (negative z-value). If this orientation is respected, the green LED is facing to the user [25], [26], [27].

4 Software

For controlling the Girona500 and the Turtlebot, the software architecture used for communication between hardware components and sending commands to the robot must be studied. This is ROS and COLA2.

4.1 ROS

At the first stage of the master thesis, the student has to become familiar with ROS (Robot Operating System). ROS is a flexible framework for writing robot software. This open-source operating system is a collection of tools, libraries and conventions that have the goal of simplifying the tasks of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS was created due to the complexity of sending commands to robots by writing code. ROS has a communication system which provides the communication patterns (publishing/subscribing and service/response). This communication system is on the low-level [28], [29], [30], [31], [32], [33], [34].

Figure 29: ROS (Robot Operating System) [35].

ROS consists out of seven different important components [31].

1. Roscore
   This is the component which will start ROS. Roscore consists of nodes and programs which are pre-requisites of a ROS-based system. Roscore starts the ROS-master, a ROS Parameter Server and a rosout-logging node. This makes sure that the nodes can communicate between each other [31] , [36].

2. ROS-packages
   The software which is implemented in ROS is organized in this packages. This is actually a folder which contains ROS-nodes, a ROS-independent library, a dataset, configuration files and a third party piece of software. Each package allows different programming languages (Python, C++, Java, …) [31] , [37].

3. ROS-nodes
   A node is a process that executes calculations. The nodes are the files which contains the code in any language the package, where the node is located, provides. A specific node had a specific name and will have his own functionality. For example: 1 node will control the wheels of a robot, 1 node will control the arm of that robot, 1 node will control the lights attached to the robot, … Nodes can communicate with each other using topics or services [31], [38].

   - Nodes can communicate with each other by publishing messages to Topics.
   - Another way of communicating is by using a Service.

4. Topics
   Topics are buses where nodes send their messages. Topics are actually the medium between 2 or more nodes. Topics will be used for continuous data flow. One of more nodes (Publishers) send messages on a topic. Some nodes are also connected to that topic but do not sent messages, but receive them. A Publisher-node only knows on from which topic it receives data, the node does not know from which node the message comes [31], [34], [39].

5. Services
   Communication on topics is only for communication in 1 way. There are listeners (Subscribers) and talkers (Publishers). For communicating in both ways (Request/Reply) a Service is needed. This service is defined by a pair of messages. One message is for the request and the other one is for the reply. A ROS-node offers a service and a client calls that service by sending a request message. After that the client will wait for the reply [31], [34], [40].

Figure 30 represents the communication between nodes by the 2 different ways:

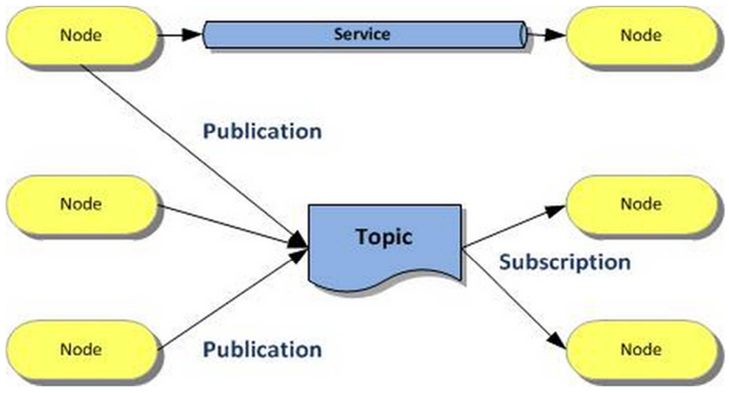


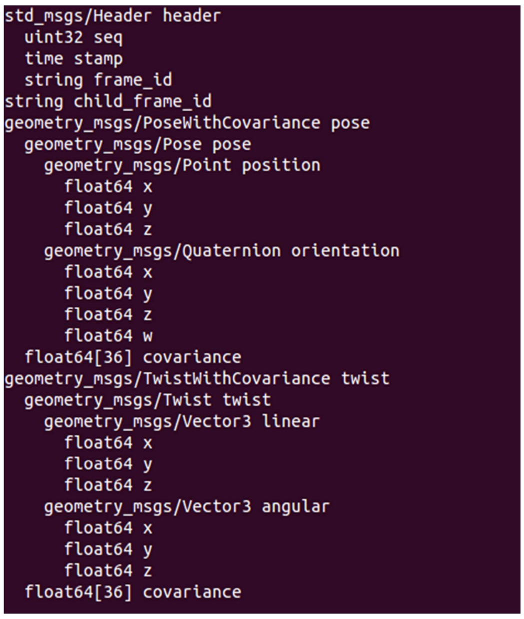
Figure 30: Communication between nodes in ROS [53].

6. ROS-messages
   Nodes use messages to communicate with each other. A message is a simple data structure which is a combination of standard primitive type of data (integer, floating point, boolean, time, duration and array) . In the message there are for example some commands which will be sent to the actuators for controlling a robot. The message will be divided in fields. Every field of the message means something [31] , [41], [42].

   For example: nav_msgs/Odometry message. This message-type exists of 4 different fields:

   - A Header standard type message. This message is used for communicate with timestamped data in a coordinate frame.
   - A String message which will give the id of the frame.
   - A Geometry message which indicates the position and the orientation.
   - A Geometry message which indicates the linear and angular speed [42].

   This is shown in figure 31.



```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

Figure 31: The message-type: nav_msgs/Odometry

A message can be created in every programming language, as long as all the fields of that message are filled in [42].

7. Launch-file
   A launch-file is a file which calls many different nodes at once. This file makes sure that the user does not have to execute all the nodes one by one. For example, by launching a launch-file the UWsim and Girona500 simulator can be started. All the nodes which are used for opening the UWsim, controlling the robot, visualizing everything, etc. are called in that launch-file [31].

4.2 COLA2

COLA2 (Component Oriented Layered-base Architecture for Autonomy) is the architecture which is used to control the Girona500 AUV. This control architecture is a set of software components which communicate with the hardware of the vehicle (actuators and sensors) and between them [14], [43], [44].

4.2.1 Theory

In figure 32 the architecture is described in a block diagram:

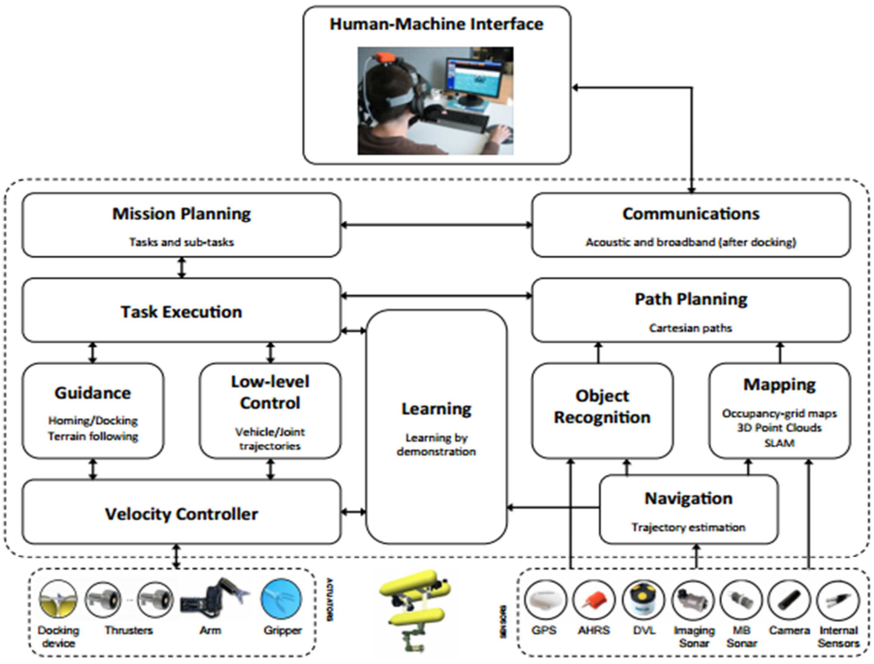


Figure 32: Block diagram of the architecture Cola2 [14].

The architecture consists of a vehicle interface module and device drivers for sensors and actuators. These drivers are in connection with a PPA-module (Perceive-Plan-Act). This module represents the world by using the values measured by the sensors. The navigation component estimates the trajectory of the vehicle and the Object Recognition and Mapping components. A multi-modal 3D representation of the world is being maintained by the Mapping component. This multi-modal will be used to provide feedback to the Navigation component. This will be done by a method called SLAM (Simultaneous Localization And Mapping). This method means that the vehicle drives in the water and builds a map. A priori knowledge to seek for matches between measurements and the object models will be used by the Object Recognition component [14], [44].

For monitoring the execution of the mission plan, the Mission Planning component will be used. The mission plan consists of a sequence of tasks which are received from the user. The user gives his inputs (position set points) into the Human-Machine Interface and these inputs will be sent to the Mission Planning with the Communications components. Each task which is sent during a mission will be executed by the Task Execution. This tasks use the Path Planning component and the Learning component. The Path Planning component generates a path which is the shortest and collision free. The Learning component performs manipulation tasks. The tasks are executed by sending the input of the user to the Guidance and Low-level Control components. These components generate from these position set points (input) the velocity set points. At the end, the Velocity controller transforms these velocity set points into commands which will be sent to the actuators [14], [44].

4.2.2 Implementation in ROS

Cola2 is a ROS-package located in the catkin-workspace. This catkin-workspace is a workspace where it is possible to modify, build and install packages. Cola2 exists out of many sub-packages [43], [45]:

- cola2_control
- cola2_detection
- cola2_launch
- cola2_lib
- cola2_navigation
- cola2_perception
- cola2_safety
- cola2_sim
- cola2_tests

These packages are making sure that there is a good communication between the hardware and the software. All those packages are responsible for a specific task for control the vehicle. There is a launch-file for starting the simulator of the Girona500 which is located in the sub-package cola2_launch. But this is the simulator of the basic Girona500 without the robotic arm. For controlling the arm as well, another simulator has to be launched. Because the robot is different, the launch-file is located in another package. This means that the other package: udg_pandora is communicating with the cola2-package. When the simulator is started, a lot of nodes are communicating with each other. For seeing which node is publishing/subscribing on which topic the RQT-graph can be called. This RQT-graph is shown in figure 33 [34], [43].

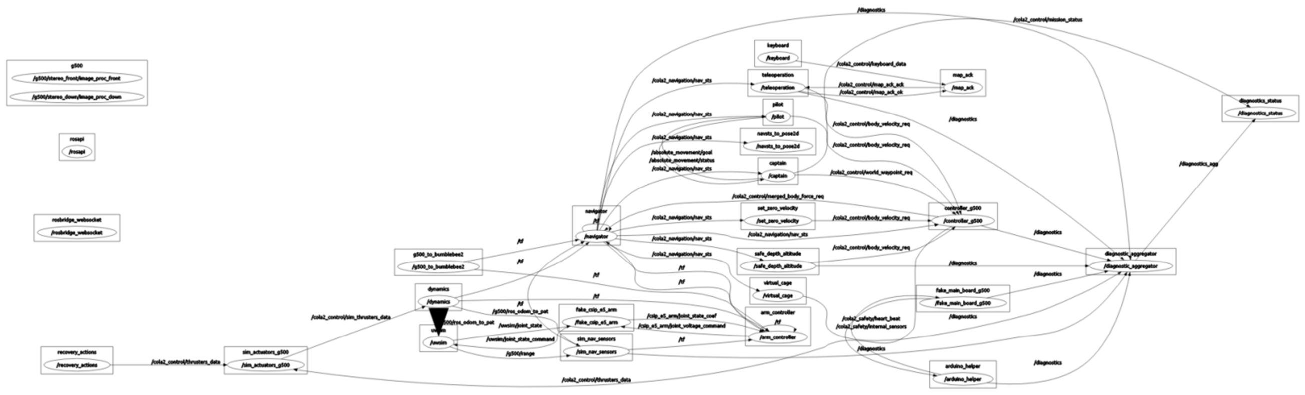


Figure 33: The RQT-graph after starting the simulator of Girona500 with the robotic arm

There are a lot of nodes communicating with each other, this makes the overview of the RQT-graph is complex. Therefore there will be more zoomed into the control of the vehicle and the robotic arm.

For controlling the vehicle and the robotic arm the RQT-graph has to be studied in detail. The control of the vehicle is now done by the keyboard or joystick. This means that the teleoperation nodes have to be found into the RQT-graph. If this is known it is possible to know where the commands of the Leap Motion device can be send.

4.2.3 Controlling the Girona500

For controlling the vehicle, messages have to be sent to the node "/controller_g500" because this is the node which communicates with the actuators (thrusts). As mentioned before, a node can listen to several topics. This node listens to 2 different topics:

- /cola2_control/body_velocity_req
- /cola2_control/world_waypoint_req

This is visible in the zoomed RQT-graph, figure 34. The blue lines are the topics where the node "/controller_g500" (red) listens to.

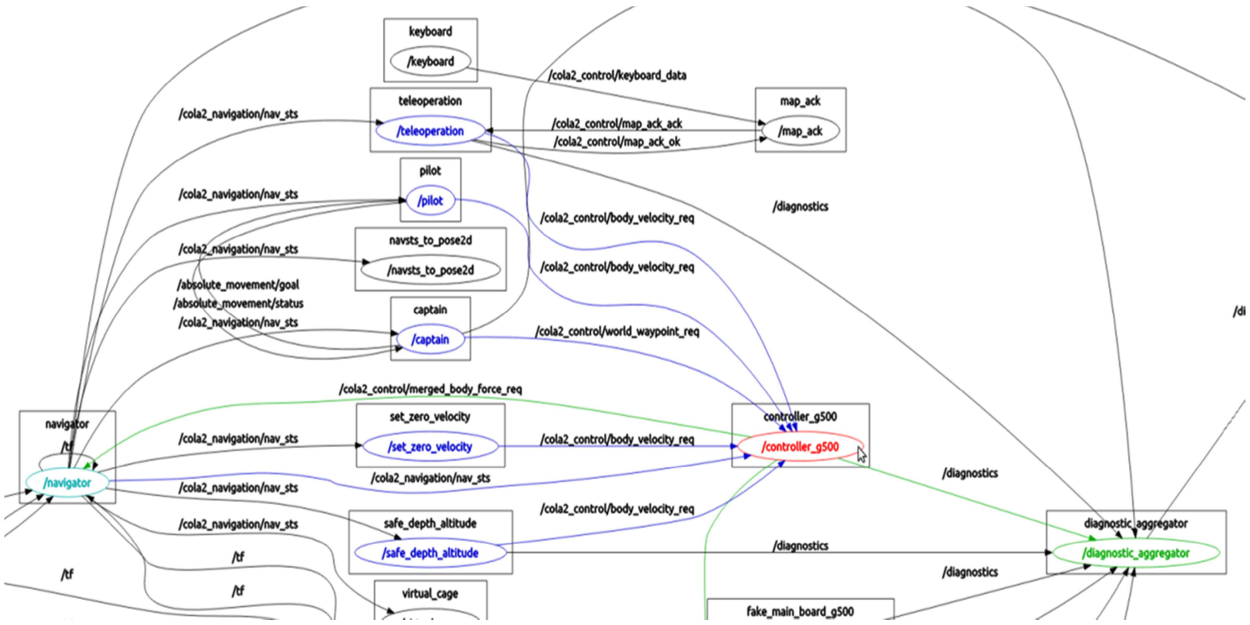


Figure 34: The zoomed-in RQT-graph after starting the Simulator of Girona500

This means that a message has to be sent to one of these topics. The node for teleoperating the robot through the Leap Motion will send commands to the topic “cola2_control/body_velocity_req” as shown in figure 35:

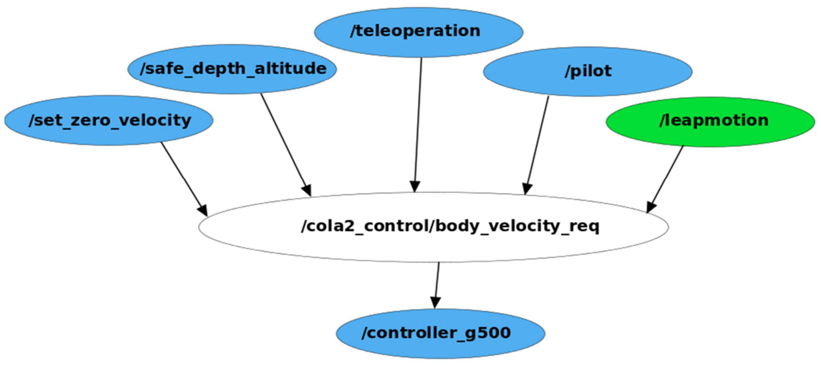
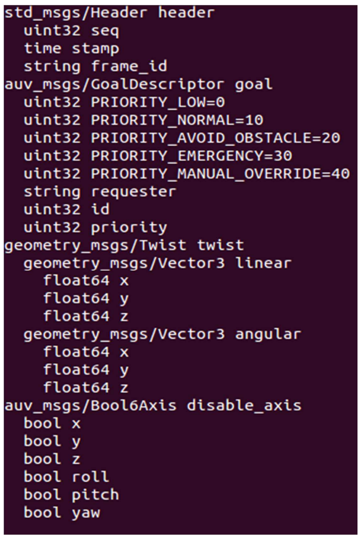


Figure 35: The implementation of the node 'leapmotion'

Because velocities have to be sent to the controller, the topic “/cola2_control/body_velocity_req” and the message-type BodyVelocityReq() will be used. This message-type BodyVelocityReq() consists out of 4 fields [46]:

- A Header standard type message. This message is used to communicate with timestamped data in a coordinate frame.
- An AUV message which describes the priority of this message. The subscriber will only listen to the message with the highest priority.
- A Geometry message which indicates the linear and angular speed.
- An AUV message which locks or unlocks the command to the topic for that degree of freedom [46].

This is shown in figure 36.



```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
auv_msgs/GoalDescriptor goal
  uint32 PRIORITY_LOW=0
  uint32 PRIORITY_NORMAL=10
  uint32 PRIORITY_AVOID_OBSTACLE=20
  uint32 PRIORITY_EMERGENCY=30
  uint32 PRIORITY_MANUAL_OVERRIDE=40
  string requester
  uint32 id
  uint32 priority
geometry_msgs/Twist twist
  geometry_msgs/Vector3 linear
    float64 x
    float64 y
    float64 z
  geometry_msgs/Vector3 angular
    float64 x
    float64 y
    float64 z
auv_msgs/Bool6Axis disable_axis
  bool x
  bool y
  bool z
  bool roll
  bool pitch
  bool yaw
```

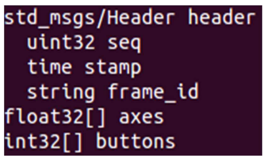
Figure 36: The message-type: BodyVelocityReq( )

4.2.4 Controlling the robotic arm ECA/CSIP Micro 4 DOF

For controlling the robotic arm, messages have to be sent to the node “/arm_controller”. This node will communicate with the joints for moving the end-effector of the robotic arm of the Girona500. For sending commands to the “/arm_controller” the message-type Joy() is used. A Joy()-message exists in the following fields: [47]

- A Header standard type message. This message is used to communicate with time stamped data in a coordinate frame.
- A float-message for the axes measurements from the Leap Motion device. These measurements represent the speed of the end-effector of the arm.
- An integer-message for the buttons measurements from the Leap Motion device [47].

This is shown in figure 37.



```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32[] axes
int32[] buttons
```

Figure 37: The message-type: Joy( ).

This message will be sent to the topic: /cola2_control/joystick_arm_ef_vel.

5 Controlling the Girona500 and the Turtlebot

Once studied the equipment and the software they use, it is possible to create a node in the software architecture of the used robot (Girona500 or Turtlebot). This node makes possible to control the robot with the Leap Motion.

5.1 Controlling the Girona500 and the robotic arm

The node is written in Python. Python is a programming language which is executable in ROS. This node will sent messages to the Girona500 for controlling the vehicle and the robotic arm. For this 2 different messages have to be sent [48].

5.1.1 Imports

First of all, the libraries will be imported in the node. Libraries are needed because they give you functions which are not available in the standard Python library. The libraries which are imported are the following:

- Leap Motion import: this is the library which gives you access to the commands for getting information of the Leap Motion device.
- ROS-imports: these are the libraries which will communicate with ROS. Rospy is one of these libraries. This is a Python library for ROS and makes sure that Python programmers are able to interface quickly with ROS-topics [49].
- The message-types: these imports make it able to sent the required message-types on a topic.
- The cola2-imports: this library makes it able to communicate in the cola2-packages.
- An import for doing math.

5.1.2 Class LeapMotionListener

After the imports, the class LeapMotionListener is created. This class contains methods that will be called when a controller event occurs. The used function is “on_init”. This function will initialize the listener. This listener listens to the Leap Motion device. Thanks to the Leap Motion library the data of the device can be used in this node.

5.1.3 Function talker

The position and movement of the user’s hands will be detected by the Leap Motion device. In this function the data which is coming from the Leap Motion device will be translated in commands. These commands will be sent to the topics.

5.1.3.1 Initialization

At first the function will be initialized before sending the messages to the Girona500. In this initialization, the listener will be created in this function. The listener listens to the commands receiving from the controller. Therefore also the controller has to be created. When the controller has been created and the Leap Motion device is connected with the computer, the data of the user's hands tracked by the Leap Motion device will be received. The listener will listen to this data and will respond to important controller state changes. After the creation of the listener and the controller, the listener has to receive the events from the controller. Therefore the created listener must be added to the created controller [50].

When this has happened, it must be declared that this node “leapmotion” is publishing to the topic “/cola2_control/body_velocity_req” using the message “BodyVelocityReq”. This is for controlling the vehicle Girona500. The node is also publishing to the topic “/cola2_control/joystick_arm_ef_vel” using the message “Joy”. This is for controlling the robotic arm of the Girona500. Next rospy needs to know the name of the node, which is used for sending this messages on this topics. Rospy needs to know this information otherwise it cannot communicate with the ROS Master. After that the amount of messages per seconds will be defined. This is the frequency [34].

5.1.3.2 Getting information from the Leap Motion device

After the initialization, a while-loop is introduced. This while-loop keeps until rospy is shutted down. In the while-loop it occurs a new initialization. In this new initialization the variables of the control of the vehicle and the robotic arm will be initialized. The control of the Girona500 will be done by the left hand and the control of the robotic arm will be done by the right hand. Before the node starts, it does not know if any hand is detected, so the variables, which represent if the hands are visible or not, will be initialized as not detected. The value of these variables will be set to zero. Also the maximum velocities will be set. These velocities are:

- The forward velocity
- The sway velocity
- The turning velocity
- The arm velocity
- The roll velocity of the arm

This initialization happens every time at the beginning of the while-loop. This is for making sure that the variables always have a value. After the initialization in the while-loop, the hands which are detected by the Leap Motion device will be checked. Here the distance between the fingertip of the index-finger and the fingertip of the middle-finger will be calculated. Later on the meaning of this distance will be explained. To do the calculation of this distance, the position of the fingertips have to be known. Therefore in a loop, all the detected fingers of the hand will be checked. If the finger is an index- or middle-finger, the position of those fingers will be saved. When the positions of the index- and middle-finger are known, the x-, y- and z-position of the fingertips are extracted from the position and the distance between the fingertips can be calculated.

If the right hand is detected, the x-, y- and z-position and the roll-angle of the right hand will be saved. Also the distance between the fingertips (index- and middle-finger) will be saved in a specific variable for the right hand. When the left hand is detected, the x- and y-positions and the pitch- and roll-angles will be saved. Also here the distance between the fingertips will be remembered. Because the coordinate systems of the Girona500, the robotic arm and the Leap Motion device are different, there has to be a transformation. Therefore the coordinate system of the Leap Motion device has to be known. This coordinate-system can be seen in figure 38.

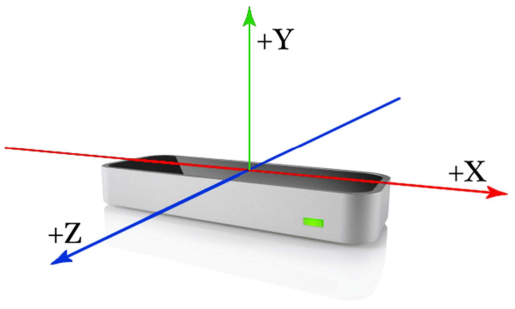


Figure 38: The coordinate-system of the Leap Motion [26].

5.1.3.3 Right hand detected

If the right hand is detected, there will be a right-hand-counter which increases every time the right hand is detected. After 1 second, the counter has the same value as the frequency. When this occurs, the information of the hand which was saved (x-, y- and z-position and the roll-angle of the right hand) will be the start-position and -orientation. If the hand remains all the time like this, the commands sent to the topic will be zero. Every movement of the right hand in the x-, y-, z-position or in the roll-angle will result in a movement of the robotic arm when the distance between the fingertips of the index- and middle-finger is small enough.

Because the coordinate-system of the robotic arm is not the same as the coordinate-system of the Leap Motion, the axis will be different.

- A movement in the z-direction of the Leap Motion coordinate-system will cause a forward/backward-movement of the robotic arm. A forward/backward-movement is a movement parallel to the x-axis of the robotic arm.
- A movement in the x-direction of the Leap Motion coordinate-system will cause a left/right-movement of the robotic arm. A left/right-movement is a movement parallel to the y-axis of the robotic arm.
- A movement in the y-direction of the Leap Motion coordinate-system will cause an up/down-movement of the robotic arm. A up/down-movement is a movement parallel to the z-axis of the robotic arm.
- A movement of the roll-angle will cause a roll-movement of the robotic arm.

When the movements are known, the message can be sent to the topic.

5.1.3.4 Left hand detected

If the left hand is detected, the left-hand-counter will increase every time the left hand is detected. Also after 1 second, the start-position and -orientation of the left hand (x- and y-positions and the pitch- and roll-angle) will be declared. When this is occurred and the fingertip-distance between the index- and middle-finger is small enough, every movement away from the start-position and -orientation will create a movement of the vehicle. Also here the coordinate-systems are not the same for both, so the axes will be different as well:

- A movement of the roll-angle will cause a turning of the vehicle in the horizontal plane.
- A movement of the pitch-angle will cause a forward/backward movement of the vehicle.
- A movement in the y-direction of the Leap Motion coordinate-system will cause an up/down movement of the robot. An up/down-movement is a movement parallel to the z-axis of the robotic arm.
- A movement in the x-direction of the Leap Motion coordinate-system will cause a sway-movement of the robot. A forward/backward-movement is a movement parallel to the y-axis of the robotic arm.

When the movements of the robot are known, the message will be sent to the topic. When a hand is not detected, the hand-counter will be zero again, so the reference-point has to be declared again. Also when the distance between the fingertips is too big, the messages will not be sent to the topic.

5.2 Controlling the Turtlebot

Now the Girona500 and his robotic arm can be controlled, there will be tested if it is possible to control other robots which have ROS as Operating System. The test will be done on the Turtlebot. For controlling this robot, the First Person technique will be used. The controlling will be the same as the one which is described in the paragraph 'other applications' in the literature review and is shown in figure 39.

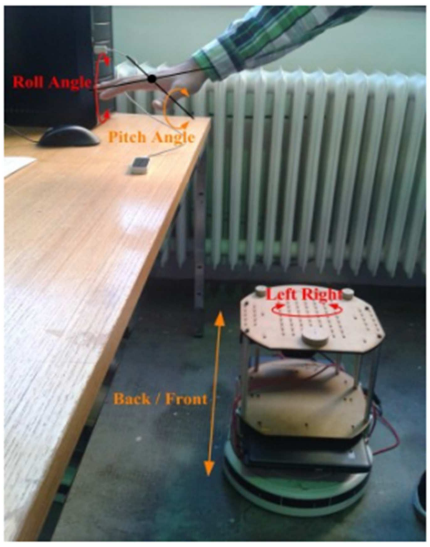


Figure 39: Roll/Pitch – Rotate/Move Translation Scheme [12]

The goal of this assignment is to test if it is possible to control the Turtlebot with the Leap Motion device. If this is possible, then it is also possible to control every robot which has ROS as Operating System. The Turtlebot does not work in the package Cola2, so therefore the implementation of the Turtlebot in ROS will be different than the implementation of the Girona500 in ROS.

5.2.1 The Turtlebot in ROS

The Turtlebot has a package which is divided in different sub-packages. These sub-packages are making sure that there is a good communication between the hardware en the software of the Turtlebot. All those sub-packages are responsible for a specific task for control the vehicle and making the movements visible in the simulator. There is a launch-file for starting the Gazebo-simulator of the Turtlebot which is located in the sub-package “turtlebot_gazebo”. When the simulator is launched, a lot of nodes will be sending messages on different topics and they will communicate with each other. For seeing which node is publishing/subscribing on which topic, the RQT-graph can be called. This RQT-graph is shown in figure 40.

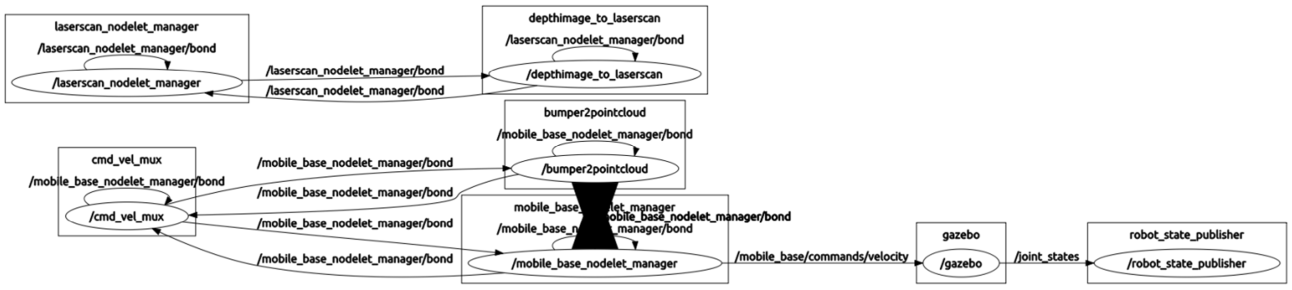
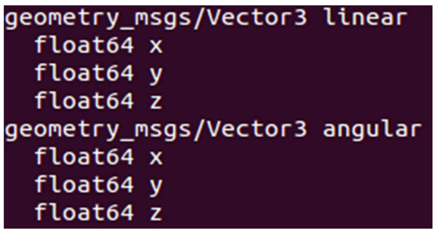

laserscan_nodelet_manager
/laserscan_nodelet_manager/bond
/laserscan_nodelet_manager
/laserscan_nodelet_manager/bond
/laserscan_nodelet_manager/bond
depthimage_to_laserscan
/laserscan_nodelet_manager/bond
/depthimage_to_laserscan
bumper2pointcloud
/mobile_base_nodelet_manager/bond
/bumper2pointcloud
cmd_vel_mux
/mobile_base_nodelet_manager/bond
/cmd_vel_mux
/mobile_base_nodelet_manager/bond
/mobile_base_nodelet_manager/bond
/mobile_base_nodelet_manager/bond
/mobile_base_nodelet_manager/bond
mobile_base_nodelet_manager
/mobile_base_nodelet_manager/bond
/mobile_base_nodelet_manager
/mobile_base/commands/velocity
gazebo
/gazebo
/joint_states
robot_state_publisher
/robot_state_publisher

Figure 40: The RQT-graph after starting the Gazebo-simulator

In this RQT-graph, the topic “/mobile_base/commands/velocity” will sent velocities to the Gazebo-simulator. Therefore a message must be sent on this topic for controlling the Turtlebot with the Leap Motion device. The message-type will be “geometry_msgs/Twist”, shown in figure 41. This message-type will sent velocities on the desired topic and consists of 2 fields:

- A linear geometry message
- An angular geometry message



```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Figure 41: The message-type: geometry_msgs/Twist

The both fields represent a vector in free space. These are the linear and angular velocities which have to be sent to the topic “/mobile_base/commands/velocity”.

5.2.2 The node 'leapmotion'

When all of this is known, the node can be created. The node is written in Python. Python is a programming language which is executable in ROS. This node will sent messages to the Gazebo-simulator for controlling the Turtlebot.

5.2.2.1 Imports

First of all, the libraries will be imported in the node. Libraries are needed because they give you functions which are not available in the standard Python library. The imported libraries are:

- Leap Motion import: this is the library which gives you access to the commands for getting information of the Leap Motion device.
- ROS-imports: these are the libraries which will communicate with ROS. Rospy is one of these libraries. This is a Python library for ROS and makes sure that Python programmers are able to interface quickly with ROS-topics [49].
- The message-types: these imports make it able to sent the required message-types on a topic.
- An import for doing math.

5.2.2.2 Class LeapMotionListener

After the imports, the class LeapMotionListener is created. This class contains methods that will be called when a controller event occurs. The used function is “on_init”. This function will initialize the listener. This listener listens to the Leap Motion device. Thanks to the Leap Motion library the data of the device can be used in this node.

5.2.2.3 Function talker

The position and movement of the hands will be detected by the Leap Motion device. In this function the data which is coming from the Leap Motion device will be translated in commands. These commands will be sent to the topics.

5.2.2.3.1 Initialization

At first the function will be initialized before sending the messages to the Turtlebot. In this initialization, the listener will be created in this function. The listener listens to the commands receiving from the controller. Therefore also the controller has to be created. When the controller has been created and the Leap Motion device is connected with the computer, the data of the user’s hands tracked by the Leap Motion device will be received. The listener will listen to this data and will respond to important controller state changes. After the creation of the listener and the controller, the listener has to receive the events from the controller. Therefore the created listener must be added to the created controller [50].

When this has happened, it must be declared that this node “leapmotion” is publishing to the topic “/mobile_base/commands/velocity” using the geometry message “Twist”. Next rospy needs to know the name of the node, which is used for sending this messages on this topics. Without this information rospy cannot communicate with the ROS Master. At the end of the initialization the forward-velocity and the turning-velocity will be initialized to zero and the maximum-velocities will set to their desired values. The counter will be zero and the amount of messages per seconds will be defined. This is the frequency [34].

5.2.2.3.2 Getting information from the Leap Motion device

After the initialization a while-loop will start to run. In this while-loop the information of the Leap Motion device will be translated in values which are usable for controlling the Turtlebot. The while-loop keeps until rospy is shutted down. Then the communication with the ROS Master stops. In the while-loop a new initialization will occur.

The new initialization starts with the defining of the angles pitch and roll. These will be initialized as zero. Before the node starts, it does not know if any hand is detected, so both hands will be initialized as not detected. The corresponding variable will be set at zero. The same reason can be given why the distance between the finger is initialized as zero. The new initialization happens every time at the start of the while-loop. This is for making sure that the variables always have a value.

After this new initialization, the hands which are detected by the Leap Motion device will be checked. If it is a right hand, nothing will be calculated or saved. If it is a left hand then the distance between the fingertip of the index-finger and the fingertip of the middle-finger will be calculated. Later on the meaning of this distance will be explained. To do the calculation of this distance, the position of the fingertips have to be known. Therefore in a loop, all the detected fingers of the hand will be looked at. If the finger is a index- or middle-finger, the position of those fingers will be saved. When the positions of the index- and middle-finger are known, the x-, y- and z-position of the fingertips are extracted from the position. When the distance between the fingers is calculated, the variable hand_detected will be equal to 1 and the pitch and roll angle of the left hand will be saved.

If the left hand is detected, so the variable hand_detected is equal to 1, the counter will be increasing by 1. This happens a couple of times per seconds, depending on the variable hertz, which is the frequency. When 1 second is passed, so the counter has the same value as the value of hertz, the start-orientation of the hand will be declared. When the counter is bigger than the value of the value of hertz, the assignment of the values will occur. For assigning the values of the pitch and roll angle to the velocities of the Turtlebot, the coordinate-system of the Turtlebot has to be known. This coordinate-system is shown in figure 42.

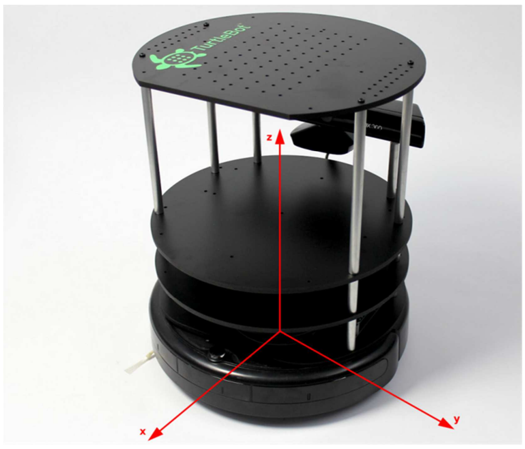


Figure 42: The coordinate-system of the Turtlebot [55].

Every movement of the left hand in the pitch- or roll-angle will result in a movement of the Turtlebot when the distance between the fingertips of the index- and middle-finger is small enough.

- A movement of the roll-angle will cause a turning of the vehicle in the horizontal plane. The turning in the horizontal plane corresponds to a turning around the z-axis.
- A movement of the pitch-angle will cause a forward/backward movement of the vehicle. The forward/backward movement corresponds to a speed in the direction of the x-axis.

This message will be created and be published to the topic "/mobile_base/commands/velocity".

5.2.3 Conclusion

After it was known that the Girona500 and the robotic arm attached to the Girona500 can be controlled by using the Leap Motion device, it is now become known that the Turtlebot also can be controlled with the Leap Motion device. The both robots work on an Operation system, called ROS, which is earlier explained. ROS is a modular architecture and therefore all the robots, which works on the Operation System ROS, can be controlled by a Leap Motion device. This is true for small and big robots. The result of the controlling of the Turtlebot in the Gazebo-simulator with the Leap Motion device can be seen in the following video.

- http://tiny.cc/ControllingTurtlebot

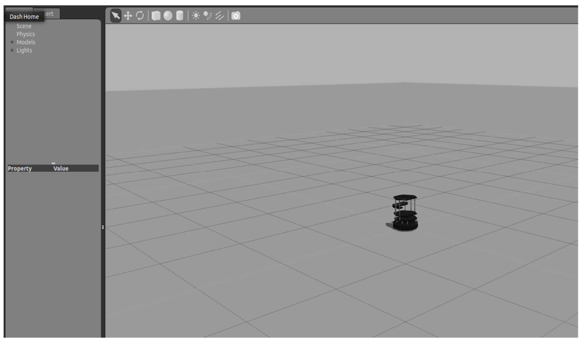


Figure 43: The Turtlebot in the Gazebo-simulator

6 Guide robotic arm to valve

The purpose of an intervention task for an underwater robot is to turn a valve in an underwater environment (for example in an oil-industry). The underwater robot drives into the water (or oil,…) and will be guided in the liquid. In this liquid an intervention panel with some valves is installed. The robot will detect the valves and then the second part of this master thesis starts. The purpose is to guide the robotic arm to the valve with the Leap Motion device. After the guidance to the valve, the valve should be turned for i.e. 90°. The turning of the valve is not a part of this master thesis. The valves on the intervention panel in the UWSim is shown in figure 44.

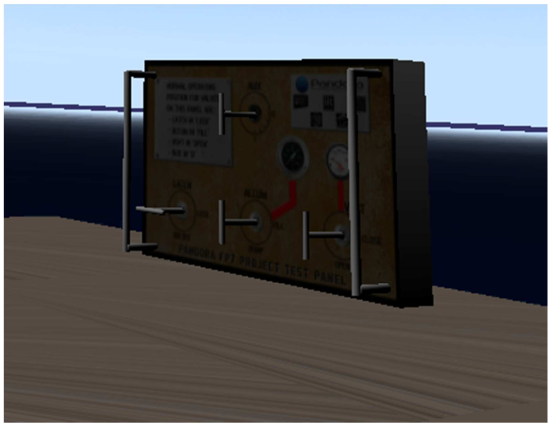
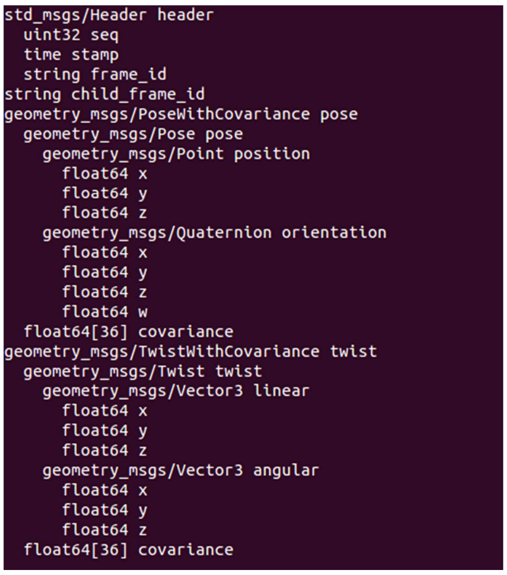


Figure 44: The Valve in the UWSim of the Girona500 Pandora.

6.1 Implementation in Cola2

First of all the node which will be created must be in the package of cola2, because the robotic arm of the Girona500 will be controlled. Because of this, the node must sent a message on the same topic as before, because the node "/arm_controller" listens to that topic. The name of this topic was "/cola2_control/joystick_arm_ef_vel". The message which will be sent on this topic is now a "nav_msgs/Odometry" message, shown in figure 45. This message-type is used because in this part of the master thesis the orientation and the position of the hand has to be sent and not the velocities. This message-type "nav_msgs/Odometry" exists of 4 different fields:

- A Header standard type message. This message is used for communicate with timestamped data in a coordinate frame.
- A String message which will give the id of the frame.
- A Geometry message which indicates the position and the orientation.
- A Geometry message which indicates the linear and angular speed.



```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

Figure 45: The message-type: nav_msgs/Odometry.

The pose represents the position and the orientation of the robot-hand, while the twist represents the linear and angular velocities. The purpose of this part of the thesis is to sent the orientation and the position of the hand to the node "/arm_controller". For doing that, there will be 2 nodes created. This is because the arm has to move first to the right position. Therefore the "hand_detector" node, this is a node which will be created, will detect the hand and sent position velocities (twist) and the hand orientation (pose) to the second node "hand_controller". The "hand_controller" convert the position velocities (twist) into the hand position (pose). The implementation of the 2 nodes is visible in figure 46.

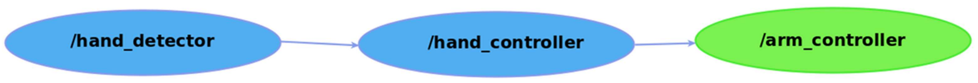


Figure 46: Implementation of the Intervention Task.

6.2 The node 'hand_detector'

As mentioned before, this node will detect the hand and sent the position velocity (twist) and the hand-orientation (pose) to the node “hand_controller”.

6.2.1 Imports

At the start of the node, the libraries have to be imported:

- Leap Motion import: this is the library which gives you access to the commands for getting information of the Leap Motion device.
- ROSPY-imports: This is a Python library for ROS and makes sure that Python programmers are able to interface quickly with ROS-topics [49].
- The message-types: these imports make it able to sent the required message-types on a topic.
- Transformation import: because the orientation is described in quaternion in the Odometry message. For transforming this into the Euler notation, this import is required.
- An import for doing math.

6.2.2 Class LeapMotionListener

This is the same class as the one which is used in the node 'leapmotion'.

6.2.3 Function talker

The purpose of this function is also the same as the one on the node for controlling the Girona500. The movements of the hand, which will be detected by the Leap Motion device, will be translated in useful commands. These commands will be sent to the required topics.

6.2.3.1 Initialization

At first the function has to be initialized before the commands can be sent. The first part of this initialization is the same as the one in the node for controlling the Girona500 and the Turtlebot. The listener and controller will be created and the listener will be added to the controller.

When this part of the initialization is done, it must be declared that this node “hand_detecter” is publishing to the topic “odometry” using the message “nav_msgs/Odometry”. This topic is used for sending the velocities (twist) and the orientation (pose) to the next node “hand_controller”. Next rospy needs to know the name of the node, which is used for sending this messages to the topic “chatter”. The node will take the name “hand_detecter”. Rospy needs to know this information otherwise it cannot communicate with the ROS Master. After that the amount of messages per seconds will be defined. This is the frequency [35], [50].

After this the maximum speed for the arm will be set and an orientation_counter will be initialized to zero. This orientation_counter will be explained later on. Then 3 other parameters (add_yaw, add_pitch and add_roll) will be initialized. These parameters represent the last orientation of the hand, before the user's hand leaves the visible area of the Leap Motion device.

6.2.3.2 Getting information of the Leap Motion device

When the rospy is not shut down, the code keeps continuing. This is a while loop which comes after the initialization. The previous initialization is not included in this while-loop. In the while-loop it occurs a new initialization. This is for making sure that the values always have a value.

In this new initialization some variables are initialized. The speed in every degree of freedom and the orientation in every degree of freedom in the Euler notation will be set to zero. The scale-factor of the position and the orientation also get a value. After that the variable hand_detected will be zero, because the node does not yet know if there is a hand visible or not. If this is 1, it means that the left hand is visible for the Leap Motion device. The variable for the distance between the fingertips will also be initialized as zero.

After the initialization in the while-loop, the data of the hands which are detected by the Leap Motion device will be received. From this data the position of the fingertips of the index- and middle-finger are used. With the positions of these fingertips, the distance between these positions is calculated. If the hand, which is detected, is a right hand, the parameter hand_detected will be changed into 1. The position velocities (twist) will be the same values as the position of the hand according to the Leap Motion device. Knowing that the Leap Motion device has a coordinate system like figure 47 and the hand is in a coordinate system which is described as:

- The velocity in the x-direction is according to the direction from the palm of the hand to the middle-finger.
- The velocity in the y-direction is according to the direction from the palm of the hand to the thumb.
- The velocity in the z-direction is pointed from the palm of the hand to the ground.

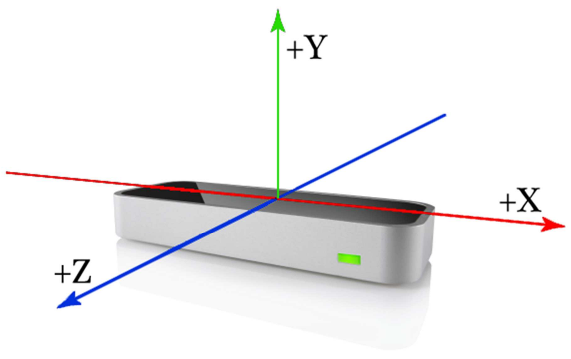


Figure 47: The coordinate-system of the Leap Motion [26].

The x-, y- and the z-axis of the coordinate-system of the hand correspond respectively to the y-, z- and x-axis of the hand coordinate system. The roll-, pitch- and yaw-angles stay the same. When the distance between the fingertips is bigger than 6 centimeters the orientation_counter will be increased. If this counter is equal to 1, the add_yaw, the add_roll and the add_pitch are the current values of this angles of the hand added by the previous add_yaw, add_roll and add_pitch. If the distance between the fingertips is smaller than this 6 centimeters, the orientation_counter is back 0. The meaning of this is that when the user opens the fingers, the orientation will be saved and if the user returns the hand to the frame, the orientation which is saved is the reference-orientation.

When the hand is detected, another counter will be increased. If this counter has the same value as the parameter “hertz”, the hand is 1 second visible for the Leap Motion device. At this moment the reference-point and -orientation are fixed. After this second, the values of the velocities can be changed by moving the hand above the Leap Motion device:

- Moving the hand forward/backwards, the node “hand_controller” will receive a forward/ backward velocity.
- Moving the hand right/left, the node “hand_controller” will receive a right/left velocity.
- Moving the hand up/down, the node “hand_controller” will receive a up/down velocity.

For the orientation, the orientation of the hand will accord to the orientation which will be sent to the node “hand_controller” with the addition of the add_yaw, add_pitch and add_roll.

After this the message can be created. The velocities can be sent directly to the node “hand_controller” and the orientation has to be transformed from Euler-notation to quaternion-notation. After this the message can be published to the topic “odometry”.

6.3 The node 'hand_controller'

This node will receive the message which is sent by the node “hand_detector” to the topic “odometry”. The velocities which are received have to be transformed into positions. After that a message with the position and the orientation of the robotic hand has to be sent to the topic “simulated_odom”.

6.3.1 Imports

First of all, the libraries will be imported in the node. The libraries which are imported are:

- Rospy-imports: This is a Python library for ROS and makes sure that Python programmers are able to interface quickly with the ROS-topics [49].
- The message-types: these imports make it able to sent the required message-types on a topic.

6.3.2 Global Variables

There are some global variables initialized. This is because there is a callback function and for making sure that these values, which are found in this function, are available in other functions. These global variables are:

- x, y, z: the velocities of the robotic hand.
- Quaternion: the orientation of the robotic hand in quaternion-notation.
- The time: these value is not initialized.

6.3.3 Function callback

In this function the node will listen to the data which is sent to the topic “odometry”. The message, which contains the values, which are received from the topic, will be saved in the global variables. Only the time is in nanoseconds, so this value has to divided by 1 billion so the time will be in seconds.

6.3.4 Function handController

Now the data is saved in the global variables, the velocity values of the hand can be converted to the position of the hand. The function gives rospy the name of the node “handcontroller”, which is used for sending messages to topics. Without this information rospy cannot communicate with the ROS Master. After that the subscriber and the publisher are declared. The subscriber will listen to the topic “odometry” and will receive an Odometry-message. When there are new messages to the topic, the callback-function is called with the message as the first argument. The callback-function receives the data from the message. The publisher will sent messages to the topic “simulated_odom” of the message-type “Odometry”. After that the amount of messages per seconds will be defined. This is the frequency. Then the start-position of the hand (pose) will be declared, which is (0,0,0). Also the variable “time” will be initialized as zero and the variable “previous_time” will be equalized to this variable “time”.

After initializing the variables, there is a while-loop which keeps when the rospy is not shutted down. In this while-loop the difference between the variables previous_time and time is calculated. When this difference is bigger than zero and smaller than 0,8 seconds, the position of the robotic hand will be calculated. This will be done by adding the multiplication of the speed and the time-difference. This will be done by the 3 Degrees of Freedom (x, y, z).

$$position = position + speed * \Delta time$$

After that the previous_time will be the same as the time, because for the next message, the time will be increased. When this variable is updated, the message can be created. Now in this message only the pose will be filled in with the position and the orientation. When the required fields (frame_id, stamp and pose) are filled in, the Odometry message will be sent.

7 Results

In this section the tests carried out with the Girona500 and the Turtlebot will be described. The Girona500 was tested in two different trails sessions, because there was space for improvement after the first one.

7.1 Girona500

As explained in previous sections the code was first tested in simulation. This helped us to save time and resources as the developing time of any new module for a robot is not negligible and testing code under development in a real robot is both tedious and dangerous. During the simulations we realized that controlling the Girona500 with the Leap Motion was relatively easy. On the other hand, controlling the robotic arm was not so easy, due mainly to the fact that the arm has limits on the joint was reached, it was hard to figure out which command was required to unlock that position and move away from the limit. Once the results on simulation were successful we moved to try the control system with the real Girona500. The tests were performed in the water tank of the underwater research centre (CIRS) of the University of Girona. The controlling of the Girona500 with the Leap Motion device was executed in two sessions, because during the first test some issues appeared.

7.1.1 The first water tank test

In this first water tank test, there were 2 different tasks executed:

1. The first task was controlling the Girona500 and the robotic arm with the Leap Motion device in order to check if every command was easy to execute. After tuning the gains of the control system for the real robot, as the dynamics of the robot are different in simulation than in the real robot, the control of the Girona500 was successful although some issues emerged. We realized there were some commands which were not implemented in the code, namely the sway and the backwards commands. Also the movements of the robot using the Leap Motion were not accurate. When a command was given to the robot and the command stopped, the vehicle still moved according to the command for a second longer. This because the maximum speed was too big. The accuracy could become better reducing the speed of the vehicle, but then it takes longer for reaching the required position and orientation. On the other hand, controlling the robotic arm had the same problem as in the simulator. The limits were all the time reached and it was hard to move the arm without reaching the limit in any of the joints.

2. For the second task, the Girona500 was placed on a random place in the water tank. This place in the tank was set as the reference point for the navigation, this is, all the navigation values (x,y,z,yaw) were set to zero. Then, the Girona500 was moved was moved to another place of the tank using the joystick. The mission was to sent the vehicle back to the initial position with the Leap Motion device, so that all values of the navigation were back to zero. Due to the commands missing (swaying and backwards), and the fact that the Leap Motion has less accuracy than the joystick it was very hard to complete the task.

Other issues found during the first water tank test were the following ones:

- If the user's hand comes into the visible field of the Leap Motion device, the Girona500 or the robotic arm (depending which hand is in the visible field) started to move at the first moment that the hand was visible for the Leap Motion device. The cause of this was that there was a reference point in the space above the Leap Motion device. It has to be a coincidence that the hand was correctly place at the reference point. Any deviation relative to the reference point makes the vehicle or the robotic arm move. A possible solution would be that the reference point is set as the position of the hand at the first moment detected. As a result, the Girona500 or the robotic arm remain still unless there is a movement of the user's hand.

- Another issue was that if the user wishes to stop the control of the vehicle of the robotic arm, the user's hand had to get out of sight of the Leap Motion device. This would sent one last command to the robot because the hand is still visible when retreating movement is executed. A solution would be to provide a gesture that would stop sending the commands to the robot. Then the hand can easily be removed from the visible field of the Leap Motion device.

7.1.2 The second water tank test

Before this test, the missing commands (swaying- and backwards-commands) were implemented. All the other issues found were solved too. The problem for withdrawing the hand in the field of the Leap Motion device is solved by calculating the distance between the index- and middle-finger. If the distance between those fingers is bigger than a certain value, the transmission of the commands will stop so that the user's hand can be removed from the visible field of the Leap Motion device without sending commands.

During the second water tank test two different tasks were tested:

1. The first task was controlling the Girona500 and the robotic arm with the Leap Motion device and controlling if every command was easy to execute . Controlling the Girona500 was pretty smooth. With the missing commands of the first test, it was easier to go from one place to another in the pool. The maximum speeds were adapted because the accuracy was not optimal. Regarding the control of the robotic arm, it had the same problems as in the first test.

2. The task was to capture a rope hanging vertically in the water with the robotic arm, as shown in figure 48. To do this mission, the Girona500 was in a random place in the water tank. The first phase of the test was therefore to guide the Girona500 just in front of the rope. In the second phase, the robotic arm had to grab the rope. This was a very difficult task because the accuracy of the control of the vehicle with the Leap Motion device is not optimal and the robotic arm was very often at its limits. To guide the vehicle exactly in front of the rope was difficult and once the vehicle was facing the rope, the robotic arm had to grab it. When grabbing the rope, the robotic arm was putted in a position that when the vehicle arrived in front of the rope, the robotic arm only needed a command forwards. After trying it approximately 5 times, the robotic arm grabbed successfully the rope.

The videos of the tests of the Girona500 can be seen on the following links.

- http://tiny.cc/controllingGirona500
- http://tiny.cc/controllingArm

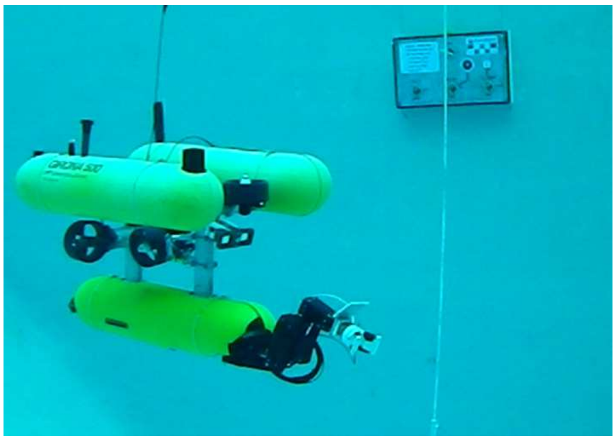


Figure 48: The Girona500 catching a Rope in the Water Tank

7.1.3 Catching the valve

The test for catching the valve is only executed with the simulator and not in the water tank due to time constraints. Another reason was that the tests in the simulator were difficult and not accurate. This means that it needs a lot of tuning. The simulator that was used for testing this task was the UWSim. For testing it in a real mission RVIZ (Robot Visualization). RVIZ can visualize ROS messages in realtime, allowing the user to observe what the robot is observing. It can create a virtual panel when the robot detects the real one.

7.2 Turtlebot

The tests on the Turtlebot are only executed in the Gazebo simulator, because the purpose of the controlling of the Turtlebot with the Leap Motion device was for demonstrate if it was possible to control other robots, which work under ROS and if the Turtlebot can be controlled in the simulator, the real Turtlebot also can be controlled. If the real Turtlebot can be controlled, then other robots which have ROS as Operating System can also be controlled. The results of the tests in the Gazebo simulator were positive. The turtlebot can be controlled with the Leap Motion device, so the conclusion is that other robots, which have ROS as Operating System, can be controlled with the Leap Motion device.

8 Conclusion

After doing a literature review of other projects working with hand gestures based HMI we decided to use the First Person technique as it seemed to be very intuitive and the easiest to use. For adapting this technique to underwater robots we needed little adjustments for the sway and turning commands because the turning was difficult to execute for the pilot. The Girona500 is easy to control with the Leap Motion device both in simulation and in real operation. Even though the controlling is not as accurate as when using a joystick.

Controlling the robotic arm was very hard because of the limits of the robotic arm. The smallest movement towards the limit stopped the controlling of the arm unless a reverse movement was commanded. Also for controlling the robotic arm a joystick is more accurate.

The Turtlebot could be controlled with the Leap Motion device. Since the Turtlebot works on ROS and ROS is a modular architecture, other robots which have ROS as Operating System can be controlled with the gestures using a Leap Motion device.

In general, we can conclude that the Leap Motion device is a more intuitive way for controlling an underwater robot. On the other hand, the joystick can be more useful when we need accurate movements.

Using hand gestures in order to define a target object and grasping approach has a big potential, but it has only been tested in simulation due to time constrains.

9 Appendices

9.1 The manual

In this section the different commands for controlling the vehicle and the robotic arm are explained.

9.1.1 Controlling the vehicle

For controlling the vehicle the left hand is used. At first the left hand has to be visible for the Leap Motion device. When the device detects the hand, the reference-point and the reference-orientation will be fixed after 1 second. Every deviation relative to this fixed point will result in a speed for the vehicle. If the left hand is perfectly back in this position and orientation the speed in every degree of freedom will be 0.

For controlling the vehicle make sure that the fingers of the hand are close to each other. If they are not, then the vehicle does not receive commands.

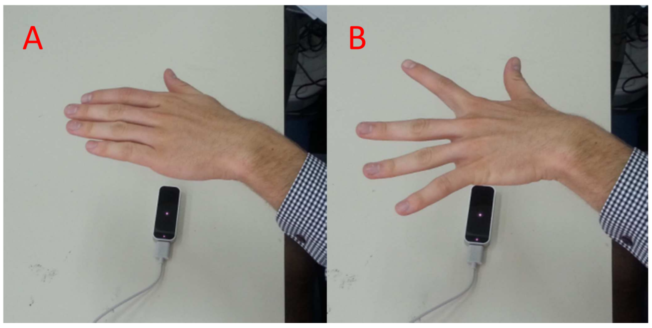


Figure 49: A: Reference point. B: Fingertips far from each other → no commands sent.

For controlling the vehicle the First Person technique is used. This is because it's an easy technique to learn and to use. Due that the turning to the right and turning to the left was hard, the technique is not completely the same as the First Person technique. The commands for controlling the vehicle are shown in figure 50.

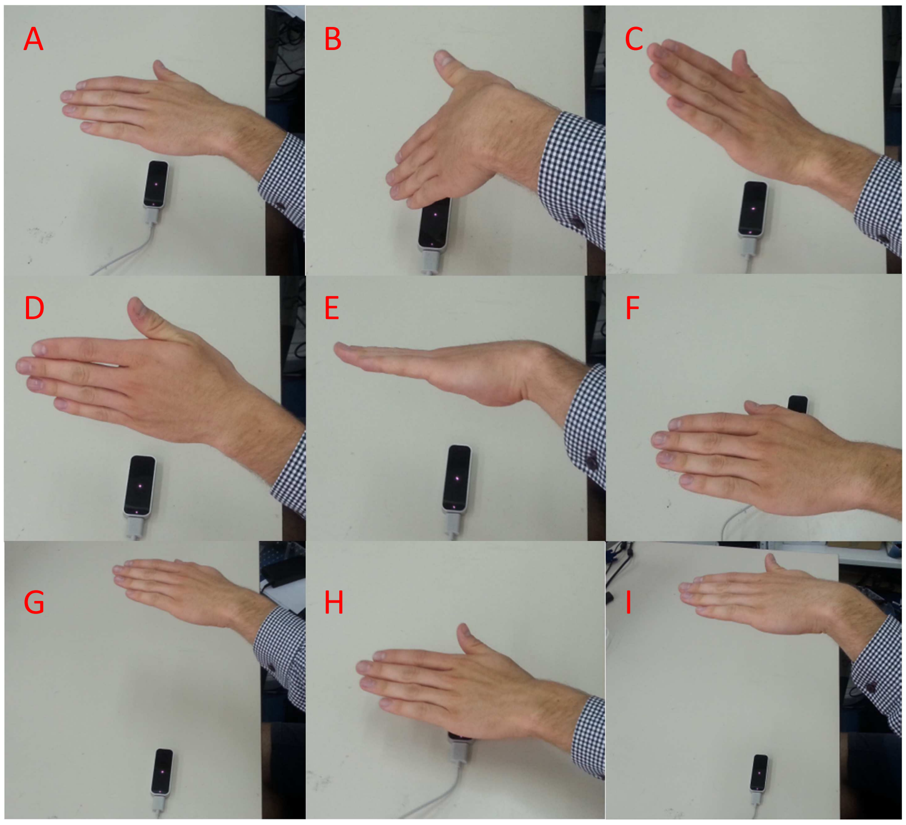


Figure 50: Controlling the vehicle commands. A: Base pose. B: Going Forward. C: Going Backward. D: Turn Left. E: Turn Right. F: Sway Left. G: Sway Right. H: Move Down. I: Move Up.

9.1.2 Controlling the robotic arm

For controlling the robotic arm the right hand has to be used. The fixed-point is the same as in the controlling of the vehicle and as well the space between the fingers for stop sending commands to the vehicle's arm.

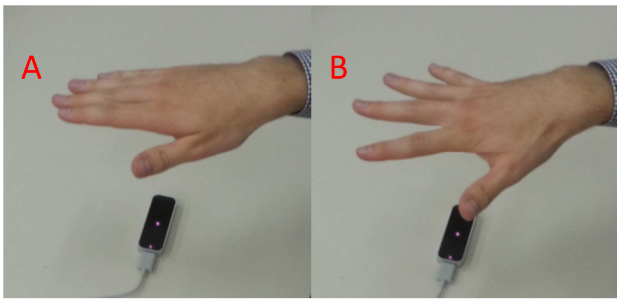


Figure 51: A: Reference point. B: Fingertips far from each other → no commands sent.

For controlling the robotic arm, the user has to pretend that his arm is the robotic arm. The commands are shown in figure 52.

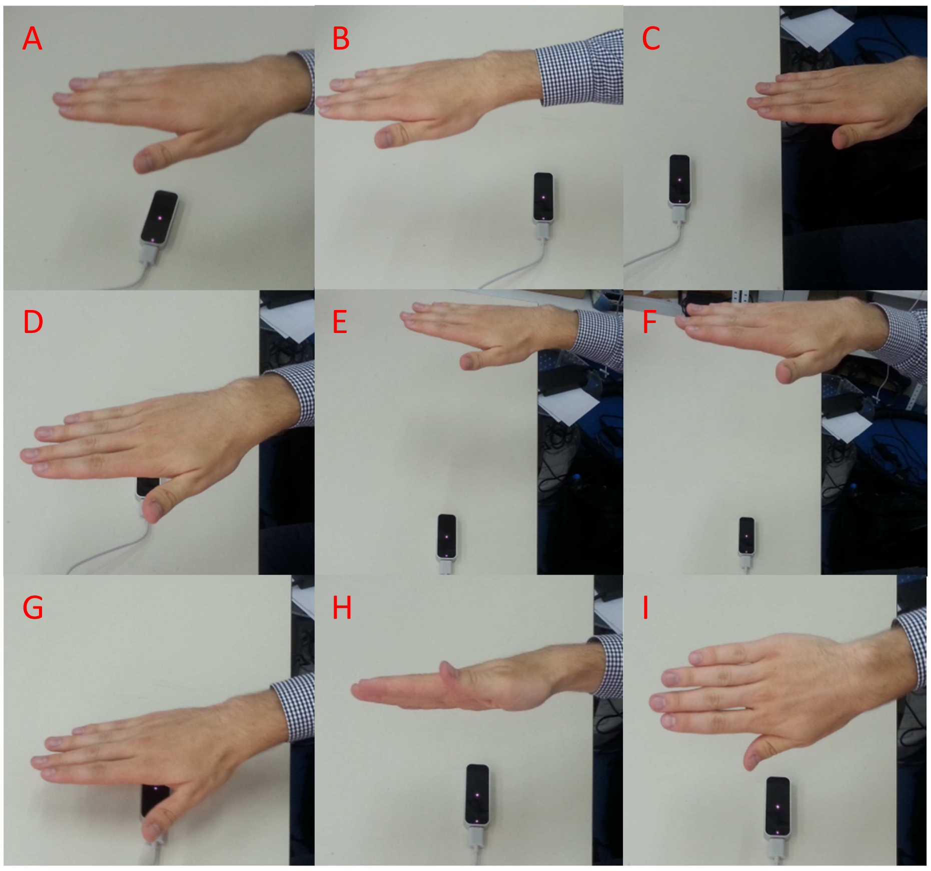


Figure 52: Controlling the robotic arm. A: Base Pose. B: Arm Forward. C: Arm Backwards. D: Arm to the Left. E: Arm to the Right. F: Arm Up. G: Arm Down. H: Roll-Movement to the Right. I: Roll-Movement to the Left.

References

[1] R. Hooper, "Learn About Robots," [Online]. Available: http://www.learnaboutrobots.com/undersea.htm. [Accessed 2015 June 1].

[2] J. Yuh, "Design and Control of Autonomous Underwater Robots: a Survey," *Autonomous Robots , no. 8, p. 18, 2000 .

[3] CIRS Girona, "Girona Underwater Vision and Robotics," CIRS Girona, [Online]. Available: http://cirs.udg.edu/auvs-technology/auvs/girona-500-auv/. [Accessed 22 5 2015].

[4] N. P. P. R. M. C. &. A. M. David Ribas, "Girona 500 AUV: From Survey to Intervention," *IEEE/ASME Transactions on Mechatronics,* vol. 1, no. 17, pp. 46-53, 2012.

[5] Leap Motion, Inc., "Leap Motion," Leap Motion, [Online]. Available: https://www.leapmotion.com/product. [Accessed 15 March 2015].

[6] S. L. K. J. J. L. J. Kevin P. Pfeil, "Exploring 3D Gesture Metaphors for Interaction with Unmanned Aerial Vehicles," in *International conference on Intelligent user interfaces*, Santa Monica, CA, USA, 2013.

[7] K. Pfeil, "An exploration of unmanned aerial vehicle direct manipulation through 3D spatial interaction," B.S. University of Central Florida, Florida, 2010.

[8] elephantseven, "PublicisPixelPark," PublicisPixelPark, [Online]. Available: http://www.publicispixelpark.de/cases/detail/89/. [Accessed 19 March 2015].

[9] N. B. D. Raju, "LinkedIn," LinkedIn, [Online]. Available: https://de.linkedin.com/pub/nagendra-babu-donthi-raju/14/a65/66. [Accessed 20 March 2015].

[10] N. B. D. Raju, "Youtube," Nagrenda Babu Donthi Raju , [Online]. Available: https://www.youtube.com/watch?v=sBzXRc9v7PU. [Accessed 19 March 2015].

[11] K. Culp, "Mercer Engeneering," Mercer University - School of Engineering, Macon, GA, 11 April 2014. [Online]. Available: https://libraries.mercer.edu/repository/bitstream/handle/10898/2953/xEGR%20102%20-%202014%20Expo%20Poster%20-%20LEAP%20Controller%20-%20K.Culp.pdf?sequence=1. [Accessed 20 March 2015].

[12] A. R. J. G. T. L. a. T. B. C. Georgoulas, "Home Environment Interaction via Service Robots and the Leap Motion Controller," in *The 31st International Symposium on Automation and Robotics in Construction and Mining* , Munich, Germany , 2014.

[13] A. Carrera, N. Palomeras, D. Ribas, P. Kormushev and M. Carreras, "An Intervention-AUV learns how to perform an underwater valve turning," in *Oceans 2014*, Taipei, Taiwan, 2014.

[14] M. C. D. R. P. J. S. G. O. C. Pere Ridao, "Intervention AUVs: The Next Challenge," in *International Federation of Automatic Control World Congress*, Cape Town, South Africa, 2014.

[15] J. P. J. J. F. ´. a. a. P. J. S. Mario Prats, "An Open Source Tool for Simulation and Supervision of Underwater Intervention Missions," in *Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, 2012.

[16] Turtlebot, "Turtlebot," [Online]. Available: http://www.turtlebot.com/build/. [Accessed 27 May 2015].

[17] Open Source Robotics Foundation, Inc., "TurtleBot," Willow Garage, [Online]. Available: http://www.turtlebot.com/. [Accessed 26 May 2015].

[18] Willow Garage, "Willow Garage," Willow Garage, [Online]. Available: https://www.willowgarage.com/turtlebot. [Accessed 26 May 2015].

[19] M. F. M. C. a. L. C. Claudio dos S. Fernandes, "A low-cost localization system based on Artificial Landmarks," in *Brazilian Robotics Symposium and Latin American Robotics Symposium*, São Carlos-SP, Brazil, 2012.

[20] Yujin Robot, "Yujin Robot," Yujin Robot, [Online]. Available: http://yujinrobot.com/eng/?portfolio=kobuki. [Accessed 27 May 2015].

[21] Yujin Robot, "Yujin Robot," Yujin Robot, [Online]. Available: http://www.robotnik.es/web/wp-content/uploads/2014/04/TB_robot.pdf. [Accessed 26 May 2015].

[22] B. Mottram, "I Heart Robotics," I Heart Robotics, 17 October 2011. [Online]. Available: http://www.iheartrobotics.com/2011/10/testing-turtlebot.html. [Accessed 27 May 2015].

[23] T. Carmody, "Wired," Wired, 11 March 2010. [Online]. Available: http://www.wired.com/2010/11/tonights-release-xbox-kinect-how-does-it-work/. [Accessed 27 May 2015].

[24] A. H. Nathan Koenig, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *International conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[25] Leap Motion, "Leap Motion," Leap Motion, [Online]. Available: https://developer.leapmotion.com/documentation/python/devguide/Sample_Tutorial.html. [Accessed 25 May 2015].

[26] Leap Motion, "leap motion," leap motion , [Online]. Available: https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Coordinate_Mapping.html. [Accessed 20 March 2015].

[27] J. A. L. C. Leigh Ellen Potter, "The Leap Motion controller: A view on sign language," in *25th Australian computer-human interaction conference*, Adelaide, Australia, 2013.

[28] ROS, "ROS," ROS, [Online]. Available: http://www.ros.org/about-ros/. [Accessed 19 May 2015].

[29] ROS, "ROS," ROS, 22 May 2014. [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed 19 May 2015].

[30] J. R. Chowdhury, "ROS: Robot Operating System," AGV KGP, Kharagpur, India, 2012.

[31] S. Generation, Artist, *Concept_de_base_de_ROS.* [Art]. Static Generation, 2014.

[32] K. N. John Kerr, "Robot Operating Systems: Bridging the Gap between Human and Robot," in *Southeastern Symposium on System Theory*, North Florida, Jacksonville, FL, 2012.

[33] M. C. K. G. B. P. F. J. F. T. L. J. W. R. a. N. A. Y. Quigley, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, Menlo Park, CA, 2009.

[34] Wiki ROS, "Wiki ROS," ROS, 4 March 2015. [Online]. Available: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29. [Accessed 25 May 2015].

[35] Nootrix, "Nootrix," Nootrix, [Online]. Available: http://nootrix.com/2012/04/ros/. [Accessed 22 May 2015].

[36] ROS, "ROS," ROS, 31 December 2013. [Online]. Available: http://wiki.ros.org/roscore. [Accessed 19 May 2015].

[37] ROS, "ROS," ROS, 14 March 2014. [Online]. Available: http://wiki.ros.org/Packages. [Accessed 19 May 2015].

[38] ROS, "ROS," ROS, 3 February 2012. [Online]. Available: http://wiki.ros.org/Nodes. [Accessed 19 May 2015].

[39] ROS, "ROS," ROS, 1 June 2014. [Online]. Available: http://wiki.ros.org/Topics. [Accessed 20 May 2015].

[40] ROS, "ROS," ROS, 15 July 2011. [Online]. Available: http://wiki.ros.org/rosservice. [Accessed 20 May 2015].

[41] ROS, "ROS," ROS, 22 May 2015. [Online]. Available: http://wiki.ros.org/Messages. [Accessed 20 May 2015].

[42] ROS, "ROS," ROS, 15 January 2015. [Online]. Available: http://wiki.ros.org/msg. [Accessed 20 May 2015].

[43] Bitbucked, "Bitbucked," Bitbucked, 28 May 2015. [Online]. Available: https://bitbucket.org/udg_cirs/cola2. [Accessed 22 May 2015].

[44] A. E.-F. M. C. a. P. R. Narcis Palomeras, "COLA2: A Control Architecture for AUVs," *IEEE Journal of Oceanic Engeneering,* vol. 4, no. 37, p. 22, 2012.

[45] Wiki ROS, "Wiki ROS," Wiki, 17 June 2014. [Online]. Available: http://wiki.ros.org/catkin/workspaces. [Accessed 22 May 2015].

[46] Docs ROS, "Docs ROS," ROS, 3 March 2015. [Online]. Available: http://docs.ros.org/hydro/api/auv_msgs/html/msg/BodyVelocityReq.html. [Accessed 25 May 2015].

[47] Docs ROS, "Docs ROS," ROS, 12 May 2015. [Online]. Available: http://docs.ros.org/indigo/api/sensor_msgs/html/msg/Joy.html. [Accessed 25 May 2015].

[48] Python, "Python," Python, [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed 26 May 2015].

[49] ROS, "ROS," ROS, 31 December 2013. [Online]. Available: http://wiki.ros.org/rospy. [Accessed 22 May 2015].

[50] Leap Motion, "Leap Motion," Leap Motion, [Online]. Available: https://developer.leapmotion.com/documentation/python/devguide/Sample_Tutorial.html. [Accessed 3 March 2015].

[51] J. Martin, Artist, *TurtleBot 2.* [Art]. CNET, 2013.

[52] *leapmotion.* [Art]. Derivate, 2014.

[53] Parrot, "Parrot AR Drone," Parrot, [Online]. Available: http://ardrone2.parrot.com/. [Accessed 20 March 2015].

[54] elephantseven, "Youtube," elephantseven, 7 August 2013. [Online]. Available: https://www.youtube.com/watch?v=PbowsxYw3IY. [Accessed 19 March 2015].

[55] D. Thomas, Composer, *Episode #7: Robot Operating System (ROS) and ROSPy.* [Sound Recording]. Talk Python To Me. 2015.

[56] ihrchive, "ihrchive," ihrchive, 14 September 2012. [Online]. Available: https://ihrchive.wordpress.com/2012/09/14/turtlebot-roomba-upgrade-kit/. [Accessed 25 May 2015].