UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Design Engineering

Visual Servoing and Grasping of Known Objects by a Mobile Manipulator

End of Degree Project

Bachelor's Degree in Industrial Electronics and Automation Engineering

AUTHOR: Kiyabala López, Alain

Tutor: Berjano Zanón, Enrique

External cotutor: URBANO DE ALMEIDA LIMA, PEDRO MANUEL

ACADEMIC YEAR: 2021/2022

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

Grado en Ingeniería Electrónica Industrial y Automática

Curso 2021/2022

TRABAJO DE FIN DE GRADO:

VISUAL SERVOING AND GRASPING OF KNOWN OBJECTS BY A MOBILE MANIPULATOR

DOCUMENTO 1. MEMORIA

AUTOR: Alain Kiyabala López
TUTOR: Dr. Enrique Berjano Zanón
COTUTOR EXTERNO: Dr. Pedro Manuel Urbano de Almeida Lima
(Instituto Superior Técnico, Universidade de Lisboa)

ACKNOWLEDGMENTS

I want to express how grateful I feel for having been part of this project.

First of all, thanks to Prof. Pedro Lima and the SocRob @Home team welcomed me from the beginning with their help and knowledge, even though I was an Erasmus student. This experience has helped me realise that I would like to focus my professional career on robotics and continue facing challenges like this.

I would also like to thank my UPV tutor Enrique Berjano for being so helpful from the very beginning. The communication with him has been excellent, and if it was not for that, I do not think I would have been able to hand it in on time.

Finally, thanks to my family and other loved ones for supporting me unconditionally all this time. Without a doubt, I owe everything I am to you. Thank you.

Resumen

El agarre y la manipulación de objetos son capacidades importantes para los robots de servicio doméstico. Se ha realizado una amplia labor de investigación en este ámbito que ha llevado al desarrollo de diferentes métodos, desde enfoques basados en la teoría de control clásica, hasta sistemas de aprendizaje automático totalmente integrales, aprovechando los avances en visión por ordenador, aprendizaje supervisado y de refuerzo.

Este proyecto propone una cadena de procesos para efectuar una tarea de pick and place. Para ello, se han planteado distintas aproximaciones aplicando visual-servoing como componente principal. La cámara de profundidad, fijada a la cabeza móvil del robot, localiza y sigue el objeto guardando su posición. Mientras tanto, el robot lee la posición de sus articulaciones y aplica una ley de control para reducir el error entre su efector final y el objeto detectado. Para detectar la posición del objeto, se da uso de una red neuronal convolucional entrenada en el laboratorio que fue fundamental en este proyecto.

La cadena de procesos que sigue es capaz de obtener la escena de octomap, ejecutar movimientos predefinidos para visualizar los objetos, activar la detección de objetos y efectuar el pick and place. Durante el proceso han aparecido diversos retos que han impedido el éxito de la tarea. En este proyecto, también se han usado las herramientas Gazebo, Rviz y Rqt para crear un entorno de simulación controlado y probar las distintas funcionalidades del robot. Además de haberse descrito problemas y soluciones abordadas en el campo de la robótica.

Palabras clave: Robótica, Manipulación, Visual-Servoing

Abstract

Object grasping and manipulation are important capabilities for domestic service robots. Extensive research work has been done in this area leading to the development of different methods, from approaches based on classical control theory, to fully end-to-end machine learning systems, leveraging advances in computer vision, supervised and reinforcement learning.

This project proposes a pipeline to perform a pick-and-place task. To this end, different approaches have been considered by applying visual-servoing as the main component. The depth camera, attached to the robot's moving head, locates and tracks the object and stores its position. Meanwhile, the robot reads the position of its joints and applies a control law to reduce the error between its fine effector and the detected object. The use of a convolutional neural network trained in the laboratory to detect the object's position was fundamental in this project.

The process chain that follows is able to obtain the octomap scene, execute predefined movements to visualise the objects, activate the object detection, and perform the pick and place. During the process, several challenges hindered the task's success. In this project, the tools Gazebo, Rviz and Rqt have also been used to create a controlled simulation environment and test the different functionalities of the robot. In addition, problems and solutions addressed in the field of robotics have been described.

Key words: Robotics, Manipulation, Visual-Servoing

Table of Contents

1.INTRODUCTION....1
1.1. JUSTIFICATION....1
1.2.1Context and motivation....1
2.Objective....5
3.State of the Art....5
3.1Robotic grasping and visual servoing....5
4. Methodology....9
4.1 Overall planning....10
4.2 Team and organization....12
5. Backround....15
5.1 ROS (Robot Operating System)....15
5.1.1 How to use it?....20
5.1.2 Programing language:....20
5.1.3 Basic ROS constructs:....21
5.1.4 ROS Filesystem Level:....22
5.1.5 Tools....23
5.2 Robot Kinematics....24
5.2.1 Position and Orientation....34
5.2.2 Forward kinematics....26
5.2.3 Inverse kinematics....27
5.2.4 Differential kinematics....27
5.3 Visual Servoing....29
5.4 Object detection....30
5.4.1 Neural Networks....30
5.4.2 Convolutional Neural Networks....34
5.4.2 Convolutional Neural Networks....34
5.4.3 Machine Learning....36
5.4.4 Deep Learning....36
6. Possible solutions....37
6.1Camera configuration....37
6.1.1 Eye-in-hand....37
6.1.2 Eye-to-hand....38
6. 2Visual Servoing Taxonomy....38
6.2.1 Image-Based Visual-Servoing....38
6.2.2 Position-Based Visual-Servoing....39
6.3 Object detection....41
6.3.1 YOLO Object Detection....41
6.3.2 DOPE (Deep Object Pose Estimation)....42
7. Proposed solution....44
7.1 General DescrIption....44
7.2 Pipeline....46

7.2.1 Preparation....56
7.2.2 Grasping....56
7.2.3 Post-grasp....56
7.2.4 Placing....57
7.2.5 Post-place....57
7.3 Implementation....57
7.3.1 Required packages....57
7.3.2 Pick and place implementation....58
7.4 Debugging....75
7.4.1 Printing messages in the terminal....76
7.4.2 Command lines....76
7.4.3 Rviz....76
8. Experiments and Results....79
8.1 Functionalities check....79
8.1.1 TIAGo tutorials....80
8.1.2 Test environment....82
8.1.3 Object detectio....82
8.2 Pick and place task....83
8.2.1 Preparation....83
8.2.2 Grasping....86
8.2.3 Post-gasp....88
8.2.4 Placing....88
8.2.5 Post-place....89
9. Environmental and social impact....89
9.1 Environmental impact....89
9.2 Social impact....90
10. Conclusion....91
11. Limitations....93
12. Future work....93
References....95
Appendix A: Repository Code....96
Appendix B: User Manual....97

1. INTRODUCTION

1.1. JUSTIFICATION

In Western countries, the population is ageing, which is due to various improvements in healthcare and lifestyle. In addition, young people have other priorities than having children and starting a family. Therefore, the issue of elderly care is in demand. The elderly population is increasing, but the number of caregivers is not, so the lack of people capable of caring for them creates the need for robot assistants to help them.

1.2.CONTEXT AND MOTIVATION

This TFG was conducted in the context of my academic staying in the Institute for Systems and Robotics which is a research centre affiliated to the Instituto Superior Técnico (IST) of the University of Lisbon. I was there during the spring semester I went as a student in the master of electronic engineering in the last semester in which I had to present a dissertation project. For this, I had to find a scientific advisor and a project to develop the thesis during the previous months. After contacting several research centres, Prof. Pedro Lima offered to participate with the socrob@home group in a robotics competition.

Prof. Pedro Lima is a full professor at IST with a PhD in Electrical Engineering from RPI, USA, and major research interests in formal approaches to several areas in multi-robot systems and their interaction with humans. He is the project coordinator and helped me in the control part, sharing different papers I can take as reference. He introduced me to Carlos Azevedo, a PhD student who is the team leader of the socrob@home group. We agreed that Carlos would help me to use the software, and Prof. Pedro Lima would help me with the theoretical part.

The idea of participating in a competition served as motivation to learn in the lab in a self-taught way. In this competition the robot has to be capable of performing various tasks related to the assistance of older adults. So that it is capable of monitoring the facility's condition, recognising people's condition, searching for objects, picking them up and transporting them to specific locations. First, it should be able to map a room in real-time to adapt to any changes in the layout of the user's house. Next, to recognise a series of objects previously given in a list, to pick up objects of different heights and drop them in a given location and to understand the voice command to find an object and find it. Each team member is responsible for a specific robot function and organising their time. There were no schedules, and most of the group worked entirely remotely. Each week we had different challenges to overcome and thus had general control of the evolution of the project. In addition to the project, we also received visits from educational centres and presented our robot's functionalities as it is shown in Figure 1 .

Figure 1: Visit from a educational centre to the ISR

In particular, I was involved in a project to prepare a robot to take care of household chores to participate in competitions and sell it to a nursing home after its improvement. My personal tasks were programming the manipulation part of the robot so it could perform a pick and place controlled by computer vision.

This project was intended to participate in the ERL CONSUMER LX 2022 (European Robotics League - Consumer Robots). This is a robot competition that is derived from its predecessor, the RoCKIn@Home competition, and focuses on tasks that service robots execute in a real home environment. The Consumer ERL is composed of several "Local Tournaments" held in different research laboratories across Europe, with certified testbeds. This competition took place in our laboratory, where a standardised scenario existed to test our robots [1] (See Fig. 2 & 3).

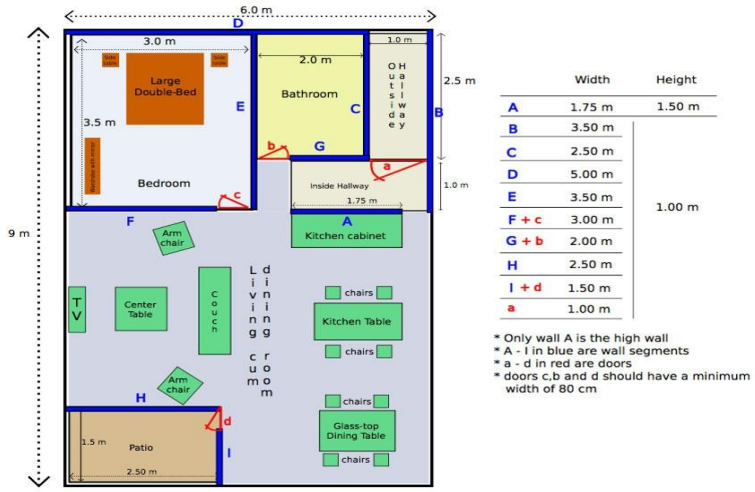

Figure 2: Mobile robotics lab, map of the scenario prepared for the competition

Figure 3: Mobile robotics lab, the real standarised scenario for the competition

2. OBJECTIVE

This project focuses on a technical solution for object grasping by a mobile robot endowed with a manipulator, with an assembled camera used to recognise and locate the object, i.e., a visual servoing problem. First, it should be able to localise the object using vision and move the manipulator to a suitable nearby location. Next, it uses vision to track the object localisation and move the manipulator to grasp it, performing visual servoing. After the object's picking, it moves the object to a relative pose and places it again on the table. This has to be tested in an apartment scenario using Gazebo simulator.

3. STATE OF THE ART

This section shows research work related to this thesis. It discusses research in the field of manipulator robots, its beginnings using vision and closed-loop visual control (Visual Servoing).

3.1. ROBOTIC GRASPING AND VISUAL SERVOING

The main tasks of using robots are grasping, manipulation and traUnsport. The use of robots starts in the automotive industry. The Unimate robot arm (See Figure 4) had a sequence of stored poses with manual control. The robot moved through the poses by interpolating its joint coordinates [2].

Figure 4 The Unimate was the first industrial robot ever built. It was a hydraulic manipulator arm that could perform repetitive tasks. It was used by car makers to automate metalworking and welding processes.

This type of point-to-point control is minimal, as it will not reach the object if it is slightly displaced from its position. Moreover, as the arm always executes the same trajectory, it is not prepared to avoid unexpected objects.

Vision-based systems were developed to solve this problem. McCarthy et al. describe one of the first [3], a camera is used to calculate the position of a cube, obtaining the target pose for the arm's end-effector. A space occupancy model is also obtained from the image data, making obstacle

avoidance possible. The limitation of this method is that it is an open-loop control. The camera obtains the object’s coordinates and sends them to the manipulator control. This is directly dependent on the accuracy and calibration quality of the camera and the manipulator [4].

A visual-feedback control loop increases the accuracy because the system can calculate the manipulator’s position and correct it to the desireds position. Shirai and Inoue developed the first visual-feedback system [5] that continuously recognises the posture of a cube in the manipulator’s hand, calculates the difference with the desired pose and moves the manipulator to compensate. The block can then be accurately placed inside a box.

Other projects test real-time closed-loop control of a robot end-effector, making the system robust to kinematic calibration errors. These systems were given the name visual-servoing.

This concept of visual servoing was firstly introduced in Hill and Park’s work [6], in which a camera is attached to a manipulator’s end-effector. The object’s position and orientation in the camera frame is obtained by processing the image. It uses the fixed transformation from camera to end-effector for describing the target in the end-effector’s reference frame. The system guides the end-effector to the target, considering the kinematic model of the robotic arm and its current joint positions.

These early visual-servoing systems were called position-based (PBVS): visual features are extracted from the image, and a geometric model of the target object is used to estimate its 3D pose. Features are often defined as the position, orientation and size of markers placed on the

target. A 3D pose can be calculated from at least 3 points, and the intrinsic calibration parameters of the camera must be known [7].

Another approach in PBVS systems is to use stereo-vision, as is described in Anderson [8] Allen et al. [9], Bukowski et al. [10], and Rizzi et al. [11]. In stereo vision, the images from two cameras are analysed to note their differences and calculate depth using disparity. An example of this approach is [12], where a PBVS system based on stereo vision techniques is capable of estimating the 3D coordinates of any point observed in two views of the same scene applying a triangulation process.

Later, the image-based visual-servoing (IBVS) technique was developed. Instead of doing 3D pose estimation, image features are directly fed into a control function that outputs the end-effector Cartesian velocity. For this, a feature-Jacobian matrix must be defined, relating the feature space change to the desired end-effector pose change. IBVS does not require computing the geometric model of the object, improving performance by making control more direct. However, choosing visual features and defining a feature-Jacobian that behaves well for multiple poses is challenging, especially as degrees-of-freedom increase [13].

There is also an approach called Direct Visual-Servoing (DVS), which considers the whole image as an input for the control system without the necessity of adding artificial image features. Collewet and Marchand propose photometric visual-servoing, an IBVS system that uses the luminance of the image’s pixels as the only essential feature [14]. An interaction matrix is defined as function of the desired image, which relates the observed image luminance to the end-effector velocity leading the arm to approach the desired pose.

4. METHODOLOGY

4.1. OVERALL PLANNING

The work team divided this project into different fields: navigation, perception and sensor fusion, decision making, user interaction, and manipulation.

Navigation involved most algorithms in robotics expose many parameters to configure, typically hand-tuned. Our group proposed a method to tune the parameters of robotics algorithms automatically. The use case is for the well-known Adaptive Monte Carlo Localization (AMCL) algorithm. As a result, we improved the localisation accuracy of our robot by automatically tuning the localisation parameters using several recorded training datasets.

Perception and Sensor Fusion involved our research in this domain include vision-based robot localisation, object tracking, simultaneous localisation and tracking (SLOT), environment modelling, laser-based robot localisation and vision-based simultaneous localisation and mapping (SLAM).

Particle filter-based (PF) methods have been the focus of our research to address most perception-related problems. Using PFs, the key issues that we have been engaged in solving include the fusion of noisy sensory information acquired by mobile robots where the robots themselves are uncertain about their poses and the scalability of such fusion algorithms

with reference to the number of robots in the team, as well as the number of objects being tracked.

For a domestic service robot working in a @Home-type environment, localisation, mapping, and object/person tracking constitute the basic requirements. In addition to this, static sensors along with mobile robots in an NRS, introduce further challenging issues for sensor-fusion algorithms. Considering these, we intend to drive forward our perception-related research in SocRob@Home actively.

About **Decision Making**, in prior work, we have addressed the problem of decision-making for teams of autonomous robots, primarily through approaches based on the theory of Discrete Event Systems (DES) and also through decision-theoretic formalisms for multiagent systems (Partially Observable Markov Decision Processes– POMDPs). Recently, we have bridged these two modelling approaches through developing and applying event-driven decision-theoretic frameworks. The fundamental insight of this line of research is that decision-making in physical environments is typically an asynchronous, event-driven process over several levels of abstraction, based on limited or uncertain sensorial information over each level, and subject to uncertain outcomes. We have explored this approach in the ongoing MultiAgent Surveillance Systems (MAIS+S) project (ref. CMU-PT/SIA/0023/2009), where we have successfully implemented an NRS for autonomous surveillance, comprising a team of mobile robots and a set of stationary cameras. The system can automatically detect relevant events in its operational environment, and the robot team can cooperatively decide on the appropriate response. In this context, we have also developed a suite of software tools to aid researchers in the systematic deployment of these abstract, decision-theoretic

methodologies on autonomous robots (the Markov Decision Making Library).

We seek to continue our work on this topic in SocRob@Home, noting that the ability to perform decision-making under uncertainty is a fundamental requirement of any potential domestic robot: given multiple tasks, such a robot must be able to manage its priorities; establish a plan for each of them, and still be able to react reliably to external events. Automated dialogue systems, which we plan to develop as part of our research effort in SocRob@Home, can also be interpreted as partially observable decision-making problems.

About **Human-Robot Interaction**, we focused on service robots in office environments, addressing symbiotic autonomy: robots execute tasks requested by users while autonomously aware of their limitations, asking humans to help the robot overcome them. More recently, we have been moving towards speech-based communication to address the requirement of natural human-robot interaction.

Finally, in terms of **Manipulation**, researchers in our team target the pick and place scenario from different sources like small and big tables and floors. We have a 7-degree of freedom (DoF) manipulator to accomplish those goals. Simultaneously, we also developed a visual servoing functionality, and we are developing a torque control interface for the gripper. Machine learning methods have been applied to object grasping to adapt easily to different targets and make systems more generally. From now on, we are going to focus on this last part that I have been working on.

4.2. TEAM AND ORGANIZATION

The work methodology was as follows: weekly online meetings in which we showed our progress to the team leader and suggested new tasks to present the following week. We have a discord server with different channels in which we consulted with the rest of the team for any doubt. There is a channel for each specific function of the robot to record and organise the information. Since each member works on a different topic, the way to evolve was mainly self-taught.

SocRob@Home is an enthusiastic team of SocRob, a long-term project at **the Institute for Systems and Robotics** from **Instituto Superior Técnico,** which focuses efforts on a group of robots to perform tasks, with a particular focus on participation in scientific competitions.

The SocRob team has been representing ISR/IST since 1998 in the world's leading scientific event on Artificial Intelligence and Robotics, RoboCup, as the application side of SocRob (Soccer Robots or Society of Robots) ISR/IST research project. Until 2013, the team's participation encompassed Simulation, 4-Legged, Middle Size and Robot Rescue Leagues in several editions of the RoboCup World Championship and various regional RoboCup events, e.g., the Portuguese, German and Dutch Opens. The project has involved more than 40 students over these 24 years, from early MSc years to PhD students. It has reached a maturity level that enables behaviour development that integrates low-level robot skills such as navigation, perception and manipulation into more complex behaviours that allow the completion of specific home tasks.

SocRob@Home addresses scientific problems that arise from the effort to deploy robots in the domestic environment to help humans. One of the goals of SocRob@Home is inculcating in young researchers the ability to work as part of an engineering team. This means solving engineering problems of diverse types (from hardware to software, including wireless communications, navigation, control, electronics, computer engineering, and software engineering), integrating contributions from modern information and communication technologies (e.g., networked robot systems require a mobile wireless network with robots, off-board computers, external sensors) and ensuring a background that opens doors for future bright multi-faceted engineers or engineering researchers.

The team covers many competencies, from Mechatronics integration to high-level decision making, including perception and task planning.

I briefly describe below the competence of each team member (in alphabetical order):

• Alain Kiyabala López: MSc student working on manipulation, specifically in visual-servoing.
• Ana Cruz: MSc student working on human-robot interaction, specifically in human intention prediction and recognition.
• Carlos Azevedo: PhD student at ISR, working on high-level planning and learning methods to tackle multi-robot coordination problems under uncertainty.

• Dmytro Kotenko: Technician at ISR, working on the design, construction and assembly of the electric, electronic and mechanical hardware.
• Emanuel Fernandes: MSc student working on Semantic Mapping and Automated Planning.
• Rui Bettencourt: PhD student at ISR, working on the cooperation of heterogeneous multi-robot systems as optimization problems.
• Pedro Lima: Faculty member who has been involved in robot competitions since the first editions of the RoboCup and ERL events. He is the project coordinator.

Until March 2022, the was the robot used by the SocRob@Home team for research (See Fig. 5). After that, the research moved to TIAGo, which is discussed below. The MONARCH project developed MBot to help hospitalised children by playing with them. After that, it was in home robot competitions, where it tested its functionalities in a realistic domestic environment. The team added to the robot a Kinova Gen2 6DoF robotic arm to make it able to perform the tasks during the competitions. After several years of use, the SocRob@Home team switched the robot to one with the factory-integrated arm, as it saved calibration problems.

Figure 5: MONARCH MBot robot with a Kinova Gen2 6DoF robotic arm..

TIAGo is the collaborative robot used by the team since the spring semester for research purposes (see Fig. 5). It is developed by the company PAL robotics to assist in household chores for the elderly. Tiago has a front laser range finder (LRF) and rear sonars for mapping, navigation and obstacle avoidance. It consists of a 7 degrees of freedom (DoF) arm with a parallel gripper as an end-effector and an elevating torso, allowing the robot to grasp objects from the floor or a high shelf. In addition, it has a speaker and stereo helpful microphone for human-robot interaction and a head-mounted RGB-D camera that provides precision and accuracy for object detection and localization, human tracking, obstacle perception and visual servo-orientation. The software architecture is based on ROS for the middleware while using off-the-shelf components wherever possible, allowing the team to focus on our research interests.

Figure 6: PAL Robotics TIAGo Steel edition robot (parallel gripper)

5. BACKROUND

5.1. ROS (ROBOT OPERATING SYSTEM)

ROS is an open-source robotics middleware that lets us build robots and reuse software between robotics applications using software libraries and tools. A middleware handles the communication between programs in a distributed system. Although ROS is not an operating System (OS), it provides services designed for a heterogeneous computer cluster, such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management.

The ROS goal is to provide a standard for robotics software's helpful development on any robot. ROS accelerates the prototyping of robot software since it is not needed to write everything from scratch. This standard allows us to focus on our application's essential features, using an existing foundation, instead of trying to do everything ourselves.

ROS is divided into four main components:

The '**Plumbing**' or communication middleware is in charge of process management and allows programs to communicate with each other. This part builds a network of programs/nodes sending and receiving different types of data to communicate with each other. Down below, there is an example of a ROS Computation graph in Figure 7.

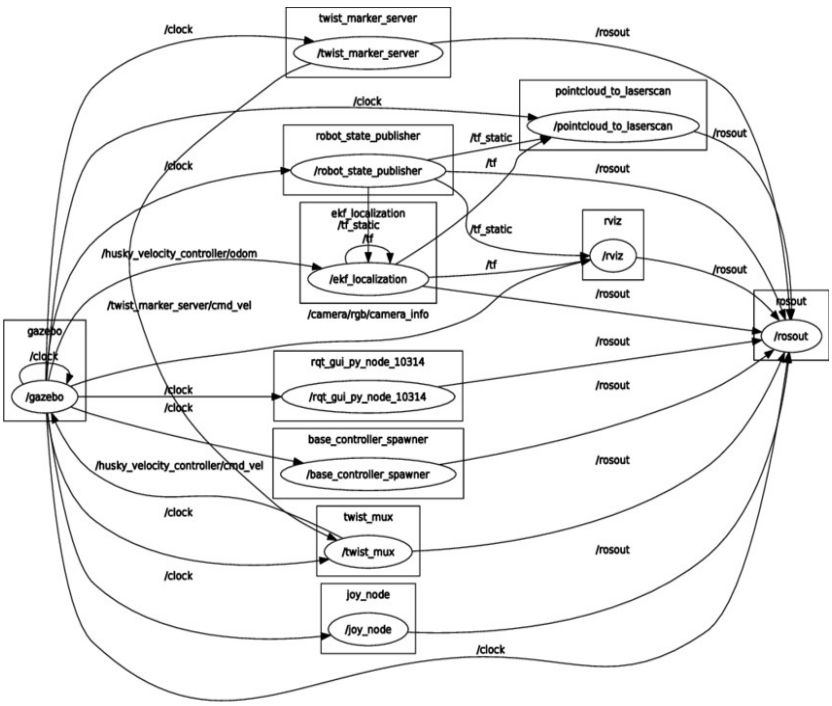

Figure 7: Example of a ROS computation graph[15].

While working on a robot, it is common to write many programs to handle sensor data and control the robot's actuators. With ROS Plumbing is easier to create a computation graph to manage the robot software.

Some **tools** help debug and monitor different kinds of data sent and received by ROS nodes. Some of them are useful, such as Rviz, Gazebo and Rqt, which will be explained later.

Many robot software blocks are built for various robotic **capabilities** such as navigation, perception, manipulation, and human-robot interaction. We can reuse ROS software and save time implementing these capabilities in their robot.

The **ecosystem** is the last component. The ROS open-source robotics framework is powered by thousands of developers worldwide who maintain and contribute thousands of ROS packages, tutorials, Q&A, and helpful material. A vibrant community of developers and users around the globe makes ROS a unique framework in Robotics.

5.1.1 How To Use It?

To see the big-picture of the system to understand what is going on.

ROS is a loosely coupled system where a process is called a node, and every node should be responsible for one task. Nodes use messages passing via logical channels called topics to communicate with each other. Each node can send or receive data from the other node using the publish/subscribe model. We are going deeper later.

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedra, Gentoo, Arch Linux and other Linux platforms. In our case, we run it in Ubuntu 18.04 LTS

While a port to Microsoft Windows for ROS is possible, it has not yet been fully explored.

5.1.2 Programing Language:

ROS is mainly developed using two languages: C++ and Python. Those are often the most used and preferred languages when developing robotics applications. It is required to install the roscpp library to write in C++ code and the rospy library to write in Python code. Our team use mainly Python to develop our tasks.

5.1.3 Basic ROS Constructs:

- ROS *master* is the brain of the whole communication, providing naming and registration services to the rest of the nodes in the ROS system, enabling communication between nodes. Every node registers at startup with the master otherwise, nodes would not be able to find each other, exchange messages, or invoke services.

- A ROS *node* is a single-purpose and executable program. It is a process that performs computation. Nodes provide modularity to robotic projects that use ROS. These nodes operate at a fine-grained scale; a robot control system usually comprises many nodes.

- The *topic* is a named bus over which nodes broadcast and receive information. Nodes can publish or subscribe to a topic to communicate with each other. These nodes are called publishers and subscribers.

- *Messages* are simply data structures comprising typed fields. These messages comprise a nested structure of integers, floats, booleans, strings and arrays of objects.

- *Service* is the request/response synchronous interaction with a node defined by a pair of message structures: one for the request and one for the reply. A service represents a node's action that will have a single result. A providing node offers a service under a name, and a client uses the service by sending the request message and awaiting the reply.

- *Actions* are more complex than services. Actions exist to provide us with an asynchronous client/server architecture, where the client can send a request that takes a long time. The actions use topics to send goal messages from a client to the server. After receiving a goal, the server processes it and can give information back to the client. This information includes the server's status, the state of the current goal, feedback on that goal during operation, and finally, a result message when the goal is complete. The client can asynchronously monitor the server's state and cancel the request anytime.

- *Bags* are helpful in ROS for storing ROS message data. There is a variety of tools to allow us to store, process, analyse, and visualise them.

5.1.4 ROS Filesystem Level:

The filesystem level concepts mainly cover ROS resources encountered on the disk, such as:

- **Packages**: Packages are the central unit for organising software in ROS. A package may contain ROS runtime processes (*nodes*), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organised together. Packages are the most atomic build item and release item in ROS, meaning that a package is the most granular thing able to be built and released.

- **Metapackages**: Metapackages are specialised Packages which only serve to represent a group of related other packages. Most commonly, metapackages are used as a backwards compatible place holder for converted rosbuild Stacks.

- **Package Manifests**: Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages

- **Repositories**: A collection of packages which share a common VCS system. Packages which share a VCS share the same version and can be released together using the catkin release automation tool bloom. Often these repositories will map to converted rosbuild Stacks. Repositories can also contain only one package.

- **Message (msg) types**: Message descriptions, stored in my_package/msg/MyMessageType.msg, define the data structures for messages sent in ROS.

- **Service (srv) types**: Service descriptions, stored in my_package/srv/MyServiceType.srv, define the request and response data structures for services in ROS.

5.1.5 Tools

There are different tools very helpful in the development process.

- Gazebo is an open-source 3D robotics simulator. It integrated the ODE physics engine, OpenGL rendering, and support code for sensor simulation and actuator control. It brings a complete toolbox of development libraries and cloud services to make simulation easy. Iterate fast on new physical designs in realistic environments with high-fidelity sensor streams. Test control strategies in safety, and take advantage of simulation in continuous integration tests.

- Rviz is a three-dimensional visualiser used to visualise robots, the environments they work in, and sensor data. It is a hugely configurable tool with many different types of visualisations and plugins.

- RQT is a software framework that implements the various GUI tools in the form of plugins. It can run all the existing GUI tools as dockable windows within rqt. Rqt makes it easy to manage all the various windows on the screen simultaneously.

Once the robot is known, a series of theoretical concepts must be understood for further development. For the manipulation of the robot, these concepts are the transform frame, kinematics (direct, inverse, differential), computer vision and deep learning.

5.2. ROBOT KINEMATICS

Robot kinematics is the field of robotics in charge of studying the relationship between a robot's joint coordinates and its spatial layout. This is useful for solving many problems, such as positioning the gripper where the object to grasp is and moving it from one point to another, avoiding obstacles and possible collisions. There is an overview of the basic concepts.

5.2.1 Position And Orientation

A description of the position and the orientation of the robot is defined by a reference frame. In an n-dimensional Euclidean space, the origin is given by a vector $\boldsymbol{p \in Rn}$, with $n = 3$ for the physical universe. Its elements px, py, and pz define distances along the axes of a corresponding reference frame such that $\vec{i}$, $\vec{j}$ and $\vec{k}$ are unit vectors corresponding to the axes of the Cartesian coordinate system.

$$p = p \cdot x \cdot \vec{i} + p \cdot y \cdot \vec{j} + p \cdot z \cdot \vec{k} = [px\ py\ pz]T \tag{5.1}$$

There are different frames in robotic systems, and knowing the coordinates of one point from one frame to another can be helpful. *The homogeneous transformation matrix* represents a mathematical relationship between two frames.

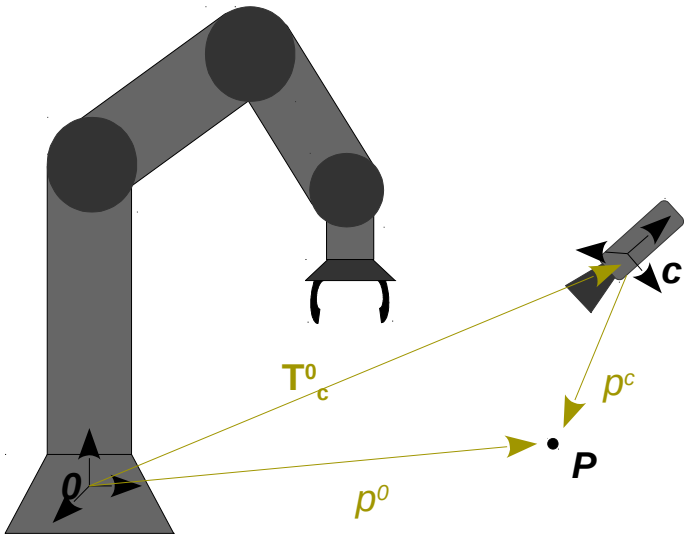

Figure 8: Representation of a point P in different coordinate frames

The figure 8 shows p^c as the Cartesian coordinates of a point *P* in the camera frame, p^0 as the coordinates of the *P* in the *0* frame and T^0_c as a homogeneous transformation matrix from *c* to *0*.

In order to calculate p_0:

$$p_c = T^0_c p^c \tag{5.2}$$

A homogeneous transformation matrix takes the form:

$$A_1^0 = \begin{bmatrix} R_1^0 & o_1^0 \\ o^T & 1 \end{bmatrix} \tag{5.3}$$

where $R^1{}_0$ is the rotation matrix of frame 1 with respect to frame 0 and $o^0{}_1$ is the translation vector from the origin of frame *0* to the origin of frame *1*. [16]

5.2.2 Forward Kinematics

Forward kinematics is in charge of calculating the frames of a robot's links, given the positions and values of all joints and the geometric link parameters as input. The main goal is to find the end-effector relative to the base as a function of the joint angles q. For a serial chain manipulator composed of n joints and n + 1 links, the end-effector's position relative to the base is obtained by concatenating homogeneous transformations between frames fixed in adjacent chain links.

$$T_n^0(q) = A_1^0(q_1) \cdot A_2^1(q_2) \cdot A_3^2(q_3) \cdot \ldots \cdot A^n - 1_n(q_n) \tag{5.4}$$

Where A_i^{i-1} is the homogeneous transformation matrix between two consecutive link frames, function of *i.* And q_i, the current angle of joint *i* connecting the links.

5.2.3 Inverse Kinematics

Inverse kinematics is the opposite process of forward kinematics. In this case, the problem is finding the joint positions' values given the end-effector frame relative to the base.

However, unlike forward kinematics, inverse kinematics cannot be solved in a closed-form expression (in general). We shall see there may be no solutions, multiple solutions, or even an infinite number of solutions to an IK problem, everything depending on the number of constraints and degrees of freedom. In this project, there is no sense in calculating them manually because there are many useful tools which provide approximate solutions for every type of manipulator[17]. Once the joint's angles are known, a motion profile can be generated using the Jacobian matrix to move the end-effector from the current to the goal position. This is explained in the following point.

5.2.4 Differential Kinematics

Differential kinematics gives the relationship between the joint velocities and the end-effector velocity. This can be done through the Jacobian matrix, which allows the calculation of the end-effector velocities given the joint velocities (direct differential kinematics) or to determine the joint velocities in order to move the end-effector with a prescribed speed (inverse differential kinematics). The mapping described depends on the current manipulator configuration.

$$v_e = \begin{bmatrix} \dot{\rho}_e \\ w_e \end{bmatrix} = J(q)\dot{q} \tag{5.5}$$

In this formula (5.5), v_e is the end-effector velocity, $\dot{q}$ is the vector of linear joint velocities, and $J(q)$ is the Jacobian matrix dependent on joint angles q.

Following direct kinematics, the Jacobian matrix can be partitioned into the (3 x 1) column vectors J_{Pi} and J_{Oi} as (5.6).

$$J=\begin{bmatrix} J_{P1} & \cdots & J_{Pn} \\ J_{O1} & \cdots & J_{On} \end{bmatrix} \tag{5.6}$$

Where J_{Pi} refers to the position and J_{Oi} refers to the orientation of the joints.

$$\begin{bmatrix} J_{Pi} \\ J_{Oi} \end{bmatrix} = \begin{cases} \begin{bmatrix} z_{(i-1)} \\ 0 \end{bmatrix} & for\ a\ prismatic\ joint \\ \begin{bmatrix} z_i - 1 \times \left(p_e - p_{(i-1)} \right) \\ z_{(i-1)} \end{bmatrix} & for\ a\ revolute\ joint \end{cases} \tag{5.7}$$

z_{i-1} is given by the rotation of z-axis unit vector, p_e and p_{i-1} are is provided by the position vector in the transformation matrices T^0_e and T^0_{i-1}.

However, inverting the Jacobian matrix can be possible to obtain a desired joint velocity dependent on desired position and orientation of the end-effector.

$$\dot{q} = J^{-1}(q) v_e \tag{5.8}$$

If J is not invertible. There is a method using the Moore-Penrose inverse (or pseudoinverse) $J^{\dagger}$. There are several ways to achieve it, but the most used computational method is using the SVD (singular value decomposition) (5.9):

$$J = UDV^T, J^{\dagger} = VD^{\dagger}U^T \tag{5.9}$$

- U, D and V are SVD(singular value decomposition) of J. Where U is an orthogonal m x m matrix, its columns are the left-singular vectors of J; V is an orthongonal n x n matrix, its columns are the right-singular vectors of J. And D is a diagona m x n matrix, element along its diagonal are the singular values of J.
- $D^{\dagger}$ = (D with reciprocals of all non-zero elements)^T

Using $J^{\dagger}$ instead of J^{-1} provides a computational method to obtain the joint velocities required for a desired end-effector velocity.

This can be used to solve inverse kinematics: instead of the desired end-effector velocity we use direct kinematics to compute (5.10)

$$\Delta p = p(q_0 + \Delta q) - p(q_0) \tag{5.10}$$

where the change in end-effector position is given by the current joint angle changes Δq. Simultaneously, Δq can be iteratively improved using the Newton-Raphson method, minimizing an error function that measures distance to the desired end-effector position.

$$error = \left\| p(q_0 + \Delta q) - p_{desired} \right\| [16] \tag{5.11}$$

5.3. VISUAL SERVOING

Visual Servoing(VS) refers to the control of the motion of the robot with feedback information extracted from a vision sensor. The idea is to keep a closed loop with the manipulator. VS is analogous to a PID controller. Therefore the image is translated into useful track progress metrics. Then, the robot moves gradually to reduce the error between the current and the desired position[4].

VS relies on different things such as good camera calibration, an accurate kinematic model of the robot, good IK and FK solvers, and a quality camera performance.

VS is a very large field of research, and many approaches have been developed. In the following paragraphs, some of them are pointed out based on their camera configurations and control architecture.

5.4. OBJECT DETECTION

Object detection is a computer vision technique which identifies and locates instances of objects within an image or vídeo. There are a diversity of techniques to perform object detection. Object detection algorithms leverage machine learning or deep learning to produce meaningful results. Moreover, some deep learning–based approaches use convolutional neural networks (CNNs), which automatically learn to detect objects within images.

5.4.1 Neural Networks

A *Neural network*, also known as an artificial neural network (ANN), is a method in artificial intelligence that teaches computers to process data in a way inspired by the human brain, mimicking how biological neurons signal to one another.

Every neuron acts as a node with its own linear regression model, where the input data act as a vector x is multiplied by a weight w, and a bias (or threshold) value b is added. The result is passed through an activation function σ, which determines the neuron's output y. Fig. 9 illustrates this process.

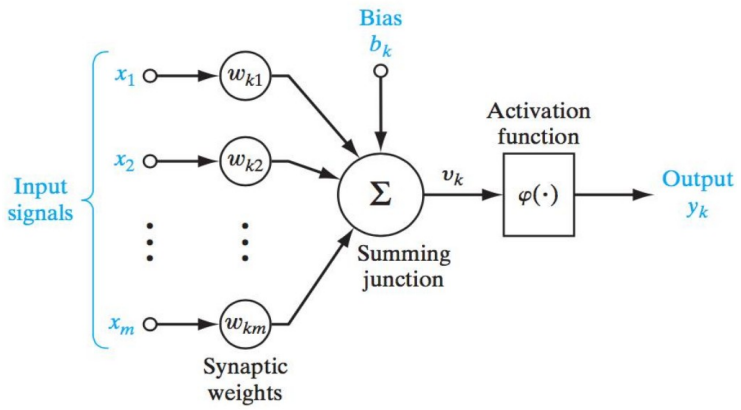

Figure 9: Architecture of a single neuron in a neural network [18]

There is a given dataset with several input/target pairs {x,t}. The ANN aims to learn a model of the relationship between x and t. To train the neural network is to obtain the weight vector w that produces a function y as close as possible to t.

Target function t does not use to be a linear function of x, so nonlinearity must be introduced in the system. This is how activation function σ acts. Several nonlinear functions can be used with different performance characteristics, the most common being the sigmoid, tanh and ReLU. See Fig. 10.

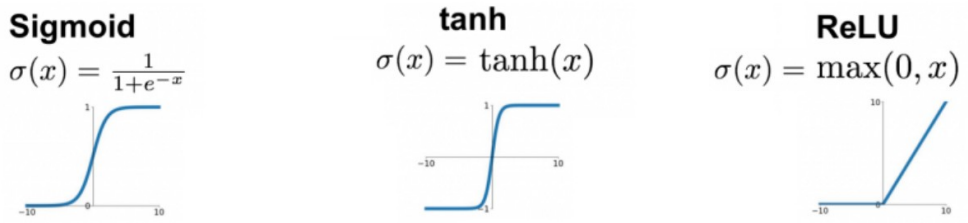

Figure 10: Popular activation functions

A machine learning process called deep learning uses interconnected nodes in a layered structure. It creates an adaptive system that computers use to learn from their mistakes and improve continuously. In this structure, there is an input layer, some hidden layers and an output layer. Each node is an artificial neuron that connects to another and has an associated weight and threshold. The output of a node is the input of the following one. See Figure. 11

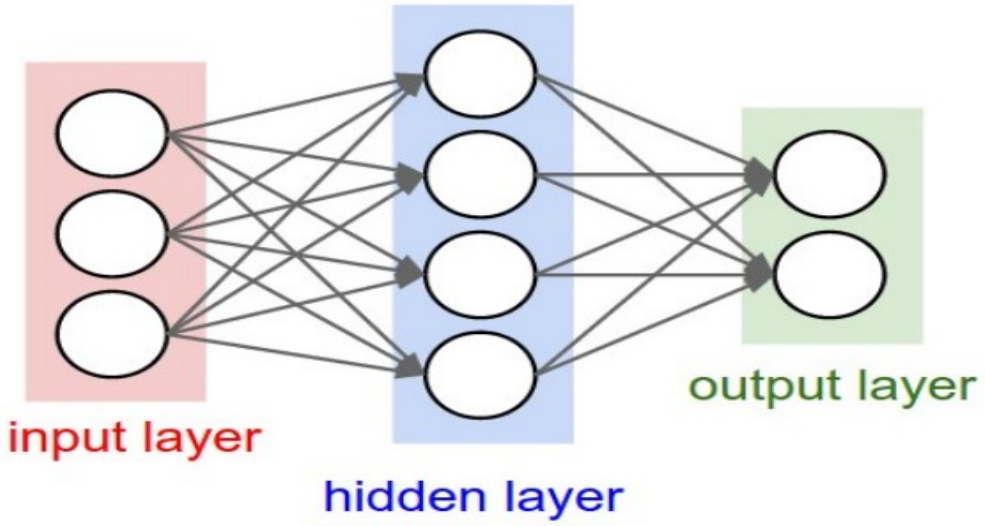

Figure 11: Example of a neural network with only one hidden layer. This network's layers are fullyconnected: all neurons connect to all neurons in the next layer [19]

Several network topologies have been developed for different purposes, e.g. Recurrent Neural Networks (mainly used for natural language processing) contain loops, allowing the network's output to be influenced by temporal sequences in the input. Convolutional Neural Networks (primarily used for computer vision tasks) introduce convolutional layers that apply image filters.

5.4.2 Convolutional Neural Networks

A convolutional neural network (CNN or ConvNet), is a type of network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction. ConvNet architectures make the explicit assumption that the inputs are images, which allows to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. CNNs provide a more scalable approach to image classification and object recognition tasks, leveraging principles

from linear algebra, specifically matrix multiplication, to identify patterns within an image. These convolutional layers create feature maps that record a region of image which is ultimately broken into rectangles and sent out for nonlinear processing. It consists of a set of learnable filters, that slide (convolve) through the image space and activate when they detect visual features such as edges or special shapes. The initial convolutional layer detects simple shapes, while deeper layers detect more complex patterns.

There are inserted pooling layers between consecutive convolutional layers. Their purpose is to downsample the image representation, reducing its spatial size and therefore the amount of parameters, making the network more efficient and less susceptible to noise. Pooling layers don’t add learnable parameters to the network, since they apply a fixed operation on the input. Fig. 12. shows a pooling layer operation.

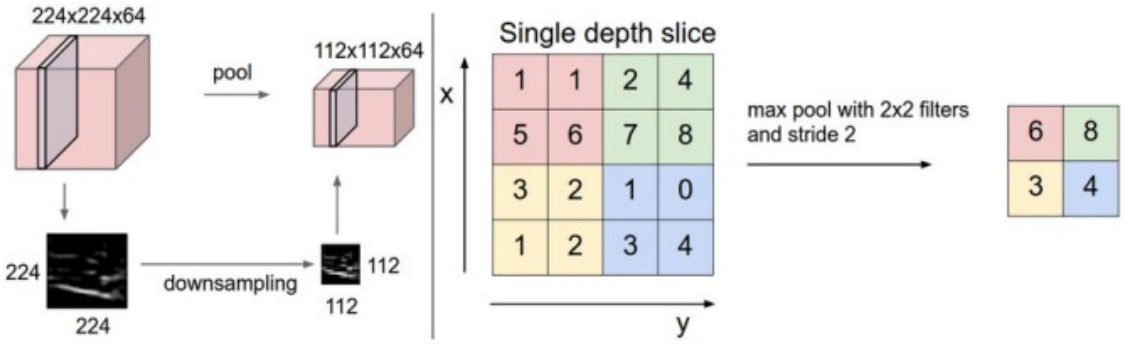

Figure 12: Pooling operation. Left shows the downsampling effect. Right shows how it’s achievedthrough max-pooling, the most common method [19]

It is a common practice to place fully connected layers at the end of the CNN after the highest level feature outputs. This is done so that these high-level features can be learned in nonlinear combinations. In the case of image classification, the size of the final output vector is equal to the number of detectable classes (see Fig. 13).

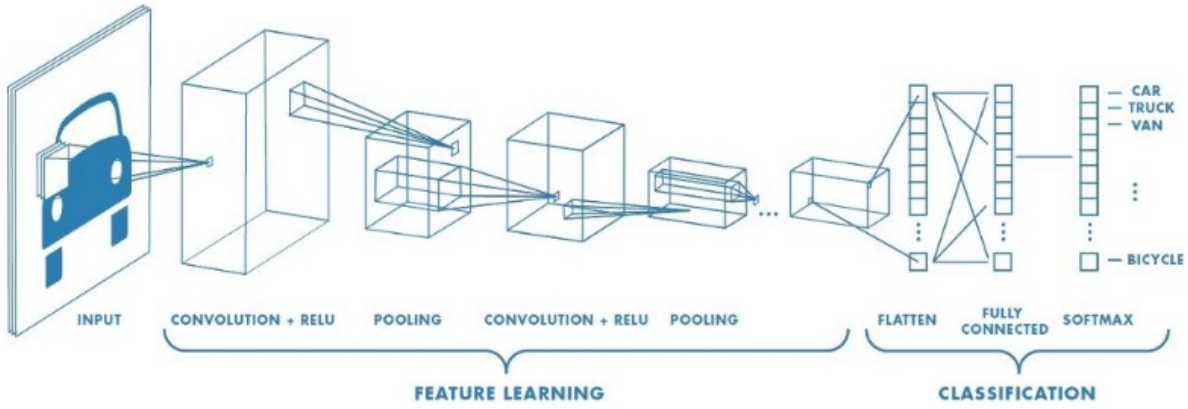

Figure 13: Full CNN architecture: convolutional layers interspersed with pooling layers produce acompact representation of high-level image features. The feature matrix is flattened, and finally one ormore fully connected layers are placed. [20]

5.4.3 Machine Learning

Machine learning (ML) is a subset, an application of Artificial Intelligence (AI) that offers the capacity to the system to learn and improve gradually from experience without being programmed to that level. Machine Learning uses data to train and find accurate results. It accesses the data and uses algorithms to learn from itself, imitating how humans learn.

5.4.4 Deep Learning

Deep Learning (DL) is a subset of Machine Learning in which the artificial neural network and the recurrent neural network are related. Deep learning and machine learning differ in the way each algorithm learns. Machine learning is more dependent on humans because human experts set the features to understand the differences between data inputs, requiring more structured data most of the time. On the other side, deep learning can use labelled datasets, also known as supervised learning, to inform the algorithm, but it is not needed. Deep learning can admit unstructured data in its raw format and classify the set of features from different categories of data from one to the other. The way of creating algorithms is the same as machine learning but with many more levels of algorithms. It solves all the complex problems with the support of algorithms and its process.

6. POSSIBLE SOLUTIONS

6.1. CAMERA CONFIGURATION

There are two common-used configurations for the camera and the joint effector. These are called eye-in-hand, where the camera is attached to the end-effector, and eye-to-hand, where the camera is fixed in the workspace, observing the target and the motion of the end-effector.

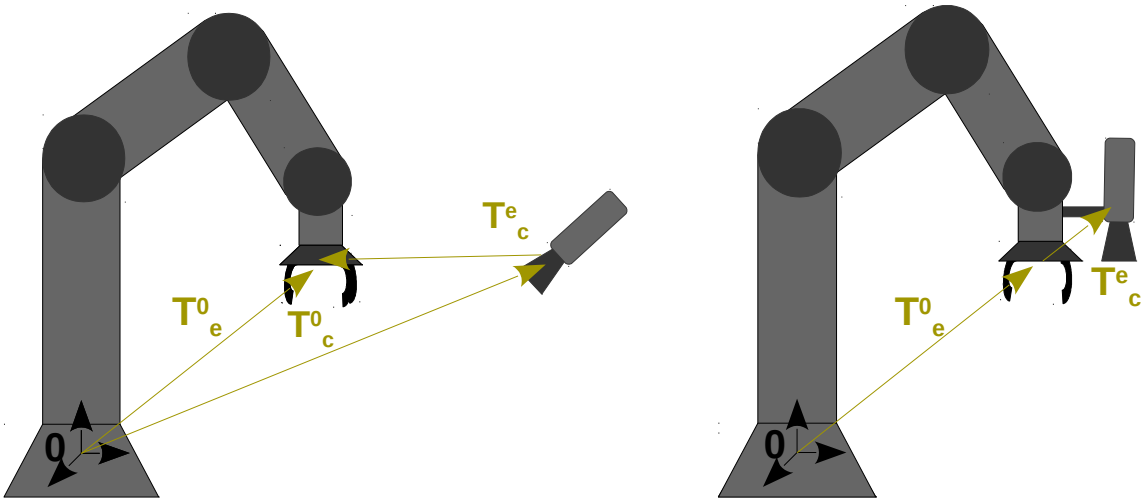

Figure 14: Camera configurations. Left: eye-to-hand. Right: eye-in-hand

6.1.1 Eye-in-hand

In the *eye-in-hand* system, there exists a known and constant relationship between the end-effector and the pose of the camera. This has the advantage that the target position estimation in the end-effector frame is direct. As a drawback, the camera's view is limited by the end-effector pose, so there is no view of the whole workspace, and as a consequence, it is more difficult to avoid obstacles.

6.1.2 Eye-to-hand

In the *eye-to-hand s*ystem. The camera observes the robot within its workspace, providing a panoramic view. In this configuration, obtaining the target's pose is necessary, and it is additional work to do.

6.2. VISUAL SERVOING TAXONOMY

Visual Servoing systems can be classified by their arquitecture in two main groups: Image-Based (or 2D) and Position-Based (or 3D) visual servoing systems.

6.2.1 Image-Based Visual-Servoing

Image-based visual servo control consists of a feedback signal that is composed of pure image-space information. Which control is based on how similar the desired image is with the current image, which actually knows that because of the error between current and desired features on the image plane. The features can be coordinates of visual features, lines or moments of regions. Several advantages characterize IBVS. Firstly, direct control of the feature motion in the image plane allows the implementation of strategies to keep the target always in the field of the camera's view, with approximately straight line trajectories for the image feature point. Another advantage of IBVS is that the positioning accuracy is insensitive to the camera and target modelling errors, eliminating errors due to the calibration. It is essentially a model-free method, without the explicit requirement of the target model in practical applications, and convergence is generally robust w.r.t. disturbances and uncertainties in the camera/robot model.

However, some knowledge of the transformation between the sensor and the robot frame is still required. There are a few drawbacks regarding IBVS. Firstly, image-based methods are that since the controller is based on image-feedback, the robot could be commanded to some configuration that is not physically possible. The end effector and the robot may reach their joint limits. Secondly, the end-effector translational and rotational motions are not directly controlled. The usual coupling between these motions makes it challenging to plan a pure rotational or a pure translational motion. Also, the controller stability analysis is difficult to obtain in calibration uncertainty. Furthermore, usual IBVS is only locally asymptotically stable and may fail in the presence of a large desired displacement, necessitating a path planning step to split a large displacement into smaller local movements. Finally, potential failure occurs when IBVS is subject to image singularities or local minima.

6.2.2 Position-Based Visual-Servoing

In Position-based visual servoing (PBVS), features are extracted from the image and used to estimate the target's pose concerning the camera. In this case, the camera detects the object, and the robot controller controls the end-effector. The principle advantage is that it is possible to describe tasks in Cartesian pose, as is common in robotics. Feedback is computed by reducing errors in estimated pose space. This approach avoids image-Jacobian singularity and local minima problems, generating physically realisable trajectories. However, the approach is susceptible to inaccuracies in the task-space reconstruction if the transformation is corrupted (e.g., uncertain camera calibration). Also, since the controller

does not directly use the image features in the feedback, the commanded robot trajectory may cause the feature points to leave the field-of-view. In that case, the 3D parameters have to be estimated from a pose estimation process using the knowledge of the 3D target model. PBVS is known to have global asymptotic stability, i.e., a controller's ability to stabilise the camera's pose from any initial condition if 3D estimation is perfect. When accurate 3D estimation is employed, decoupling rotation and translation is obtained. Calibration errors propagate to errors in the 3D world, so accurate 3D estimation is essential to ensure the robustness of PBVS.

To sum up, the main advantages PBVS provides are its performance's accuracy and robustness. Also, the control law design is more accessible than in IBVS because once the target position is known, the servoing scheme attempts to minimise it by moving the robot around, ideally towards the final desired pose. Moreover, it introduces several simple positioning primitives based on directly observable feature points, which can be compounded to achieve more complex positioning tasks.

Nevertheless, there are some disadvantages. PBVS relies too much on calibration. Feedback is computed using estimated quantities that are functions of the system calibration parameters. It can become susceptible to its error. Also, It is dependent on having an accurate model of the target object. Furthermore, the computation time is an issue because of the high computational effort. And the possible loss of target for significant errors.

6.3. OBJECT DETECTION

6.3.1 YOLO Object Detection

Prior CNNs were mostly used for image classification, which finds the most relevant class (e.g. cat, flower, car, etc.) given an input image. Later, CNNs were adapted for object detection: identifying and locating several objects in an image. Early object detection systems were simply repurposed classifier networks, applying them to various image subregions. The location and scale of the subregions can also be learned parameters. The runtime performance of these systems is poor since the classification process must be run many times.

YOLO (You Only Look Once) is a state-of-the-art, real-time object detection system that uses a single CNN to predict bounding boxes and class probabilities of objects. This means that prediction in the whole image is completed in a single algorithm run. This improves its runtime performance, enabling it to perform real-time object detection at more than 35 frames per second.YOLO is speedy and accurate. Furthermore, YOLO reasons globally about the image when making predictions. It sees the entire image during training and test time, so it implicitly encodes contextual information about classes and their appearance. This decreases the number of background errors. In addition, YOLO learns generalisable representations of objects, so it outperforms top detection methods like DPM and R-CNN by a wide margin. It is less likely to break down when applied to new domains or unexpected inputs.

Moreover, it is accessible to tradeoff between speed and accuracy simply by changing the model’s size without retraining it. Also, it is difficult for YOLO to detect small objects that are very close or intersecting with each other. See an example in Figure. 14.

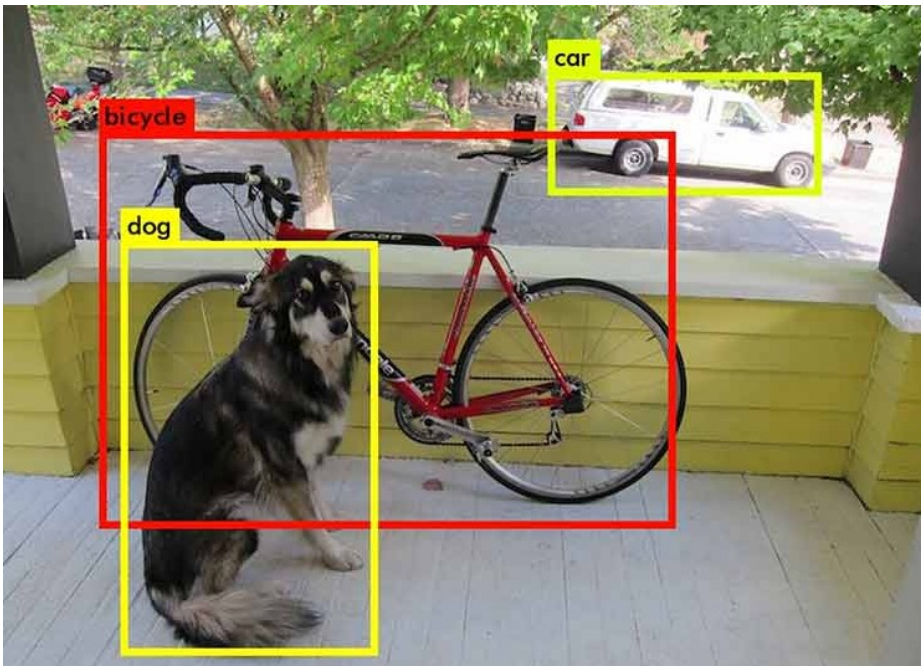

Figure 15: YOLO object detection example.[21]

6.3.2 DOPE (Deep Object Pose Estimation)

While 2D object detection problems have been successfully applied by deep neural networks, 3D object detection and pose estimation have been recently applied. Unlike 2D object detection, manually labelling data for 3D detection is not a possibility. Due to this difficulty of collecting large amounts of labelled training data, such approaches are typically trained on actual data that are highly correlated with the test data (e.g., same camera, same object instances, similar lighting conditions). As a result, one challenge of existing approaches is generalising to test data significantly different from the training set. Synthetic data is a promising alternative for training such deep neural networks, capable of generating an almost unlimited amount of pre-labelled training data with little effort.

Networks trained on synthetic data usually do not perform well on actual data without additional fine-tuning or other tricks. However, this problem is solved by domain randomisation.

This one-shot deep neural network can perform competitively against a state-of-the-art network trained on a combination of real and synthetic data. It is called DOPE (for “deep object pose estimation”) and infers, in near real-time, the 3D poses of known objects in the clutter from a single RGB image without requiring post-alignment. This system uses a simple deep network architecture, trained entirely on simulated data, to infer the 2D image coordinates of projected 3D bounding boxes (See Fig.15), followed by perspective-n-point (PnP). It bridges the reality gap for real-world applications by combining non-photorealistic (domain randomised) and photo-realistic synthetic data for training robust deep neural networks. Thus, its performance is comparable with state-of-the-art networks trained on real data. Furthermore, the estimated poses are of sufficient accuracy to solve real-world tasks such as pick-and-place, object handoff, and path following

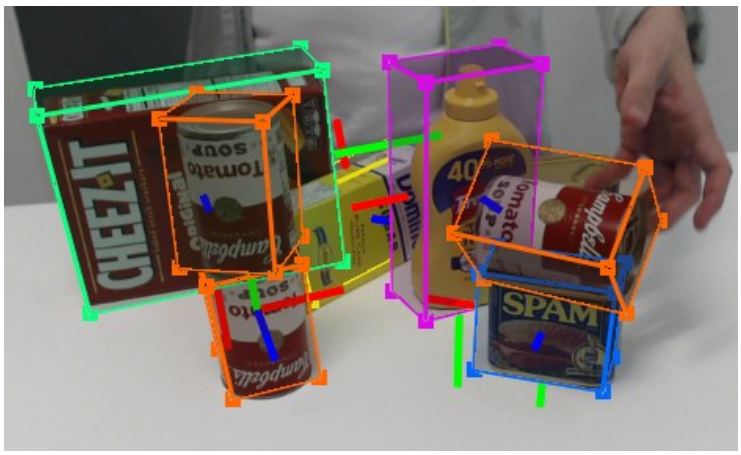

Figure 16: DOPE example which shows the 3D bounding boxes in a 2D image

7. PROPOSED SOLUTION

7.1. General DescrIption

The proposed solution consists of an eye-to-hand camera configuration because it is embedded with the robot and implementing another camera on the hand would increase the price and complexity of the task. In addition, by pointing it at the workspace, it is easier to avoid collisions.

The PBVS taxonomy has been chosen because of its versatility in actuator control. TIAGo is a well-calibrated robot; the task's success depends mainly on the object's positioning. Image features are extracted to estimate 3D information (pose of the object in Cartesian space) from the geometric model of the object. The control function is separated from the pose estimation problem. In this case, the feedback is calculated by reducing the errors of the estimated pose. The kinematic error function is given by (7.1):

$$E(h_0, g_0) = h_0 - g_0 \tag{7.1}$$

Where h_0 are the end-effector coordinates in the base root frame of TIAGo (0), we control this variable to move the end-effector to the desired pose, in this case, a fixed goal position g_0.

Applying the linear velocity u_0 to the end-effector minimises the above error. The proportional control law can achieve open-loop positioning.

$$T^{\wedge 0}_{c} \cdot \hat{g}_c \simeq g\,0 \tag{7.2}$$

$$u_0 = -k(\hat{h}_0 - \hat{T}^0_c \cdot \hat{g}_c) \quad (7.3)$$

Where $\hat{T}^0_c$ is the estimated transformation matrix from the camera frame to the root frame (usually based on fixed transformations from manual measurements) and $\hat{g}_c$ is the estimated grip pose of the target in the camera frame, given by the vision system. k > 0 is a proportional feedback gain.

It uses YOLO for image detection and location because it was used for different projects previously, and there was a trained model with the objects for the ERL Consumer. Also, it is fast and less computational consuming than other object detection approaches.[21]

The system divides the input image into an S × S grid. Each cell predicts B-bounding boxes centred on that cell. Confidence scores are assigned to the boxes, the confidence that shows how likely it is for the box to contain an object and how accurate the box is, measured by the intersection-over-union (IOU) metric. Each cell also predicts C class probabilities P (Class i | Object), where C is the number of detectable classes. This number is independent of the number of bounding boxes B. This means it is difficult for YOLO to detect small objects that are very close or intersecting with each other. See fig 17.

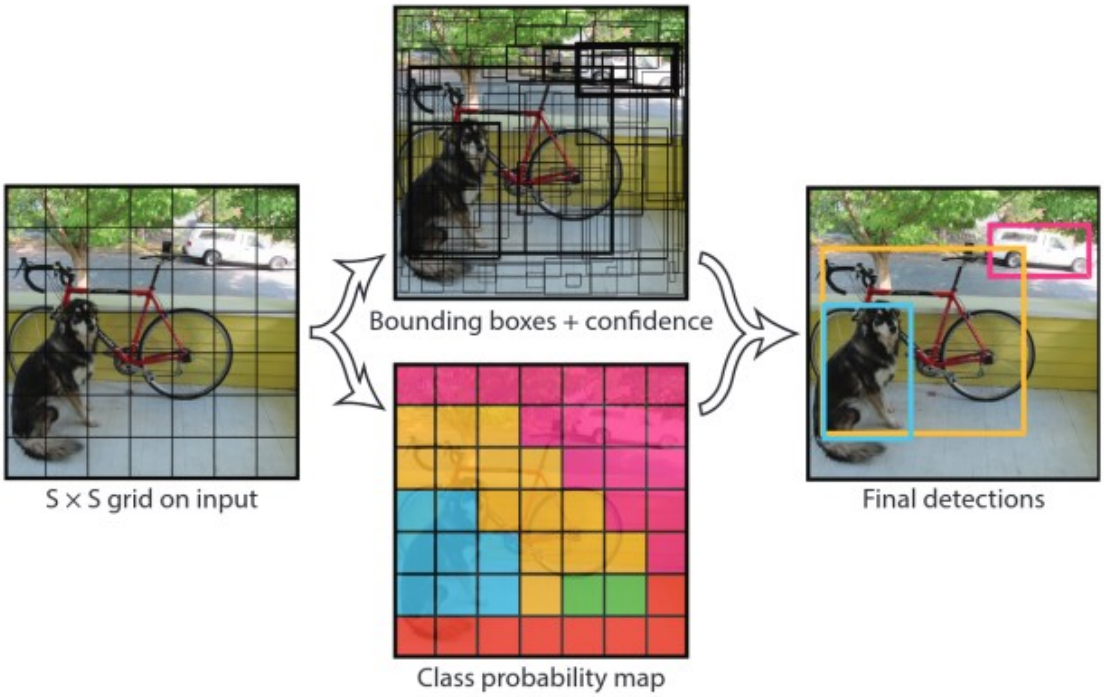

Figure 17: YOLO model predictions: bounding boxes and class probabilities are combined to optain the final result: object labels and their bounding boxes [22]

The YOLO CNN has 24 convolutional layers, with maxpool layers between them. 2 fully connected layers are placed at the end. Using YOLO, the first 20 convolutional layers are pretrained on the ImageNet dataset to detect and locate the object.

7.2. Pipeline

There are five phases which divide the task: TIAGo's preparation and detection of the objects (preparation), grasp acquisition (grasping), post-grasp transport (post-grasp), place of the object (placing), and leave the workspace (post-place); in brackets are indicated names for the pipeline. Each of these phases carries sub-tasks inside of them. See Figure 18.

Figure 18: Pipeline applied to perfomr the pick and place task

7.2.1 Preparation

Initialise the robot, every node and prepares the robot for object detection. To do it, TIAGo raises the torso, unfolds the arm, opens the gripper and moves its head down to see the objects. Then it lowers the torso a bit to get a different sight of the objects. After localising the object, creates a grasp location adding an offset to the object pose to facilitate the grasping. Then, it adds to the scene a virtual object and a virtual table to plan the grasping from them.

7.2.2 Grasping

TIAGo locates the end effector near the grasping pose, then moves the gripper linearly until it reaches the grasping pose and closes it. After this, it checks whether the grip was completed successfully.

7.2.3 Post-grasp

After the grasping phase, raise the arm and torso to their limit and hold the object until starts the next phase.

7.2.4 Placing

TIAGo moves the end effector just above the placement position, then is ready to place the object in a linear movement. When it reaches the placement position, it opens the gripper releasing the object.

7.2.5 Post-place

This is the last phase. It moves the arm to a secure position to avoid disturbing the workspace with the next movement. From this position, the robot comes back to the home pose to finalise its task.

7.3. Implementation

This section divides the implementation of the project into two parts: required packages, pick and place implementation.

7.3.1 Required Packages

These are the required packages implemented in this project.

MoveIt!

This is a ROS package used for motion planning and task handling. It consists of an inverse kinematics solver, path planning algorithms and collision detection. This project uses the TRAC-IK solver [24] because it obtains reasonable resolution rates for challenging positions while keeping

the computation time low by running two solvers concurrently. One of them is called KDL solver, which detects and mitigates local minima that may occur when joint boundaries are encountered during gradient descent. The other is called SQP (Sequential Quadratic Programming) IK, which uses quasi-Newton methods[Reference] that are known to handle non-smooth search spaces better but with longer computational time - and stopping when the first solution is found.

The Open Motion Planning Library (OMPL) [25], a collection of sampling-based motion planning algorithms, is the default planner in MoveIt! The RRT-Connect algorithm [26] was chosen for its efficiency, which is achieved by building two Randomized Random Exploration Trees (RRTs), one from the start point and one from the endpoint, progressively moving towards each other until they are connected.

Play motion

The function of this package is to be able to execute predefined motions.

7.3.2 Pick And Place Implementation

Launch files

Launch files are essential for starting the main features of the package.

Launch files are XML format files that, using the roslaunch tool, allow the start of the master node (necessary for ROS 1 to work) and multiple nodes simultaneously. They can manage the passing of arguments and the

loading of parameters, possibly by calling additional launch files and thus generating a launch-tree.

This project uses three fundamental launch files. The first one starts the simulation in gazebo, places the robot in front of a table with three cans and enables the TRAC-IK solver. TIAGo's manufacturer, PAL Robotics, developed this document. To access it, download the TIAGo tutorial repository [Reference]. To run it, it is necessary to enter into the workspace folder and run the following command:

```
IK_SOLVER=trac_ik roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=true robot:=steel world:=tutorial_office gzpose:="-x 1.40 -y -2.79 -z -0.003 -R 0.0 -P 0.0 -Y 0.0" use_moveit_camera:=true
```

Terminal 1: Command line to initialise the gazebo simulator and use trac-ik as ik solver.

The following launch file activates object tracking. It activates a node called "*bayes_object_tracker*", which is in charge of tracking the detected objects. In addition, this file also calls two additional launch files: *darknet_ros_py.launch*, in charge of detecting objects, and *mbot_perception.launch*, used to log the detected objects. This is done by running the command:

```
roslaunch bayes_objects_tracker bayes_objects_tracker_no_namespaces.launch
```

Terminal 2: Command line to initialise the object tracker.

The last launch file is responsible for loading the predefined motions for the tasks, starting the *pick_and_place_server node* by adding the target object parameters and opening the Rviz model, ready to observe the

robot's behaviour. It is necessary to understand how these actions are implemented.

- *Predefined motions*

There are some predefined motions used during this proyect. To create them there are three steps to follow.

1. Pose the robot
2. Capture the values of the joints
3. Save joints' values in a yaml file

To pose the robot to a desired position, there are many ways. It can be done manually with the physical robot by moving every joint or either use rqt_joint_trajectory_controller or the motion planning tool in Rviz.

In the rqt_joint_trajectory_controller way, there is a menu where it is possible to select the group of joints and then it is allowed to change the position of every joint. To do it, just run:

```
rosrun rqt_joint_trallectory_controller  rqt_joint_trallectory_controller
```

Terminal 3: Command line to open a graphical frontend for interacting with joint_trajectory_controller instances

Press the power button to set up, select the group of joints and change the position of the joints. As soon it is visible the desired position, there are two ways of continue. See figure 19.

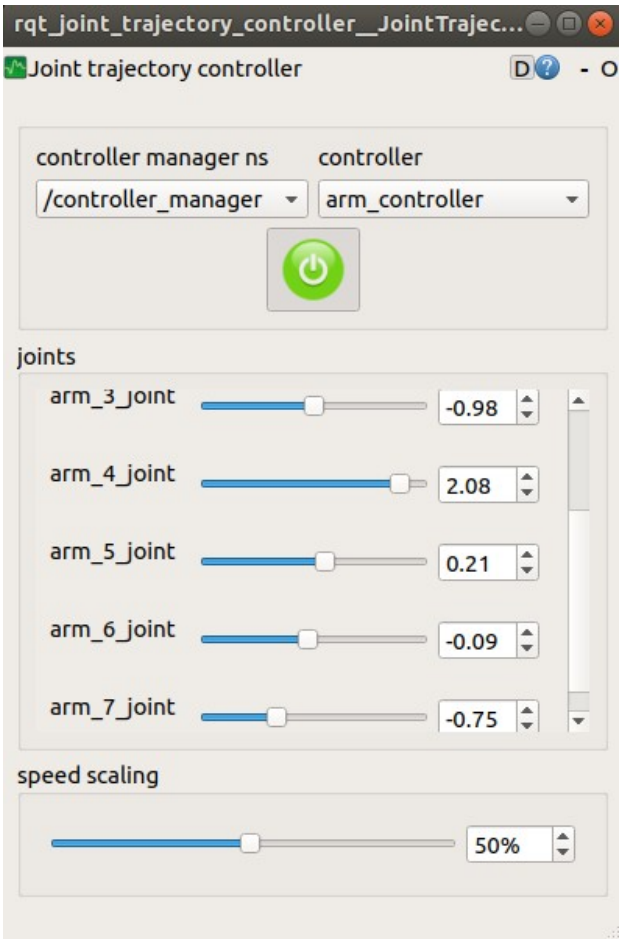

Figure 19: Rqt joint trayectory controller GUI

First is done by reading the state of the arm and torso and writing them into a yaml file. To read the state of the arm:

```
rostopic echo /arm_controller/state
```

Terminal 4: Command line to read the arm controller state

To read the state of the torso:

```
rostopic echo /torso_controller/state
```

Terminal 5: Command line to read the torso controller state

Then, write the values into the *pick_motions.yaml* file. It should be similar to the example below.

```
play_motion:
  motions:
    pregrasp:
      joints: [torso_lift_joint, arm_1_joint,
      arm_2_joint, arm_3_joint, arm_4_joint, arm_5_joint,
      arm_6_joint, arm_7_joint]
      points:
      - positions: [0.15, 0.20, -1.34, -0.20, 1.94, -1.57, 1.37, 0.0]
        time_from_start: 0.0
      - positions: [0.34, 0.20, -1.34, -0.20, 1.94, -1.57, 1.37, 0.0]
        time_from_start: 2.0
      - positions: [0.34, 0.21, -1.02, -0.20, 1.94, -1.57, 1.52, 0.0]
        time_from_start: 3.5
      - positions: [0.34, 0.21, 0.35, -0.2, 2.0, -1.57, 1.52, 0.0]
        time_from_start: 6.5
      - positions: [0.34, 0.21, 0.35, -0.2, 0.0, -1.57, 1.52, 0.0]
        time_from_start: 10.0
      - positions: [0.34, 0.21, 0.35, -3.0, 0.0, -1.57, 1.52, 0.0]
        time_from_start: 12.0
      - positions: [0.34, 0.05, -0.07, -3.0, 1.5, -1.57, 0.2, 0.0]
        time_from_start: 17
```

Code extraction 1: Example of a predefined motion in a .yaml file.

The other way of doing it is with the play_motion_builder.

This tool presents a simple interface to create play_motion based motions by defining lists of keyframes which the system will then interpolate between. The tool simplifies capturing, editing and modifying these keyframes.

To create new motions play_motion_builder packages have to be installed correctly[27].

Play motion builder must to be running, to do it:

```
rosrun play_motion_builder play_motion_builder_node
```

Terminal 6: Command line to run play_motion_builder_node

Then, we run rqt_play_motion_builder to use the tool which allows simple control of the motion creation pipeline. See figure 20.

```
`rosrun rqt_play_motion_builder rqt_play_motion_builder``
```

Terminal 7: Command line to run GUI to handle the creation of new motions for play_motion called rqt

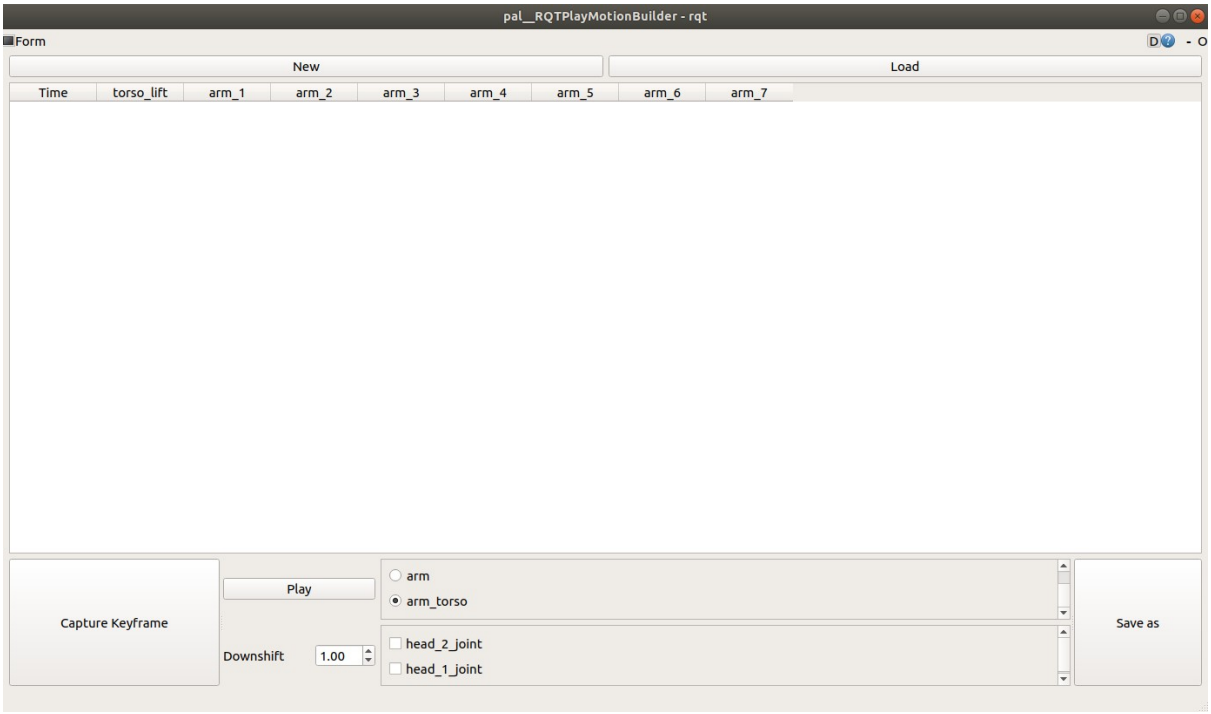

Figure 20: rqt play motion builder GUI

Press *New* to set up, there is a box where it is able to select the group joints which are wanted to safe and also can be added the head's joints if

it is required. Then, press to *Capture Keyframe* and edit the time if it is needed. Save it and it can be used with the rest of pre-defined configurations.

After all, in both ways it is needed to load the yaml file to the parameter server.
rosparam load <*yaml_file_path*> the new motions are saved:

```
rosparam load <yaml_file_path>
```

Terminal 8: Command line to load a file in the project

- *Rviz test environment*

The rviz model of the project was the default model in the octomap tutorial, edited later to suit the project better. To do this, it ran the launch files needed to activate the nodes and the rviz model in the terminal:

```
roslaunch tiago_moveit_tutorial octomap_tiago.launch
```

Terminal 9: Command line to laucnh the octomap_tiago.lauch which initialises octomap.

Moreover, from this RViz model, complementary tools were added. When clicking on add, a window opens that shows the possibility to create a visualisation from the display type and add from the topic. In this case, the last was chosen. See Image ... All active topics are displayed here. From there, it is easy to add them:

/bayes_object_tracker/pose_array - PoseArray
/object_detector/detection_image/compressed - Image
/object_localizer/localized_object_poses - PoseArray

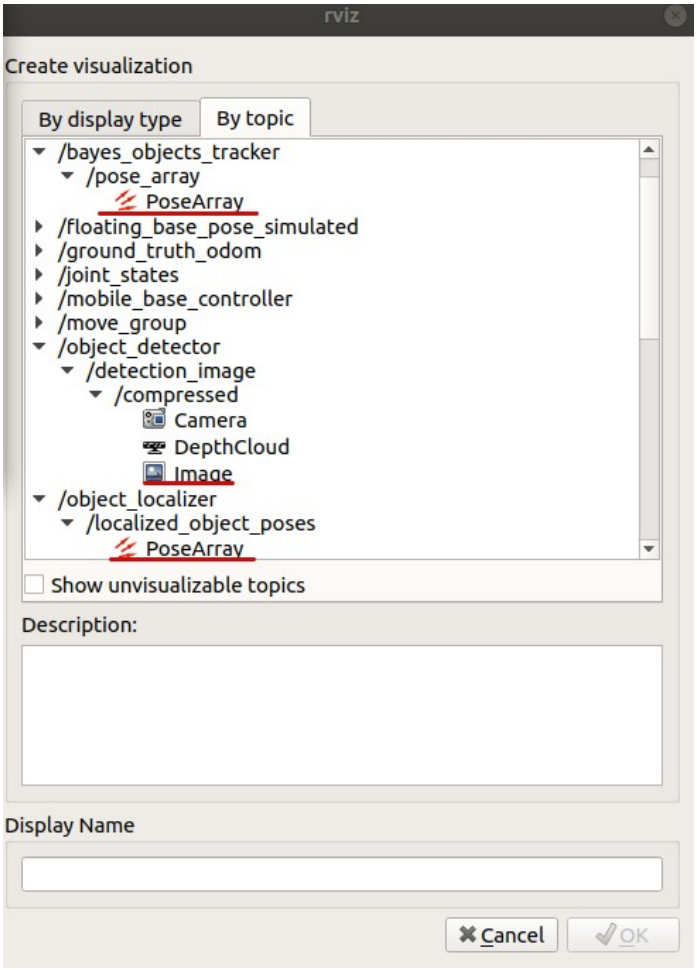

Figure 21: Rviz configuration by topics underlined

Next, the gripper frame is added. To do this, click on add, in by display type search for tf in the rviz folder and leave only gripper_grasping_frame visible. To change the colour of the PoseArray, select any of them. The name of the topic and the colour appears. By clicking on it, it can be changed easily. See image 22.

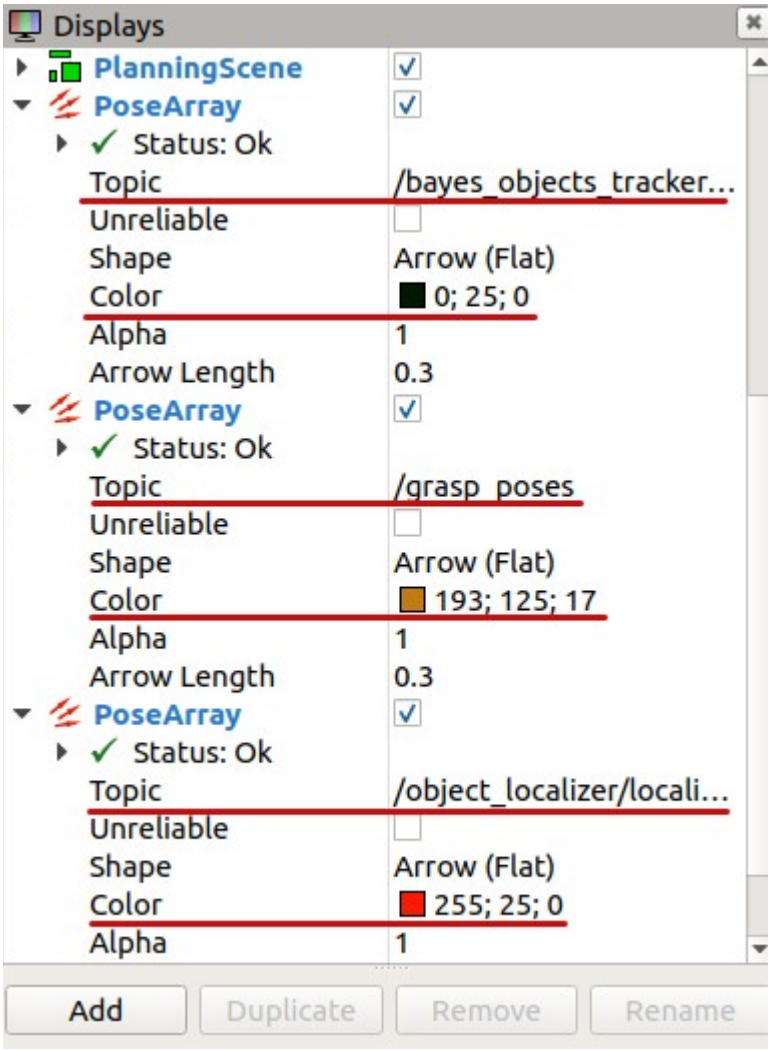

Figure 22: How to change PoseArray colours in Rviz

Knowing this, the pick_place_server.launch can be run in the terminal:

```
roslaunch manipulation pick_place_server.launch
```

Terminal 10: Command line to run the pick_place_server.launh which initialises the server.

pick_and_place_server.py

This code is inside the tiago tutorials folder. The project works with a system of actions in which the server manages the project's tasks and messages. Because there is a pick and place demo, it has been decided to use the same server as it fits with this project editing the object parameters. For this purpose, it has been added to the launch file as follows:

```
 <!-- Pick & place server -->
<node name="pick_and_place_server" pkg="tiago_pick_demo"
type="pick_and_place_server.py" output="screen">
<rosparam command="load" file="$(find
tiago_pick_demo)/config/pick_and_place_params.yaml" />
<param name="object_width" value="0.11" />
<param name="object_height" value="0.07" />
<param name="object_depth" value="0.11" />
</node>
```

Code extraction 1: Pick and place server node declaration in pick_place_server.launch

manipulation.py

The manipulation actions are implemented in this file. When this code is executed, the task is started. The design pattern used by the team is singleton, so this code consists of only one class. For this reason, two classes have been imported from different files called: PickObjectPose() and PlaceObjectPose().

```
import rospy

from pick import PickObjectPose
from place import PlaceObjectPose
```

Code extraction 2: How the classes PickObjectPose and PlaceObjectPose are imported.

Where the program runs:

```
# Usage Example
if __name__ == '__main__':

rospy.init_node('manipulation_client')

manipulate = Manipulation()

manipulate.pick(pick_object_uuid='e7be927c-3e58-507f-a9e6-

eaadce759be2',pick_object_name='cup', lift_object=True)

manipulate.place(predefined_location='table', tuck_arm=True)
```

Code extraction 3: Usage example from manipulation.py.

Where, it is shown *manipulate.pick(pick_object_uuid='e7be927c-3e58-507f-a9e6-eaadce759be2',pick_object_name='cup', lift_object=True),* It calls a function of the PickObjectPose() class and *manipulate.place(predefined_location='table', tuck_arm=True)* calls another one of PlaceObjectPose().

pick.py

This is the code from which the PickObjectPose() class comes. Its pseudo-code is presented below.

```
class PickObjectPose(object):
def __init__(self):

def pick(self, pick_object_uuid=None, pick_object_name=None,
lift_object=True):

   if not pick_object_uuid and not pick_object_name:#Exeption

   self.prepare_robot()

   object_detected = False

   if pick_object_uuid: # localized_objects =                    rospy.wait_for_message("/
bayes_objects_tracker/tracked_objects",     TrackedObject3DList)

   elif pick_object_name: # localized_objects =

   rospy.wait_for_message("/object_localizer/localized_objects",

   RecognizedObject3DList)

   object_perceived = object_detected

   if not object_detected:# Failed to detect object

   self.transformToBaseFrame(object_perceived)

   object_to_grasp = object_perceived

    self.transformToBaseFrame(object_to_grasp)# Add offset to the pose

   self.pick_as.send_goal_and_wait(object_to_grasp)

   rospy.loginfo("Done!")
   if lift_object:

      self.move_arm_to_post_grasp()

   rospy.loginfo("Pick Success")


   return Truee
```

Code extraction 4: Pseudo-code from pick.py

There are several functions to discuss from Code extraction 4.

Firstly, the *init()* function is in charge of initialising the Pick Client and Object tracker, connecting to the /pickup_pose action server, setting publishers to the torso and head controller, connecting to /play_motion action server, and clear octomap service.

The *prepare_robot()* function is in charge of unfolding the arm safely, opening the gripper and looking from different sights to detect objects.

At this point, depending on the argument received it will read the topic */bayes_objects_tracker/tracked_objects* or */object_localizer/localized_objects*. In either case, it senses the object and stores its pose in the detected object.

As the pose of the detected object is given with the camera frame, it is entered into a function that transforms the pose to the *base_footprint* frame.

This pose is stored in a variable called *object_to_grasp*, and an offset is added to it. The object detected pose is from the front face of the object and seeks to be grabbed from the centre. Therefore, a slight offset is added to correct this error.

Once the pose is saved, it is sent to the *pick_as*, which is in charge of performing the grasping routine. This routine has no visual feedback because it was not implemented due to some limitations explained later.

When finished, it lifts the object and finishes this part of the task.

place.py

This is the code from which the *PlacebjectPose()* class comes which is slightly different from *pick.py*. Its pseudo-code is resented below.

```
class PlaceObjectPose(object):
      def __init__(self):
      def place(self, location_uuid=None, location_name=None,
location_pose=None,
      predefined_location=None, tuck_arm=True):

   if not location_uuid and not location_name and not location_pose and not

   predefined_location: # Execption

   self.lower_head()

   if location_uuid: # The idea is to perceive a suitable location and place the

   object there

   elif location_name: # The idea is to place the object above an object given

   as parameter.

   elif location_pose: # Here the location pose is given

   elif predefined_location: # Here a predefined motion name is given to run it

   self.predefined_motion('open') # open the gripper

   self.move_arm_to_post_place()

   if tuck_arm: # tuck the arm back

      self.predefined_motion('home')

   rospy.loginfo("Place Success")


   return True
```

Code extraction 5: Pseudo-code from place.py

In Code extraction 5, it is also interesting to consider a few things. The init() function does the same as in *pick.py*.

Next, it lowers its head to get a better view of where to release the object.

Then, depending on the given argument, it takes the object pose data as written in the comments. To reduce the complexity of the task, it performs a predefined motion for placing.

When the object is placed in the given pose, it opens the gripper and moves the arm to a safe pose that does not interfere with the workspace by executing the *move_arm_to_post_place()* function.

Finally, TIAGo returns to the home pose and finishes the task.

7.4. Debugging

Debugging in ROS could be challenging. There are many packages connected, and it is difficult to understand where the error comes from if there is no organisation in the program. Since this project is done with part of the work from previous teammates in the group with different coding styles and robots, merging these functionalities with this project could be arduous. That is why some tools help understand what is going on in the project.

7.4.1 Printing Messages In The Terminal

One of the most basic techniques for debugging our system is printing messages in the terminal, that says which function is running at every moment.

7.4.2 Command lines

Some *commands* are substantial to read what the nodes are doing and how they are connected.

-rosnode list: Display a list of active nodes from our application
-rosnode info: Print information about the node
-rostopic list: Display a list of active topics from our application
-rostopic info: Print information about the node
-rostopic echo: Print messages to the screen
-rosservice list: Lists active services from the application
-rosservice type: Display the name of the node that provides a particular service

7.4.3 Rviz

Rviz is a visualiser for ros. It can represent in a graphical way what nodes see. In this project, there is a rviz configuration. This shows a 3D model of the robot, the gripper's frame (which shows a 3D representation of the environment seen by the robot called octomap), the position of the object detected with the preteained neural network and another trained for this project, and a virtual model of the object and the table with the possible grasping poses as it shows the figure 23.

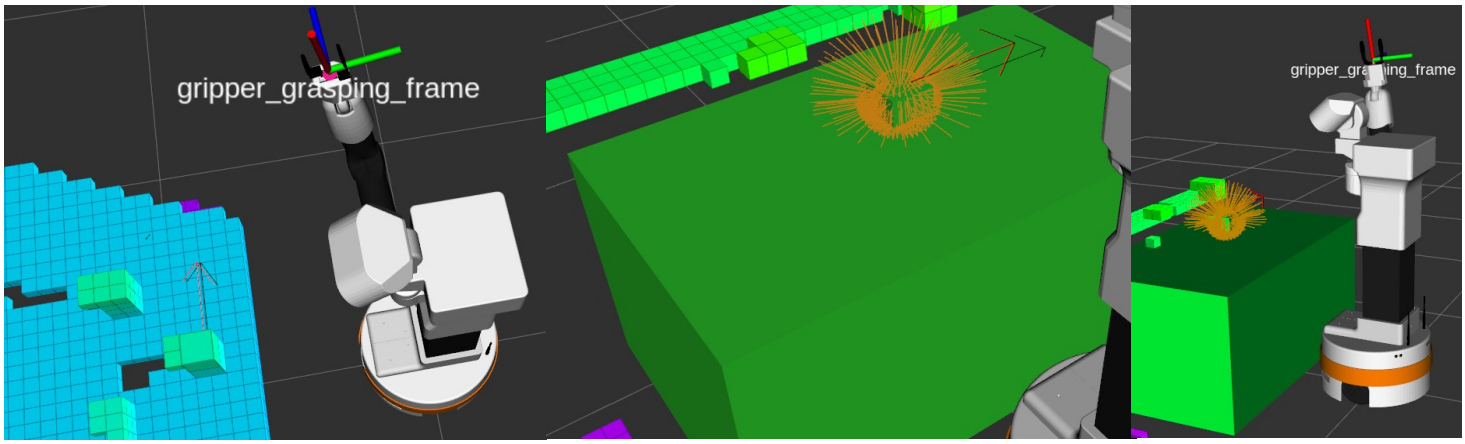

Figure 23: The first capture shows the octomap view with the pose of the object reprensented by arrows, the red is localised by the pretrained neural network and the black with the trained. The following capture shows the virtual object and table with the possible grasp in yellow. The last capture shows all together.

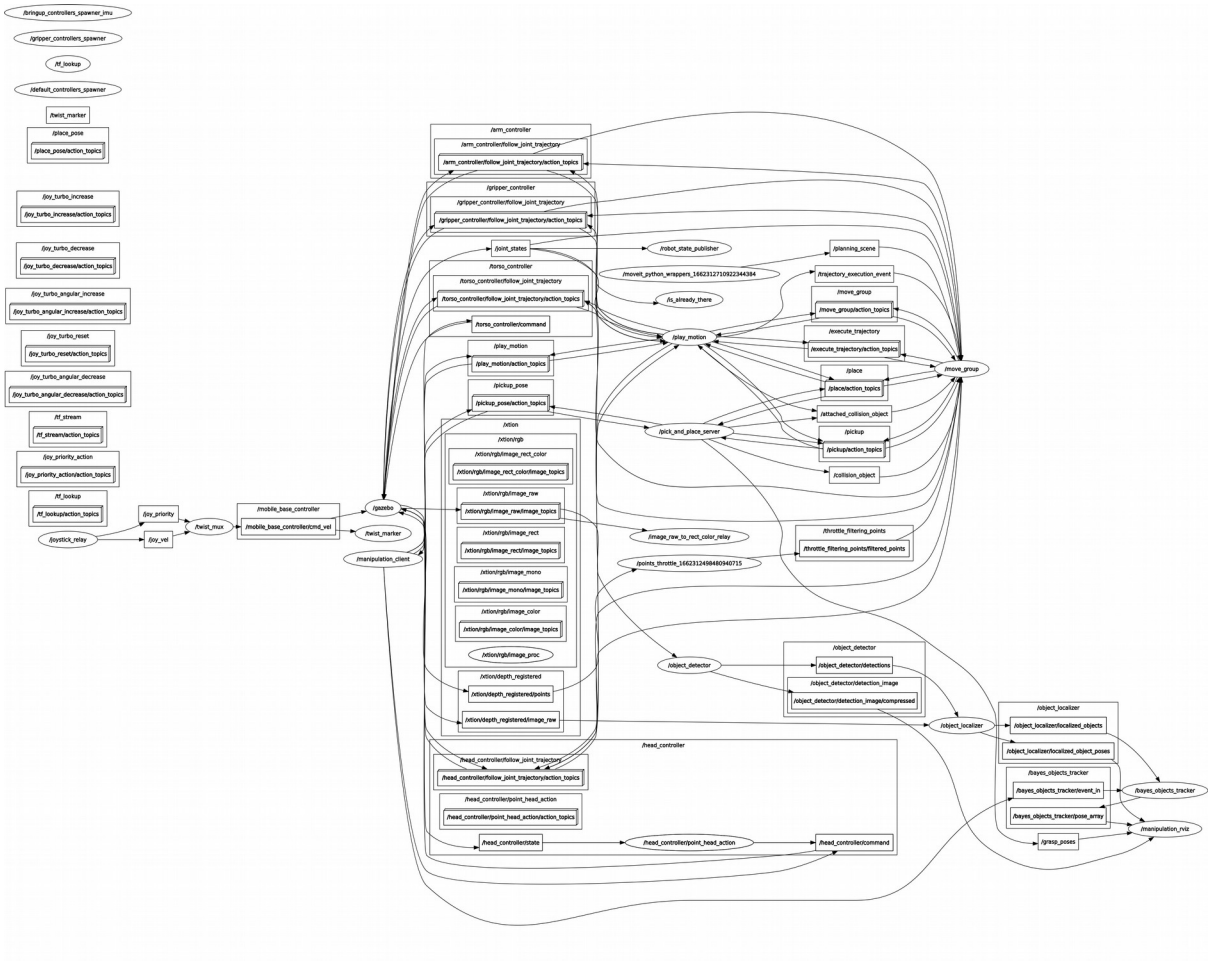

Figure 24: Shows the magnitude of the computation graph

8. EXPERIMENTS AND RESULTS

In this section, the experiments done are discussed. These experiments are divided into two main blocks: functionalities check and pick and place.

8.1. FUNCTIONALITIES CHECK

Before developing the project task, it has been checked that all the components work correctly. For this purpose, the official TIAGo tutorials

[Reference] developed by PAL Robotics have been used. These are useful to understand how TIAGo works and its limitations. After the tutorials were completed, a test environment was created to carry out one's experiments. Finally, object detection has been tested.

8.1.1 TIAGo Tutorials

Several tutorials have been beneficial.

Joint trajectory Controller

Use *joint_trayectory_controller* to move the TIAGo arm showing the type of messages it uses. The mechanisms described for sending trajectories to the controller are through actions or topics. The results are correct, and it moves on to the next one.

Playing pre-defined upper body motions.

Learn how to create, visualise and run pre-defined upper body motions with TIAGo using the play_motion package, which enables executing simultaneous trajectories in multiple groups of joints. Different motions have been created to test their operation at the limits of the robot. With RViz motion planning, moves the robot, extracts and saves the values of the joints with the other motions. After achieving positive results, we move on to the following tutorial.

Planning in Cartesian space.

This tutorial teaches the use of MoveIt! to place the end effector frame at the desired position in Cartesian space. It has been tested by sending the

arm to random positions, some of which were impossible and aborted its execution. There are positive and robust results, so we move on to the following tutorial.

Planning in Cartesian space with TRAC-IK

Unlike the previous tutorial, this one uses TRAC-IK, an alternative Inverse Kinematic solver. TRAC-IK runs two IK implementations. One is a simple extension to KDL's Newton-based convergence algorithm that detects and mitigates local minima due to joint limits by random jumps. The second is an SQP (Sequential Quadratic Programming) nonlinear optimization approach which uses quasi-Newton methods that better handle joint limits.The results are also correct.

Planning with Octomap demo

This tutorial is an example of using Octomap in Movit! to plan with TIAGo. It gives a 3D representation of the environment around the robot, letting the robot avoid them in the planning.

Pick & Place demo

It is a grasping example with TIAGo. A simulation environment comprising a table and a box with an ArUco marker is defined. The robot then locates the object in the RGB of its camera and reconstructs its 3D pose. Then, MoveIt! is used to plan a pick trajectory to grasp the object, which is lifted up and a place trajectory is planned to restore the object to its former position. Sometimes release the object in the lifting part, failing the task. Nevertheless, this is a great beginning.

8.1.2 Test Environment

This is a simulated robot space in front of a table with three cans on top of it. In this space, the robot can carry out its experiments with ease. To check if it works correctly, RViz is opened so that a graphical representation of the individual nodes can be seen. This environment helps with debugging.

8.1.3 Object detection

Testing the object detector in the simulator runs the test environment and the perception module. When the robot is looking at the objects, it is able to recognise the can in the centre but has difficulty recognising the ones on the sides, and if it does, it distinguishes mainly the one on the right. This is because the can in the centre is more prominent, closer, and the difference in colour to the background is more striking. Moreover, when it distinguishes the two cans, it recognises them as different classes: the can in the centre is recognised as a cup, while the one on the right is recognised as a bottle. The reason for these results is that the neural network has not been trained with the can class, and as they have different sizes and colours, it does not associate them with the same class. On the other hand, these locations are correct, being the decisive part of performing the task. Therefore, despite the errors in recognition of the object, this phase is admissible because its location is correct, and the rest of the project can continue without further modifications. (See Fig.20)

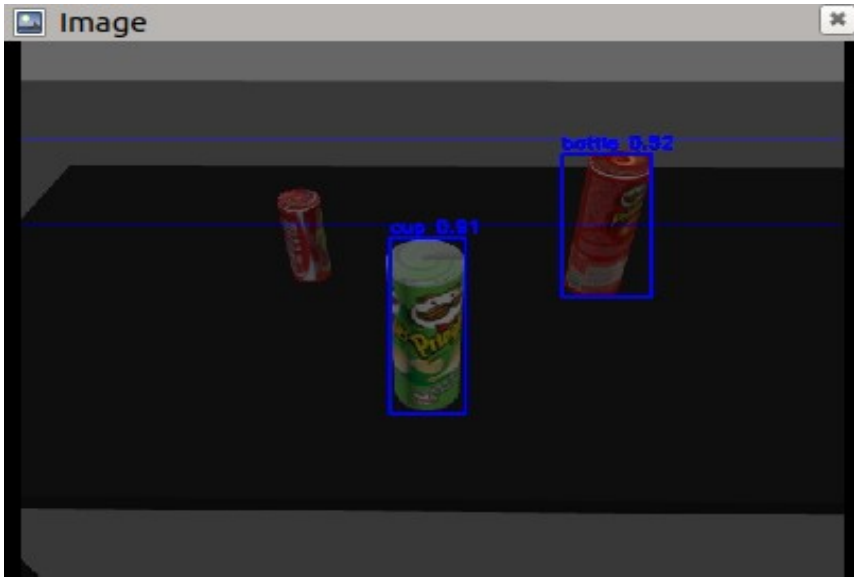

Figure 25: Image from the object detection with two objects detected and the confidence of their recognision

After these experiments, it is concluded that the robot fulfils the necessary capabilities to perform the pick and place task.

8.2. PICK AND PLACE TASK

TIAGo has to perform a pick and place task with one of the test environment cans. In one of the steps discussed below, a bug appears that makes it impossible to finish the task. Therefore, the experiment is divided into five pipeline phases:

8.2.1 Preparation

Before starting the grasping, the robot must be prepared.

To do this, it initialises and connects all the necessary nodes to start the task. While this happens, it prints messages on the terminal as it completes each step to check its status.

Then, the robot raises its torso, lifts its arm into a comfortable position and opens the gripper. This pose allows the arm to move towards its target without taking complicated trajectories to avoid the table. From this position, the robot lowers the head to locate the objects and lowers his torso to see them from a different perspective. This increases the probability of detecting them. This practice was added after several experiments as it was difficult to detect objects occasionally. After this correction, the results improved considerably.

Next, save the can's coordinates, and add a virtual model of the object and the table to the workspace (See Fig. 21). After several tests, an offset was added to the object's location by placing it back further. This is because it locates the object's front face instead of at its centre, giving errors in the next phase (grasping).

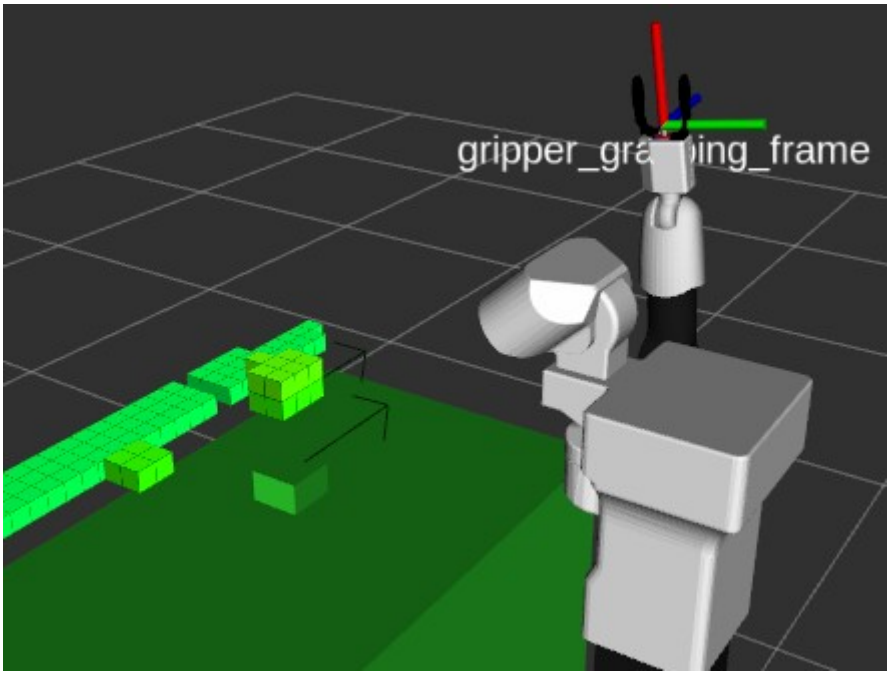

Figure 26: Virtual table and virtual object represented in green

Finally, calculate the different grasping possibilities and choose one. These grasping possibilities are represented in Rviz in Figure 27.

The preparation phase works correctly on most occasions, but when it fails, it is due to two main failures. The first one is in object detection since, for unknown reasons, the detector does not work, and the program has to be restarted. The second is calculating trajectories because sometimes it saturates without getting a result and has to abort the task. The calculating trayectories error was solved using TRAK-IK solver, the other error occur infrequently, The project development has continued , as this error is usually solved by restarting the program.

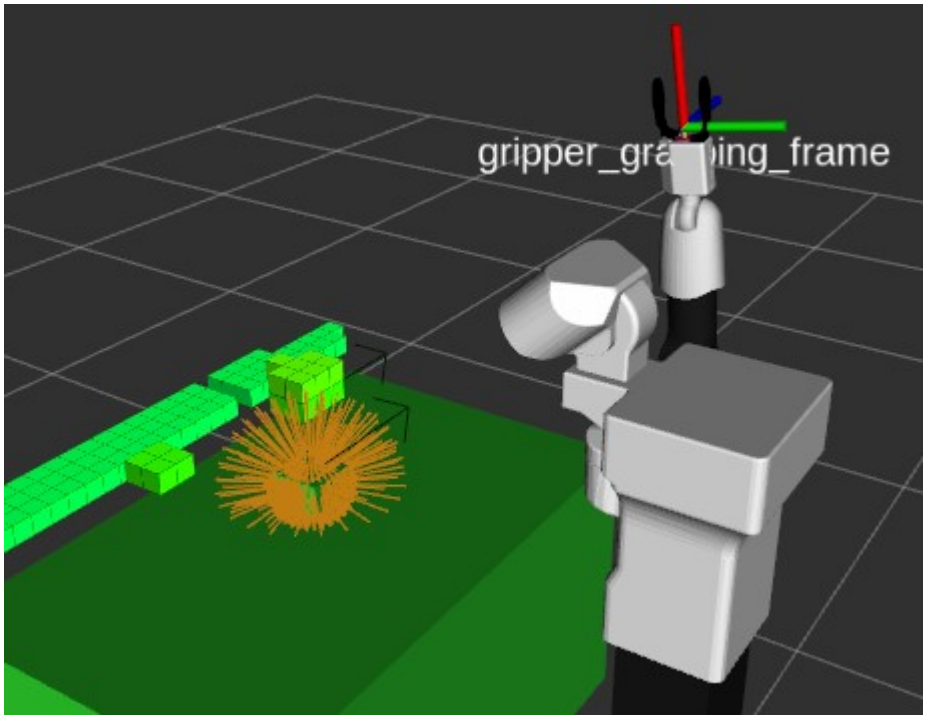

Figure 27: Virtual objects in green and possible grasp poses in yellow.

8.2.2 Grasping

Once the object is located, and the orientation of its grip has been chosen, the next step is grasping. TIAGo places the arm close to the object at a point where it starts a rectilinear trajectory to the grasping position of the object. Then, it closes the gripper and checks that the object has been grasped correctly.

TIAGo usually succeeds in grasping the object, but the results are confusing. When it successfully grasps the object, it sends an error message and aborts the task by returning to the robot's home position. See Fig. 28.

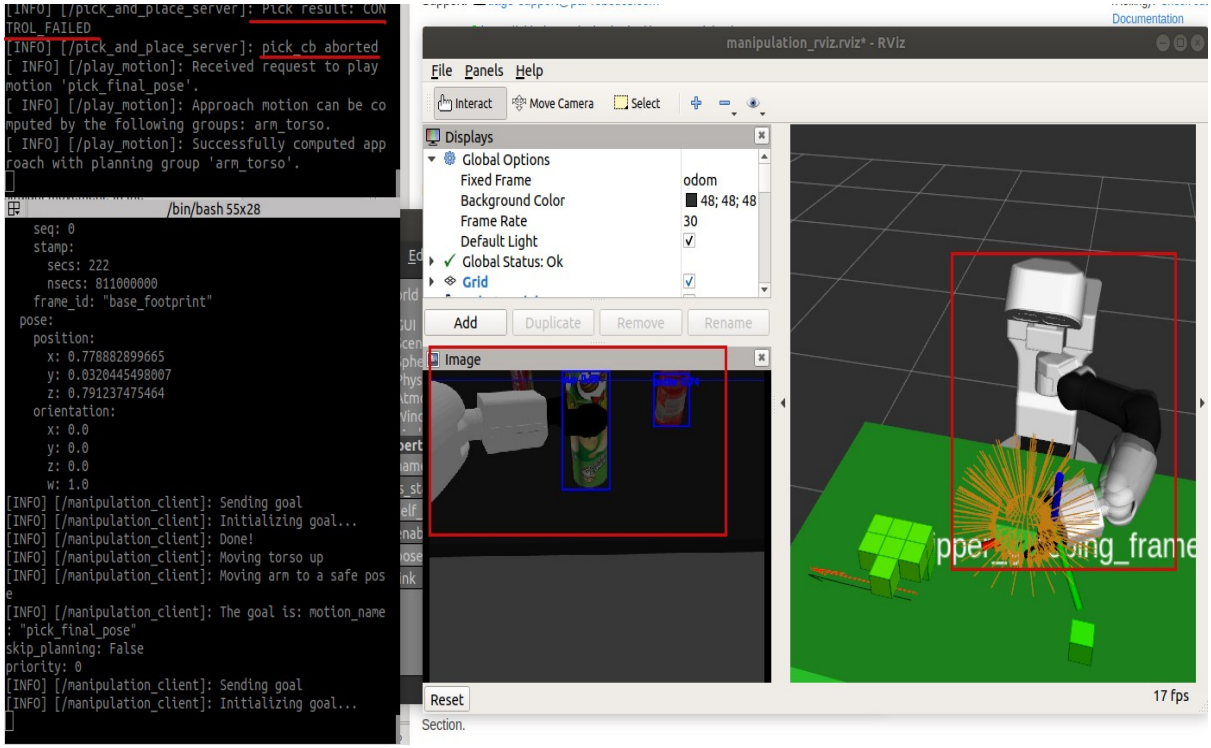

Figure 28: In the left, the terminal prints an underlined error aborting the task. Then, inside the red squares, it is visible that the grasping was actually succesful.

On the other hand, when it fails to grasp and closes the gripper, it detects that the grasp was successful and continues with the task as it is shown in the figure 29, Priority is given to the failure to verify that the grasping has been successful, as the grasping fails sporadically. The task runs faultlessly until it reaches this error that has not been solved.

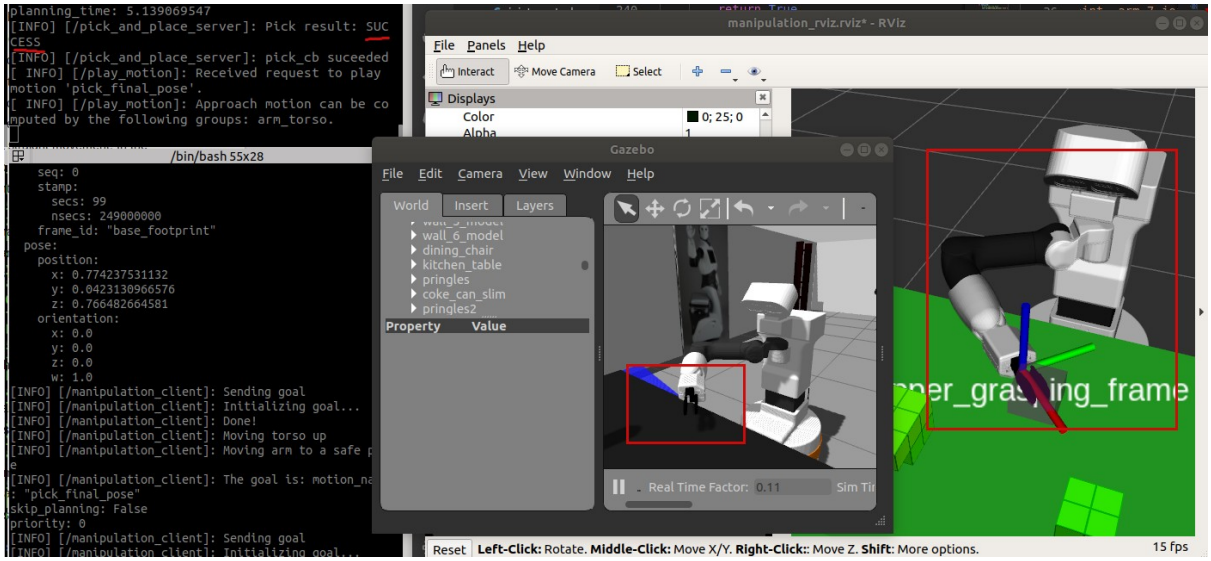

Figure 29: The terminal prints the pick result as succesful. TIAGo picks the virtual object but there is not actually an object as it is shown inside the red squares.

Then, the rest of the tests were carried out by removing the object just before it was grasped to prevent it from aborting the task. This way, the grasping is booming, and the experiment can continue as the virtual object is represented as grasped.

8.2.3 Post-gasp

After grasping the object, it raises the torso and the arm to its limits and holds the object there, showing that it is correctly grasped. This task has no errors considering that there is not an object grasped.

8.2.4 Placing

Lower the torso again to put the object down and place it above the initial position. Then lower the end effector in a rectilinear movement and open

the gripper releasing the object. Despite not having the actual object, it does not show any error in the virtual object.

8.2.5 Post-place

Once the object is released, this phase takes care of moving the end effector away from the workspace without interfering with the object returning to the home pose. Previous tests did not find paths to return to the home pose without colliding with the object. Therefore, several waypoints have been added to the postplace motion to solve this problem. So the task is completed correctly.

9. ENVIRONMENTAL AND SOCIAL IMPACT

The abilities of service robots have gradually exceeded human's performance in specific areas over time. It is necessary to consider some impacts these new technologies bring with them. Ethics is a fundamental field in robotics, so in this chapter, there is a small discussion about environmental and social impact.

9.1. ENVIRONMENTAL IMPACT

Innovation in robotics looks to be linked to energy consumption and other environmental issues. This is an overview of them. Like any other industry, robotics requires energy consumption for production and after it during their performance. In service robots, the performance consumption is lower, and this energy can be gathered using clean sources, so the impact relies mainly on the energy supplier. Also, it is essential to point out that robots work more efficiently than humans, reducing energy waste and materials.

Another issue comes after the life span of the robot. If the robot is not recycled correctly, it will negatively impact waste management.

9.2. SOCIAL IMPACT

There are some ethical questions about how ready society is for robots. This type of service robot will have a substantial socioeconomic level. These robots will work in professional areas like medicine, cleaning, construction, space, etc. In the same way, they will reach our homes with domestic robots which will bring entertainment and household services. There are some topics which concern people about this kind of robot.

- The integration of robots into society depends on several factors, such as their appearance. If a robot looks too much like a human, it can cause a lot of 'creepy' rejection. On the other hand, a friendly-looking robot can facilitate the process of acceptance by humans.

- Aspects like gross losses in employment make people worry about robots. However, this type of robot is intended to support other professionals. In nursing, different tasks can put the nurse in a vulnerable situation in terms of hygiene. Therefore, a collaborative robot such as TIAGo could perform the part of the tasks that could be harmful to humans with the support of a nurse to complement the task.

To conclude, this project might be a very tiny step for developing service robots, and thus, it might contribute to these processes making it an important thing to reflect on. There are many possibilities in the future,

and it is necessary to understand which role robotic engineers must play in it to perform a responsible innovation.

10. CONCLUSION

Different libraries have been implemented to create the final program. The most helpful was the action server from Tiago's pick and place demo used in the test environment. To perform the task, it uses artificial vision to obtain the object's position. The robot functionalities were tested separately, demonstrating that the task can be performed.

The different tasks have been discussed for a correct analysis of the project development.

Following the pipeline, the first is the preparation. This task initialises every node without problems and performs the predefined motions to look at the object correctly. This shows that the implementation of the new motions was successful. This phase activates object detection, in charge of the perception and localisation of objects, which was achieved. Despite exceeding the target, object detection is not very robust, as it can distinguish and position the closest object in the scene but struggles to identify the other two. This can happen for several reasons. One is that the neural network has not been trained with these objects but with objects in the lab. Therefore, when it switches to the simulation because they are different objects, it has trouble identifying them. In addition, colour, distance and size also influence its recognition. Next, TIAGo adds the virtual objects loaded from the pick_and_place.launch file to the scene. Rviz renders them accurately, so this part is also a success. Now it is ready to calculate the possible grab to finish this phase. In previous

tests, sometimes the planner exceeded the calculation time aborting the task. Since the trac-ik solver was implemented, this is no longer a problem.

The next target to be overcome is the grasping phase, starting with the pregrasp pose. It brings the arm into a position relative to the object so it can execute the grappling with a straight movement. In the grappling part, TIAGo grabs the object perfectly, but due to a bug, every time it grabs it, it sends an error message that prevents it from continuing with the pipeline. This is a weird error because the real robot works without problems, but this bug appears in the simulation.

From this point on, the tests were carried out without the actual can, as explained in the results chapter. The post-grip, placement and post-placement phases are error-free, but to be confident in them, it is necessary to test them with the objects.

During the development of this thesis, the concepts of robotics, programming and operating systems have been consolidated. In addition to learning how to use ROS and python programming.

During my laboratory stay, I have seen how a research group works, developing teamwork and self-learning skills.

Finally, the project could have been approached in a different way, as it has been challenging to program some things from scratch instead of relying on GUI that could have facilitated its implementation. This was so time-consuming and prevented the implementation from being completed. The code should have been approached in a modular way with the help of a behaviour tree.

11. LIMITATIONS

There are factors that have influenced the project that has limited its results.

Previous knowledge was required for its realisation, and a large part of the project time was spent on acquiring this knowledge, such as the need to learn ROS and Python. The rest of the team worked telematically, which made communication between us slow.

Firstly, the project had to be implemented on the physical robot, but due to lack of time, I had to return to Spain and adapt the project to the simulation. Also, it would be presented as a master's thesis at Tecnico de Lisboa instead of a bachelor's thesis in Spain, so the project requirements had to be adapted too. I also had to go to Spain on several occasions to take exams in site for subjects that I could not validate during my erasmus.

Regarding the limitations presented by the robot, the gripper was parallel, so it was impossible to grasp the objects optimally. For example, it cannot grasp objects with complicated shapes such as keys. It is arduous to debug the program due to there is not modularity and test every function

12. FUTURE WORK

This thesis opens research problems requiring further consideration to implement the visual servoing correctly. This section lists ideas, proposals or advice for future works of various kinds, whether they are additions, modifications or improvements.

- Wait for the fix of the bug with the pick up action in the grasping phase of the pick-and-place pipeline to verify the complete task.

- Try DOPE in object detection in order to locate the objects accurately. This brings the possibility of training the network efficiently.

- To be able to create a 3D model of the grasping object without knowing the object previously.

- Implementation of real-time functionality for task execution in environments with more complex dynamics, such as pick-and-place of moving objects.

- To generate a task manager modularly with the help of the behaviour tree library. This isolates each task from the rest of the code, making it possible to communicate among all tasks. This way, it is not a problem to add new tasks, and the debugging problem disappears, because when something is not working quite well, it is possible to debug just the task that is not working without affecting the rest of the tasks. Another advantage of being modular, is that the system can be built in a different order, just ordering differently these modules, like black boxes, without having big problems.

References

[1] ISR. Isrobonet@home testbed, 2014. URLhttps://welcome.isr.tecnico.ulisboa.pt/isrobonet/.

[2] B. Singh, N. Sellappan, and P. Kumaradhas. Evolution of industrial robots and their applications.2013.

[3] J. McCarthy, L. Earnest, D. R. Reddy, and P. J. Vicens. A computer with hands, eyes, and ears. In Proceedings of the December 9-11, 1968, fall joint computer conference, part I, pages 329–338,1968.

[4] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. IEEE Transactions on Robotics and Automation, 12(5):651–670, 1996. ISSN 1042296X. Doi: 10.1109/70.538972.

[5] Y. Shirai and H. Inoue. Guiding a robot by visual feedback in assembling tasks. Pattern Recognition,5(2):99 – 108, 1973. ISSN 0031-3203. doi: 10.1016/0031-3203(73)90015-0.URLhttp://www.sciencedirect.com/science/article/pii/0031320373900150.

[6] J. Hill. Real time control of a robot with a mobile camera. 1979
.
[7] F. Chaumette and S. Hutchinson. Visual servo control. I. Basic approaches. IEEE Robotics and
Automation Magazine, 13(4):82–90, 2006. ISSN 10709932. doi: 10.1109/MRA.2006.250573.

[8] Andersson, R.L.: Real time expert system to control a robot ping-pong player (1988)

[9] Allen, P.K., Yoshimi, B., Timcenko, A.: Real-time visual servoing (1990)

[10] Bukowski, R., Haynes, L., Geng, Z., Coleman, N., Santucci, A., Lam, K., Paz, A., May, R., DeVito, M.: Robot hand-eye coordination rapid prototyping environment. In: Proc. ISIR, vol.16 (1991)

[11] Rizzi, A.A., Koditschek, D.E.: Preliminary experiments in spatial robot juggling. In: Experimental Robotics II, pp. 282–298. Springer (1993)

[12] Hager, G.D., Chang, W.C., Morse, A.S.: Robot hand-eye coordination based on stereo vision. IEEE Control. Syst. Mag. 15(1), 30–39 (1995)

[13] P. I. Corke.Visual Control of Robot Manipulators — A Review, pages 1–31.
10.1142/97898145037090001.doi:URL
https://www.worldscientific.com/doi/abs/10.1142/9789814503709_0001.

[14] C. Collewet and E. Marchand. Photometric visual servoing. Robotics, IEEE Transactions on, 27: 828 – 834, 09 2011. doi: 10.1109/TRO.2011.2112593.

[15] Lentin, A. (2020, July 3). What is ROS? *Roboacademy*. URL https://robocademy.com/2020/07/01/what-is-ros/

[16] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: Modelling, Planning and Control. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 1846286417.

[17] S. Haykin. Neural Networks and Learning Machines. Pearson Education, third edition, 2008. ISBN 9780133002553. URL https://books.google.pt/books?id=faouAAAAQBAJ.

[18] [26] S. Haykin. Neural Networks and Learning Machines. Pearson Education, third edition, 2008. ISBN 9780133002553. URL https://books.google.pt/books?id=faouAAAAQBAJ.

[19] S. University. Cs231n convolutional neural networks for visual recognition, 2020. URL https://cs231n.github.io/convolutional-networks

[20] MathWorks. Deep learning - convolutional neural network, 2020. URL https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[21] Redmon, J. (n.d.). *YOLO: Real-Time Object Detection*. Retrieved September 13, 2022, from https://pjreddie.com/darknet/yolo/

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 779–788, 2016

[24]P. Beeson and B. Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), pages 928–935, 2015. doi: 10.1109/HUMANOIDS.2015.7363472.

[25]I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. IEEE Robotics & Automation Magazine, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. https://ompl.kavrakilab.org.

[26]J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics andAutomation. Symposia Proceedings (Cat. No.00CH37065), volume 2, pages 995–1001 vol.2, 2000.

[27] Lopez, V., Moreno, I., & Fernandez, D. (2021, November 18). *Play Motion Builder*. Github. https://github.com/pal-robotics/play_motion_builder

Appendix A: Repository Code

All the software packages developed have been uploaded into the IOC robotics lab repository. The link is presented next.

https://github.com/alain00lpez/akiyabala_tiago

Before starting, make sure that ROS is already installed and tiago's packages for the simulation. Follow the next tutorial:

http://wiki.ros.org/Robots/TIAGo/Tutorials/Installation/InstallUbuntuAndROS

This repository contains the work developed for Alain's thesis. Here is the route to find the files developed:

- /akiyabala_tiago/perception_packages/bayes_objects_tracker/launch/bayes_objects_tracker_no_namespaces.launch
- /akiyabala_tiago/manipulation/launch/pick_place_server.launch
- /akiyabala_tiago/actions_tiago_src/actions_tiago_ros/manipulation.py
- /akiyabala_tiago/actions_tiago_src/actions_tiago_ros/pick.py
- /akiyabala_tiago/actions_tiago_src/actions_tiago_ros/place.py

Please check PAL Robotics documentation:

https://cloud.pal-robotics.com/index.php/s/eocg4B27ITWbsvl?path=%2FTIAGo%20Iron%20(no%20arm)%2FROS%20Melodic#pdfviewer

Appendix B: User Manual

The first step is to install Ubuntu 18.04. Click in the next link to go to the turotial to do it.
https://ubuntu.com/tutorials/install-ubuntu-desktop-1804#1-overview

Now the instalation of ROS Melodic is required.

Follow these instructions:
http://wiki.ros.org/melodic/Installation/Ubuntu

Or use the following commands:

```
sudo apt update
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -
sc) main" > /etc/apt/sources.list.d/ros-latest.list'
sudo apt install curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
sudo apt-key add -
sudo apt update
sudo apt install ros-melodic-desktop-full -y
sudo rosdep init
rosdep update
sudo apt install python-rosinstall -y
source /opt/ros/melodic/setup.bash
```

Terminal 11: Commands for installing ROS Melodic.

Now, follow these command lines to set up SocRob/TIAGo environment.

```
mkdir -p ~/ros_ws/src
cd ~/ros_ws
sudo apt install python-catkin-tools -y
catkin init
catkin build
cd ~/ros_ws/src
git clone https://github.com/alain00lpez/akiyabala_tiago.git
cd akiyabala_tiago/
./repository.debs
source ~/ros_ws/devel/setup.bash
cd  ~/ros_ws/
catkin build -c
```

Terminal 12: Commands for setting up SocRob/TIAGo environment.

Please, follow this tutorial to set up TIAGo drivers and simulation

http://wiki.ros.org/Robots/TIAGo/Tutorials/Installation/InstallUbuntuAndROS

Take into account that your environment is called `ros_ws` and not `tiago_public_ws`.

In case of compilation errors, report them to the older members. Some package may not compile, but it may be outdated packages that you don't need.

In case you get permission denied error when running any node, try *`sudo rosdep fix-permissions`*

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

Grado en Ingeniería Electrónica Industrial y Automática

Curso 2021/2022

TRABAJO DE FIN DE GRADO:

VISUAL SERVOING AND GRASPING OF KNOWN OBJECTS BY A MOBILA MANIPULATOR

DOCUMENTO 2. PLANOS

AUTOR: Alain Kiyabala López
TUTOR: Dr. Enrique Berjano Zanón
COTUTOR EXTERNO: Dr. Pedro Manuel Urbano de Almeida Lima
(Instituto Superior Técnico, Universidade de Lisboa)

1.OBJECT

This project is entirely software based. Therefore, no hardware requiring drawings has been implemented. All the graphical information needed to understand this TFG has been included as figures in the "DOCUMENTO 1. MEMORIA".

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

Grado en Ingeniería Electrónica Industrial y Automática

Curso 2021/2022

TRABAJO DE FIN DE GRADO:

VISUAL SERVOING AND GRASPING OF KNOWN OBJECTS BY A MOBILA MANIPULATOR

DOCUMENTO 3. PRESUPUESTO

AUTOR: Alain Kiyabala López
TUTOR: Dr. Enrique Berjano Zanón
COTUTOR EXTERNO: Dr. Pedro Manuel Urbano de Almeida Lima
(Instituto Superior Técnico, Universidade de Lisboa)

1. OBJECT

The purpose of this document is to show the project's total cost as approximately as possible. The following sections detail the costs necessary for realising the project and its execution, dividing the budget into three distinct blocks: materials, labour, and other expenses. The sum of these costs makes up the cost of executing the material.

1.1 HARDWARE

The hardware cost is only from TIAGo robot as it is shown in table 1.

Table 1: Hardware cost

Hardware			
Units	Description	Manufacturer	Fixed cost [€]
1	TIAGo Robot with parallel gripper	PAL Robotics	35 000,00 €
		Total:	35 000,00 €

1.2 LABOUR COST

There is only an industrial technical engineer in charge of the software development and documentation. This is shown in table 2.

Table 2: Labour costs

Labour cost					
Description	**type**	**number**	**cost/hour**	**Hours**	**cost**
Software development	Industrial technical engineer	1	25,00 €	300	7 500,00 €
				Total:	7 500,00 €

1.3 OTHER EXPENSES

Other expenses are taken into account, such as the electricity consumption of each computer and its depreciation during its use. In this case, there is only one computer used.

Table 3: Other expenses prevision

Other expenses						
Cost factor		**Fixed cost [€]**	**Life Expectancy [h]**	**Variable Cost [€/h]**	**Time refered to The proyect [ħ]**	**Cost Related to The project**
Lab computer	Depreciation	1500	12000	0,13	300	39,00 €
	Electric Consumption			0,14	300	42,00 €
					Total:	81,00 €

1.4 MATERIAL EXECUTION

The execution cost is the sum of the hardware with labour and other costs as it is shown in table 4.

Table 4: Material execution cost

Material execution	
Hardware	35 000,00 €
Labour Cost	7 500,00 €
Other expenses	81,00 €
Total	42 581,00 €

1.5 TOTAL COST

Finally, the budget for the whole project is obtained, applying the taxes corresponding to the IVA and the profit derived from the realisation of the project as it is shown in table 5.

Table 5: Final budget summary

Budget Summary	
TIAGO ROBOT WITH SW IMPLEMENTED	42 581,00 €
EXECUTION MATERIAL	42 581,00 €
13% GENERAL COSTS	42 581,13 €
7% INDUSTRIAL PROFIT	29 806,70 €
PARTIAL SUM	114 968,83 €
21% IVA	24 143,45 €
TOTAL BUDGET	139 112,28 €

The budget for this project is one hundred and thirty-nine thousand and one hundred and twelve euros and twenty-eight cents (139,112.28 €).

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

Grado en Ingeniería Electrónica Industrial y Automática

Curso 2021/2022

TRABAJO DE FIN DE GRADO:

VISUAL SERVOING AND GRASPING OF KNOWN OBJECTS BY A MOBILE MANIPULATOR

DOCUMENTO 4. PLIEGO DE CONDICIONES

AUTOR: Alain Kiyabala López
TUTOR: Dr. Enrique Berjano Zanón
COTUTOR EXTERNO: Dr. Pedro Manuel Urbano de Almeida Lima
(Instituto Superior Técnico, Universidade de Lisboa)

1. OBJECT

This document aims to establish the minimum conditions to be met by the TIAGo robot for the support of people in a situation of dependency, specifying the durability, reliability and safety requirements.

The scope of this document extends to all the computer systems that form part of this project.

2. WORKING MATERIALS

2.1 HARDWARE.

- Laptop ASUS TUF Gaming F17 FX706HM-HX059:
  - Processor: Intel i7-8750H 4.1GHz
  - RAM: 43.9 cm (17.3") Full HD 1920 x 1080 IPS 144Hz sRGB Colour gamut 62.5%
  - Drive:SSD 1TB
  - Graphics card: NVIDIA GeForce RTX 3060 (6GB GDDR6)
  - Display: 1980x1280 144Hz LED display
- Operating system: Ubuntu 18.04

- TIAGo robot:
  - 2 degrees of freedom (DoF) mobile head
  - RGB-D camera
  - 7 DoF robotic arm
  - Parallel gripper as end-effector
  - prismatic joint to raise the torso
  - mobile base which gathers all the necessary elements for navigation.

2.2 SOFTWARE

-ROS: A middleware which is responsible for handling the communication between programs in a distributed system.
-Gazebo simulator: Robot simulation with a complete toolbox of development libraries.
-Reviz: 3D visualisation tool for ROS.
-Visual Studio: IDE used for programming the robot.

3. IMPLEMENTATION CONDITIONS

3.1 STANDARDS

ISO/TS 15066:2016 - Robots and robotic devices specify safety requirements for collaborative industrial robot systems.

A robot is a quasi-machine, according to Directive 2006/42. The manufacturer relies on harmonised standards (EN ISO 10218-1) to ensure compliance with the Machinery Directive.

- The integrator is the one who integrates the robot and other components into a system for a robotic application (specific use). To comply with the Directive, he must perform a risk assessment and protect against residual risks of the resulting machine and robotic application. It can rely on existing harmonised standards or technical specifications (EN ISO 10218-2). Robots with collaborative operation can be based on the TS 15066 specification. Following the risk assessment, safety functions shall be selected for the robotic application to ensure that no contact with the person or no harm is caused. If these measures are insufficient, additional

protective measures (guards, sensing devices, etc.) must be used in a mixed solution. Not all robotic applications can be “pure” collaborative.

- The relevant technical documentation, assembly instructions and declaration of incorporation are drawn up by the robot manufacturer and accompany the robot until it is incorporated into the final machine and forms part of its technical file.

- A competent body can certify that a collaborative robot application complies with the requirements applicable to the workstations.

3.2 DESCRIPTION

As the requirements for the implementation are already implemented in TIAGO, an initial verification test is carried out. For this purpose, a computer is used with the software programmes detailed in the user manual (appended to the report) and the installation of the code on the robot is completed.

Once the system is installed, the correct operation of the robot is verified in a controlled environment. To this end, a series of conditions must be met to evaluate its operation.

3.3 QUALITY CONTROL

For the correct evaluation of the execution of the system and its phases, a staggering procedure is used to verify the robot's operation.

In the integration phase, a specialised technician must ensure that the robot is in an environment that meets the following conditions:

- The working temperatures are between +10°C ~ +35°C.
- The terrain must be capable of supporting the weight of the robot. It must be horizontal and flat. Do not use carpets, as the robot can trip over them.
- Make sure the robot has adequate space for any unexpected operation.
- Make sure the environment is free from objects that could pose a risk if the robot is knocked, hit, or otherwise affected.
- Make sure no cables or ropes are caught in the covers or wheels; these could pull other objects over.
- Make sure no animals are near the robot.
- Be aware of the location of emergency exits and make sure the robot cannot block them.
- Do not operate the robot outdoors.
- Keep the robot away from flames and other sources of heat.
- Do not allow the robot to come into contact with liquids.
- Avoid the use or presence of magnetic devices near the robot.

The technician must then connect his computer to TIAGo and run the test. To do this, the robot must be placed in front of a flat table with a can above. The robot must be at a distance of less than one metre. When the program runs, messages indicating the status of the initialisation connections will be printed on the computer screen, and the pick and place task will start.