IMPLEMENTATION AND EVALUATION OF AN OBSTACLE AVOIDANCE SYSTEM IN A COLLABORATIVE ROBOT

IZAN LEAL GARCÍA

Thesis supervisor: CECILIO ANGULO BAHON (Department of Automatic Control)

Degree: Master's Degree in Artificial Intelligence

Master's thesis

School of Engineering
Universitat Rovira i Virgili (URV)

Faculty of Mathematics
Universitat de Barcelona (UB)

Barcelona School of Informatics (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Abstract

This thesis aims to study and implement an obstacle avoidance system for a collaborative robot application. This application was part of a previous Master's thesis, ***"Implementation and evaluation of movement primitives and a graphical user interface for a collaborative robot"***, developed by Roy Ove Eriksen. Which was developed in ROS to generate controls to follow a demonstrated trajectory with a Universal Robot 3 CB-series.

A camera feedback pipeline has been developed to place the objects of the environment inside a virtual environment. With the objects in the virtual environment, a collision detection system can detect if any object is in a collision course with the robot. If a collision with the end-effector is detected, the system produces an alternative trajectory to avoid the collision using the implemented obstacle avoidance pipeline.

The obstacle avoidance pipeline extends from the original Dynamic Movement Primitives (DMPs) code and uses Artificial Potential Fields (APFs) to generate the avoiding trajectory while following the demonstrated trajectory.

Promising results have been obtained, showing the system has accurate workplace feedback and capabilities to detect collisions and act in consonance to avoid them.

Declaration of authorship

I declare that,

the work in this Master's Thesis is completely my own work,

no part of this Master's Thesis is taken from other people's work without giving them credit,

all references have been clearly cited.

I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by the School of Engineering Universitat Rovira i Virgili (URV) and Faculty of Mathematics Universitat de Barcelona (UB) and Barcelona School of Informatics (FIB) Universitat Politècnica de Catalunya (UPC) - BarcelonaTech.

Izan Leal García		**January 15th 2024**
Student name	Signature	Date

Title of the Thesis: **Implementation and evaluation of an obstacle avoidance system in a collaborative robot**.

Acknowledgements

This project was conducted as a final Master's thesis for the Master's in Artifiacial Intelligence in School of Engineering Universitat Rovira i Virgili (URV) and Faculty of Mathematics Universitat de Barcelona (UB) and Barcelona School of Informatics (FIB) Universitat Politècnica de Catalunya (UPC) - BarcelonaTech.

I would like to thank my friends and family for the support given during this intense year and a half. Especially to Pau, who still had time to hear my digressions despite being in another country.

Also, I would like to especially thank my Master's friends for clearing my mind during the intense code development hours.

Finally, I want to extend a special thanks to Prof. Cecilio Angulo Bahon for guiding me during those moments when I hit a roadblock in making the system work as intended and for his valuable feedback throughout the report creation process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objectives

The primary aim of this thesis is to comprehensively investigate existing obstacle avoidance methods tailored for robotic arms in combination with imitation learning. The goal is to analyze approaches for facilitating obstacle avoidance in a workspace where humans or objects move alongside the robot and implement some of them. Defining the pertinent environment feedback mechanism that coherently integrates with the predetermined workspace setup ensures direct and efficient interaction with the robotic arm.

The following milestones need to be fulfilled to achieve the objective of this thesis:

1. Research on environment feedback mechanisms
2. Development of an environment feedback mechanism
3. Development of a collision check algorithm
4. Research on obstacle avoidance algorithms
5. Development of an obstacle avoidance algorithm

The ultimate goal of this thesis is to extend safety features for the previous work application while keeping its original functionality.

1.2 Framework

The work of this thesis is framed in the ESAII department at Facultat d'Informàtica de Barcelona - Universitat Politècnica de Catalunya (FIB-UPC), which provides the robot and facilities for the development of the study experiments.

It establishes the continuation of a previous Master's thesis work, *Implementation and evaluation of movement primitives and a graphical user interface for a collaborative robot* [27], developed by Roy Ove Eriksen. A detailed explanation of the extension of the previous work will be detailed in the background section.

1.3 Resources

In this section, all the available resources used for this work are mentioned and detailed to set the basis for the implementation and the experimentation.

1.3.1 Robot

The available robot to carry out the experiments for the study is the UR3, the previous version of the UR3e robot [46]. This robot is a small collaborative robot from the UR Robot family Figure 1.1, capable of performing tasks involving precise and complex movements in an environment alongside humans. Additionally, the robot has installed the 2019 version of the RG2 gripper tool [60] in the tip, Figure 1.2, but gripper movements will not be used specifically in this work.

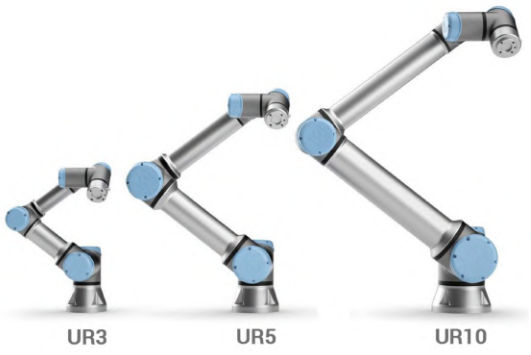


Figure 1.1: The UR Robot family [45]

Figure 1.2: RG2 gripper[70]

Characteristics

The UR3 robot has the following technical specifications, [16]:

- Degrees of Freedom: 6 rotating joints
- Max. payload: $3Kg$
- Joint limits: End joint ∞ rotation, other joints ± 360 deg
- Joint velocity: wrist joints 360 deg $/s$, other joints 180 deg $/s$
- Max endpoint velocity: Typical $1m/s$
- Power consumption: $100W$
- Repeatability error: $\pm 0.1mm$
- Communication: TCP/IP $100Mbit$ Ethernet

The robot is divided into different rigid bodies that are attached to joints. Joints act as hinges, allowing one body to rotate in a particular axis over another. Each of these bodies has a pose relative to the body they are attached to, translating into a particular pose in the world. These poses can be referred to as links, Figure 1.3 and play a key role in converting between joint and Cartesian space.

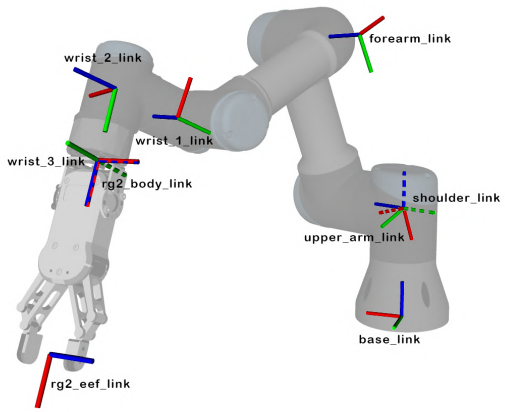


Figure 1.3: Robot rigid bodies links.

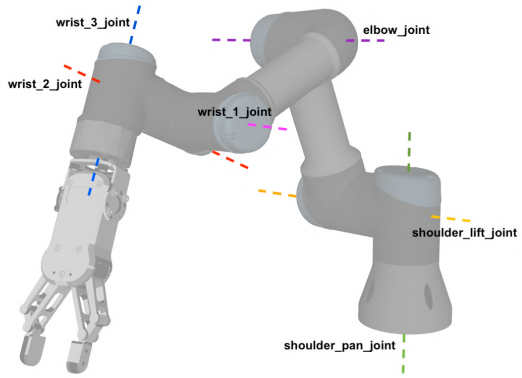


Figure 1.4: Robot rigid bodies joints.

The six moving joints are shown in Figure 1.4, whose angle will determine the pose of each link in the reference robot frame, *base_link*, that is, the base of the robot solid pose.

1.3.2 Camera

After the study of the feedback sensor requirements, detailed later in the report, it has been concluded that the alternative to use is the Intel RealSense Depth Camera D435 [58], Figure 1.5. The camera has a depth stereoscopic module, which uses structured light to generate two depth maps that can be used to generate a 3D point cloud (similar to a LiDAR). Moreover, it can also be used as a standard RGB camera.

Figure 1.5: Intel RealSense Depth Camera D435 [58].

Characteristics

The Intel RealSense D435 has the following technical specifications, [59]:

- Depth output resolution: Up to 1280×720
- Depth Field of View (FoV) $[H \times V]$: $87° \times 58°$
- Depth frame rate: Up to $90fps$
- Range: from $0.3 - 3m$
- RGB frame resolution: Up to 1920×1080
- RGB sensor FoV $[H \times V]$: $69° \times 42°$
- RGB frame rate: Up to $30fps$
- Global Shutter

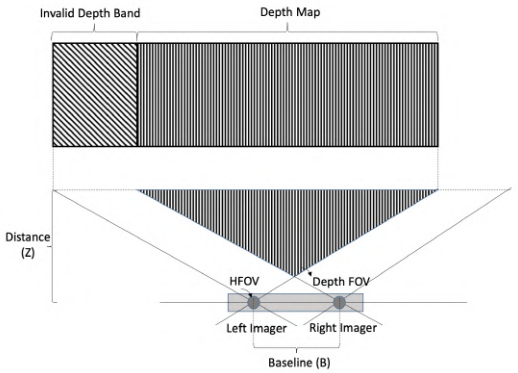


Figure 1.6: Depth module FoV [59].

It has to be taken into account that the effective depth FoV is the overlapping region between the FoVs of the two cameras of the depth module, as seen in Figure 1.6.

Despite having the limitation of 30Hz, the camera is still faster than the LiDAR alternative. Additionally, the working range is perfect for the size of the workspace, and the FoV and resolution cover it entirely with the required precision. The global shutter is also a good feature for capturing moving objects, as it captures all the pixels of the image simultaneously and not one by one, which usually causes deformations with fast-moving objects.

The point cloud resolution can be computed using the field of view, the range, and the depth pixel resolution with Equation 1.1. In the worst case scenario of $range = 3m$, depth resolution of 1280×720 and FoV of $87° \times 58°$, the cloud resolution is $\sim 0.0045m$ horizontally and vertically.

$$cloud\_resolution = \frac{2 \cdot \tan\left(\frac{FoV}{2}\right) \cdot range}{depth\_resolution} \tag{1.1}$$

1.3.3 Workspace setup

The laboratory robot setup consists of a $1 \times 1m$ table used as the UR3 robot base and a 3-color lamp that can be programmed alongside the robot Figure 1.7. The UR3 robot comes with a teach pendant that will be used to initialize the robot and configure the communication with the PC.

In the definition of the camera setup, the workspace mounting alternatives are constrained by the robot setup. The robot is only fixed to the base, and the base is not fixed to the table where it sits. Making the option of mounting the camera anywhere that is not fixed to this base unfeasible, as the position of the camera needs to be known precisely to make the volume reconstructions, and any movement that is not taken into account between the robot (base) and the camera will generate positioning error.

Mounting on the robot has also been considered, as the robot is fixed to the base, and every point of the robot can be converted to the base reference frame at any point during robot movement. However, this alternative has been discarded during the study of the alternatives.

To ensure that the camera always has the same relative position to the base and has sufficient distance to cover the whole workspace with the depth effective FoV, a custom mount has been created Figure 1.8, which can be adjusted in height and pitch.

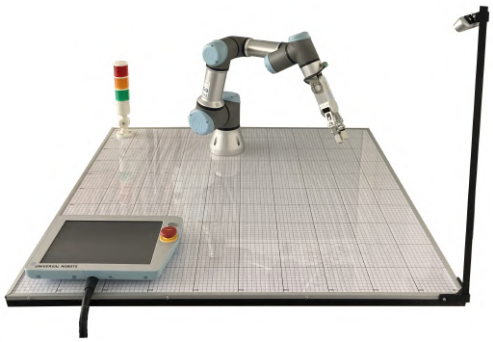


Figure 1.7: Workspace setup.

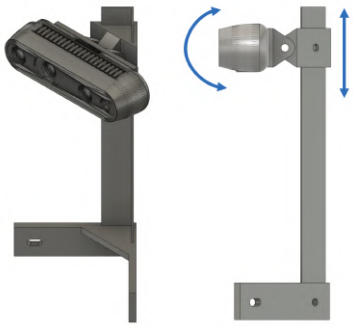


Figure 1.8: Camera mount.

An ethernet connection between the robot and the controlling computer is required. The UR3 robot is connected to the laboratory's Ethernet local network with a fixed IP. Thus, any PC with access to this network can be used.

The available PC for the experimentation is an HP ProDesk 400 G3 3.2GHz Micro Tower with an Intel Core i5-6500 and 16GB of RAM.

1.3.4 Programming environment

Robot Operating System [61] (ROS) library has been used to develop the work of this thesis. ROS is an open-source library set that brings tools to develop robotic applications. Multiple versions are available, each one specific to a particular Ubuntu distribution. The system will consist of Ubuntu 20.04 alongside with ROS Noetic [62].

ROS is very useful for complex robotic applications because it provides a modular system with standardized communication tools to join each part of the system and achieve a unified application. To achieve this modularity while providing cohesion, it uses a manager called ROS Mater to track the nodes (executing code), the communication pipelines between nodes, and the messages sent. Figure 1.9 shows a simple communication diagram between two nodes.

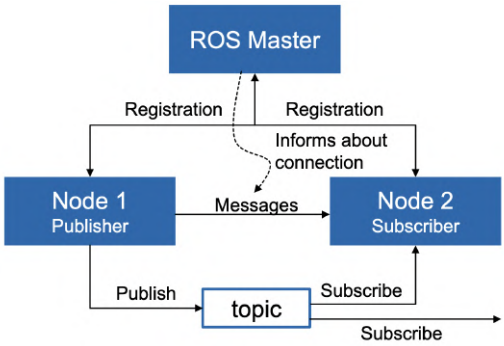


Figure 1.9: ROS communication diagram [18].

The ROS nodes can be created using multiple programming languages, such as Python, C++, and Matlab, among others. They are inside ROS packages, which can be downloaded or created from scratch. Additionally, these nodes can be grouped in namespaces (for example, in a robot package, the camera nodes can be grouped under the camera namespace). Packages are then contained inside the workspace; this workspace can be the one from ROS itself (located at /opt/ros/noetic/) or the one created by the user with catkin (this catkin workspace overlays on top of the ROS one). An overview of the workspace is shown in Figure 1.10.

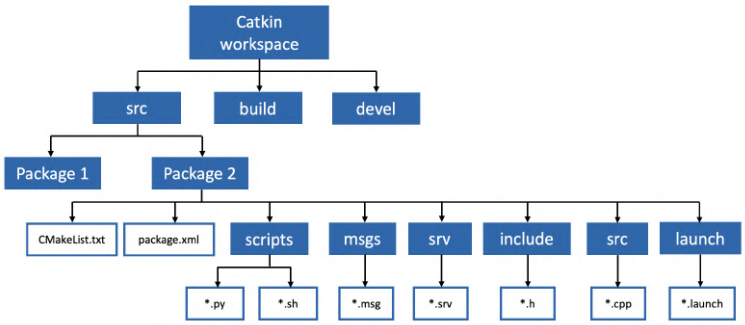


Figure 1.10: ROS workspace structure [26].

ROS also comes with simulation and visualization tools, such as Gazebo [57], a toolbox for simulation that can communicate directly with ROS to enable the development of robotic applications easily. It installs alongside ROS and can simulate simple moving robots and complex robots with sensors. For the visualization part, the RViz [63] application can be used, which makes the debugging and data analysis easier.

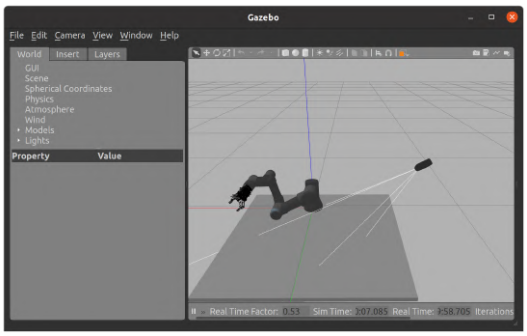


Figure 1.11: Gazebo GUI.

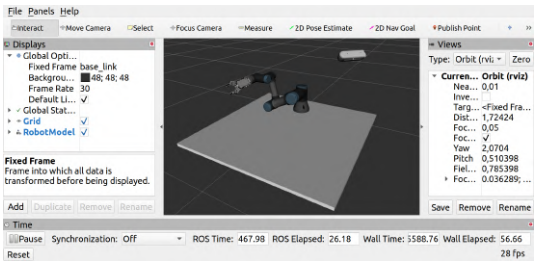


Figure 1.12: RViz GUI.

Apart from having the physical robot setup, there are some packages that need to be added to the workspace to enable the communication with the robot, such as Universal_Robots_ROS_Driver and universal_robot. The packages not only allow to communicate with the robot, but they also enable the simulation of the robot in Gazebo.

It is important to note that the Universal Robot ROS driver packages require installing an *urcap* plugin in the robot (using the Teach Pendant), following the robot preparation instructions.

The universal robot package relies on MoveIt [67] to make the control of the robot easier. In addition to control, this package also covers motion planning, manipulation, 3D perception, kinematics, navigation and collision checking. Being the latter crucial for development of this work.

The chosen camera has a proprietary ROS package (realsense-ros), that connects to the camera and passes the information to ROS topics. Moreover, this package can generate a point cloud using the depth images collected from the camera. Color and points alignment is also available, but not necessary for volume reconstruction.

A complementary ROS package (realsense_gazebo_plugin) has been implemented to enable the camera simulation using a Gazebo plugin. This allows to fully test the system without the real hardware. Additional packages will be added and detailed during the implementation phase.

The workspace setup detailed in the previous section has been replicated in the Gazebo simulation environment, to enable realistic simulations.

1.4 Problem description

This thesis is fundamentally concerned with a pivotal issue in the realm of robotics, the establishment of safe human-robot integration within the industrial landscape. The core challenge lies in orchestrating a harmonious collaboration between robots and humans, where each entity undertakes complementary tasks without instigating hazardous situations or impeding each other's actions.

The first problem to be addressed is the need for workspace environment feedback. To enable the robot's obstacle avoidance capabilities, first, the obstacles need to be mapped and introduced to the planning pipeline. This will be solved by using one or more sensors that convert from a physical magnitude to digital data.

Once the system has feedback from the workspace, the second problem to be addresed, is the detection of collisions with the robot and the workspace objects. As the robot is intended to follow a demonstrated path, and workspace obstacles can move to a position where the this path is obstructed. The robot needs to be able to detect these occlusions to trigger the avoidance plan generation. this is important because recomputing the path is a time-consuming task, and it won't be efficient to recompute the path with the updated workspace data at every time-step.

Finally, when the collision is detected, the robot must change its current plan to a new one where the workspace obstacle is avoided. The problem to be solved now is the generation of a path that is as similar as it could be to the original path (to keep the system imitation capabilities) and that leads the robot tool away from the obstacle to avoid the collision.

1.5 Background

This research serves as a natural progression from a prior master thesis (*Implementation and evaluation of movement primitives and a graphical user interface for a collaborative robot* [27]). The earlier work laid the foundation for a user-friendly control framework for the robot, eliminating the need for intricate programming skills.

This framework allows the user to record a specific movement with the "free drive" mode (a mode where the robot only compensates for the gravitational force), to generate a movement demonstration, and then desired start and end point can be changed to perform the movement task at different points of the worspace. With these parameters, the robot is able to move from start to end, following a trajectory similar to the one recorded by the user. This is possibles because the work uses the controller of Dynamic Movement Primitives (DMPs), which can perform trajectory imitation easily.

The existing interface boasts a commendable collision check mechanism integrated within it, ensuring a safety net before initiating any movements. However, as part of continuous improvement, it becomes evident that certain enhancements are imperative to elevate the functionality of the robotic system.

The identified gaps in the system develop in the previous work, set the stage for this thesis's objectives. The absence of real environment feedback necessitates a manual update of obstacles, posing a limitation to the system's adaptability. The aspiration to introduce real-time collision checks underscores the commitment to advancing the system's responsiveness and agility. Furthermore, the envisioned integration of a robot response mechanism to collision detection seeks to propel the collaborative robot from mere precaution to proactive adaptation in dynamic environments.

In essence, this research endeavors to not only address the existing limitations, but also to propel the collaborative robot into a realm of heightened autonomy and real-time adaptability. The goal is to seamlessly integrate environmental feedback into the control framework, transforming the robot from a pre-programmed entity to a dynamically responsive collaborator capable of navigating complex scenarios autonomously.

1.6 Scope

This Master's Thesis will develop an obstacle avoidance system to be used on a Universal Robots UR3, UR5, UR10 CB-edition in continuation to the mentioned prior master thesis. Due to robot availability, only the UR3 will be tested in real life.

The Intel Realsense 435 depth camera will be used to provide workspace feedback to the systems to extend the safety measures. As it is very versatile, and it could be also used to simulate other sensors by processing its output.

The proposed solutions will be specially designed to work with UR robots, but workspace feedback and the obstacle avoidance path generation will be implements as independent nodes that could be used for other robot models.

The thesis will be started with the implementation of the workspace feedback, and then the avoidance system will be constructed by incrementally adding features to the system, this features will need to keep the computation time as low as possible as the system needs to avoid moving obstacles (this path generation time needs to be less than the workspace refresh time to ensure real-time response, as detailed in chapter 2). The first will be to detect movement in the workspace, then detect collision and last generate an avoiding path. Finally, if extra time remains, the avoiding path algorithm will be modified to add robustness. Only one avoidance algorithm will be tested, as all the previous steps to reach this final implementation, are time-consuming and have their own challenges. However, features from other

algorithms may be tested if enough time is available.

Only one feedback generation algorithm will be implemented and test because the focus of this work is on developing an obstacle avoidance system capable of following a demonstrated path. This feedback needs to ensure that the robot is able to detect a moving movement before colliding and stopping movement before colliding, making the collision detection algorithm real-time (real-time will be further defined in chapter 2). In addition, only the collisions with the end-effector are considered for the avoidance task, because implementing a full robot avoidance system with learning from demonstration capabilities will require more than the available time.

1.7 State of the art

The technological landscape has undergone a remarkable transformation, catalyzing the integration of robots into industries at an unprecedented pace. The historical segregation between robots and humans, rooted in safety concerns, is now being dismantled thanks to advancements in sensing devices and autonomous decision-making processes.

In this era of technological renaissance, the emergence of collaborative robots, or co-bots, marks a paradigm shift in industrial dynamics. Unlike their predecessors, these co-bots are designed to operate seamlessly alongside humans, fostering a collaborative environment where safety takes precedence. The pivotal catalysts for this transformation are the sophisticated sensors embedded within these robots.

These sensors serve as the eyes and ears of the co-bots, generating real-time environment feedback. This feedback becomes the cornerstone for unsupervised decision-making processes, enabling the robot to adapt its actions dynamically. The implications are profound, especially in tasks where human intervention is indispensable. The once-isolated robots are now evolving into responsive collaborators, capable of working in shared spaces without compromising human safety.

Key features, such as safe stop procedures, collision sensors, and, notably, obstacle avoidance mechanisms, define the safety architecture of these co-bots. The ability to navigate through the workspace, recalibrating paths in real-time to avert collisions, represents a pinnacle achievement. This ensures the physical well-being of humans sharing the space and amplifies the efficiency and fluidity of collaborative workflows. In essence, the era of co-bots signifies a harmonious convergence of technology and human-centric design. Integrating advanced sensing technologies and safety features propels robotics into a realm where collaboration is not just feasible but inherently safe and productive. As we witness the rise of co-bots, the vision of a shared workspace, where humans and robots complement each other seamlessly, is rapidly becoming a reality.

1.7.1 Environment feedback

The environment feedback used in industrial robot applications covers a wide range of sensors and functionalities. However, for this work, the attention will be focused on the particular environment feedback for obstacle avoidance applications.

The most basic, yet very effective, sensors used in feedback are the proximity sensors [29]. These sensors detect the presence of an object when it reaches a certain distance threshold Figure 1.13, using different types of technologies [42]:

- Induction: uses Eddie currents to detect metallic objects.
- Capacitance: uses electric field to detect objects.
- Ultrasounds: emits ultrasonic waves to detect objects.
- Magnetic: detects magnetic attraction between sensor and magnetic targets.
- Optics: uses infrared light reflection to detect objects.

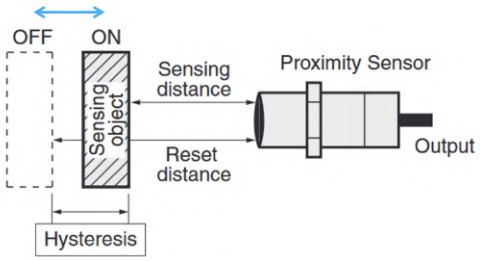


Figure 1.13: Proximity sensor diagram [42]

Typically, multiple proximity sensors are installed on the robot at strategical points, as they are relatively cheap compared to other more complex sensors. However, this setup can sometimes fail to detect small objects, as the sensor have a very narrow detection area. Additionally, they usually have a limited range of detection, thus the robot will be limited a lot when it comes to the available distance for reacting and recomputing the trajectory. Finally, some of these sensors are only suitable for metallic/magnetic targets, which won't be suitable for human detection.

One step ahead of the proximity sensors, is the LiDAR, an optical ranging sensor that can cover a greater area and range of sensing. LiDARs usually work by scanning a specific height (layer) with a rotary laser and receptor, computing the time that takes the laser beam to bounce back from the object to the sensor, but there are also solid state LiDARs that use other methods to generate the scan without rotation [37]. LiDARs can have different characteristics, such as having one or multiple layers, have a specific scanning resolution, the data reading frequency, among others. It is noticeable that the operational frequencies of most commercial LiDARs tend to be low $\sim 10 - 20Hz$ as they scan the space point by point, which is not a fast method to gather the whole scanning range.

LiDAR's data is a cloud of points in the space with their respective return intensity (XYZI), which can be then used to reconstruct the volume of the object [33]. To have an accurate reconstruction of the environment and real-time response, a great resolution with many layers and a fast scan frequency is seek. Sadly, these requirements can lead to a drastic increase in the price of the sensor.

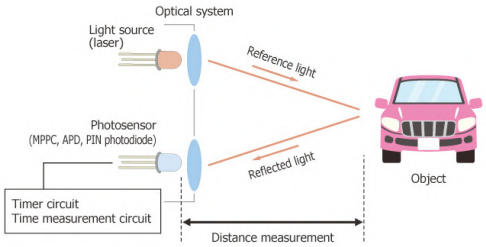


Figure 1.14: Lidar technology [21].

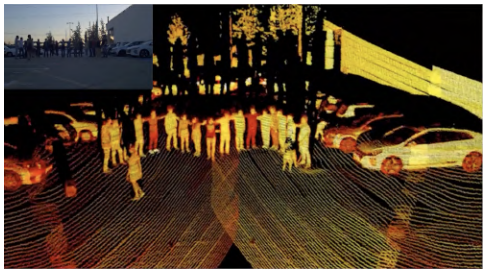


Figure 1.15: Lidar output [31].

A cheaper alternative to a LiDAR can be a camera [47]. Although standard cameras are sensors that can only produce 2D reconstruction of the environment (images), by using multiple cameras configurations and algorithms, a depth map can be generated or even a 3D point cloud similarly to a LiDAR. However, the spatial reconstructions are most of the time less precise than LiDAR reconstructions, but it can be an affordable loss as cameras are cheaper than a LiDAR.

One method to generate the feedback is detecting pose key-points inside an image and using the camera calibration matrix to make a 3D reconstruction of the operator (human) [40], or maybe detect a specific object a project the pose in 2D to 3D. Moreover, to ease the task of detecting objects, markers can be used [43].

There are in the market some cameras that directly offer the capability of outputting depth maps or 3D clouds. With the 3D cloud, the volume of the object can be updated in the digital environment model.

These depth cameras can generate the RGB-D maps by using an integrated processing unit using the following technologies [34] (Figure 1.16):

- Passive stereo: Two RGB cameras in a fixed pose with some separation.
- Structured light: An infrared projector that emits certain patterns, which are later captured with a camera and processed to estimate depth.
- Time of Flight: A projector that emits an IR light beam which is then captured back with a camera, and distance is estimated using the elapsed time between light departure and arrival.
- Neural network depth estimation using single RGB camera [35].

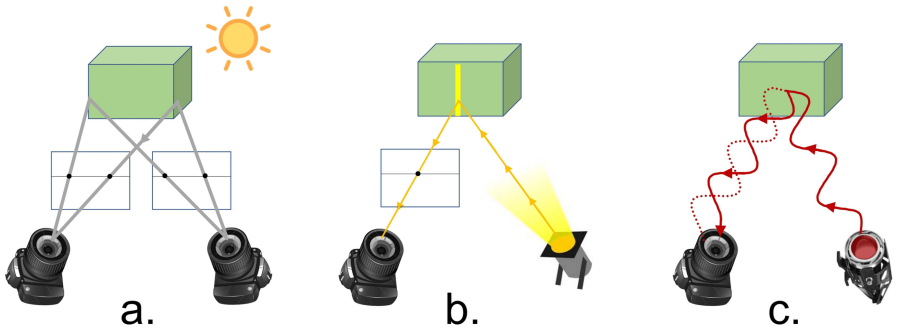


Figure 1.16: (a) Passive stereo. (b) Structured light. (c) Time of Flight. [34].

Finally, there is one more sensor that can be used to generate environment feedback, which is the Inertia Measurement Unit (IMU) [49]. The IMU is a sensor that reports the forces that are applied to it, thus if the sensor is attached to an object, the forces applied to the object are the same as the object. These forces are then translated into velocities, which can be used to determine an object orientation and where it is moving. With some code, the IMU data can be adapted to a position inside the workspace, however the object dimensions need to be known beforehand [24]. Despite being very precise, if the object has moving parts, for example moving arms, additional IMUs will need to be used for each moving part, this makes the implementation of IMUs for human detection more difficult, as they are not rigid bodies, increasing the cost of the system.

1.7.2 Robot arm obstacle avoidance

In the robot industry, one of the key aspects is the path planning step. This step is in charge of generating a feasible trajectory for the robot to follow, allowing the robot to complete the designated task. There are plenty of path planing algorithms available to use, but each one has specif characteristics, thus it is important to choose an algorithm that fits your particular application. The path planning algorithm that is pursued in this thesis, needs to be fast, as it is intended to run in real-time, and it needs to have the ability to handle path generation in dynamic environments for high dimensional dynamics robot. In addition, it also needs to be able to learn from demonstration, but as it will be further commented later, multiple algorithms can be combined their advantages and overcome their limitations.

For many years, robots have been unable to perform well in dynamic industrial environments, however in recent years, with the rapid increase of technological advances, different solutions have been developed. These solutions adapt to the dynamic environments with the use of obstacle avoidance algorithms integrated alongside with the path planning pipeline.

Some of the classical obstacle avoidance path planning approaches are not very useful in dynamic environments, such as Dijkstra algorithm (DA), artificial potential field (APF), probabilistic road map (PRM), Rapidly-exploring Random Trees (RRT), cell decomposition (CD) and Dynamic Movement Primitives (DMP). However, they could be modified to obtain better performance in dynamic environments. There are other more heuristic approaches, that can handle both static and dynamic environments, such as fuzzy logic (FL), neutral networks (NN), particle swarm optimization (PSO), genetic algorithm (GA), cuckoo search algorithm (CSO), and artificial bee colony (ABC) and Reinforcement learning (RL), among others [36] [38]. There are other approaches where path planning algorithms are combined, one without obstacle avoidance capabilities is used to general the global path, and then this path is fine-tuned locally with the use of obstacle avoidance algorithms to ensure that the system can handle the changes in the environment.

Among the heuristic approaches, there are more advanced methods that make use of Neural Networks and Reinforcement Learning to solve the path planning task problem. Some of these algorithms include Soft Actor-Critic (SAC), Asynchronous Advantage Actor Critic (A3C), Deep Deterministic Policy Gradient (DDPG), Dirichlet Process Gaussian Mixture Model (DPGMM), Q-learning and PI2 (Policy Improvement with Path Integrals).

These algorithms learn to create paths using a methodology where the interaction of the robot with the environment is generated and evaluated using rewards. The reward tells the robot if the interaction (actions taken) with the environment is good or bad for achieving the target goal. For this case of study, a positive reward could be reducing the distance between the robot and the target, or avoiding a collision with an obstacle by increasing the distance with the obstacle.

Unfortunately, sometimes the definition of the reward functions in Reinforcement Learning can be very tedious or impractical. To solve this, Imitation Reinforcement learning comes into play, this method uses demonstrations of the desired behavior to teach the robot to generate the path. However, the learning from demonstration needs multiple examples, and can take some time, which is impractical for the work developed in this thesis.

Some approaches, both classical and heuristic, have been studied for this work.

Artificial potential fields

Artificial potential fields (APFs) [51] is one of the simplest techniques used in obstacle avoidance planning for robot arm manipulators. This technique emulates the attractive and repulsive forces of an electrical potential field, where the obstacles generate repulsive forces and the target attractive ones. By tuning the parameter of the algorithm an optimal non-colliding trajectory can be generated, however this algorithm has some drawbacks, as it can not handle dynamic environments well.

But with some modifications, potential fields can vary through time, accounting for the dynamics of the objects in the environment. This approach was presented alongside with the initial artificial potential fields path planning proposal [1], Figure 1.17. Another drawback is when the robot falls int a local minim, where the attractive a repulsive forces neglect. Luckily, there are some adaptations that generate virtual obstacles in this local minima to force the robot to continue moving [41].

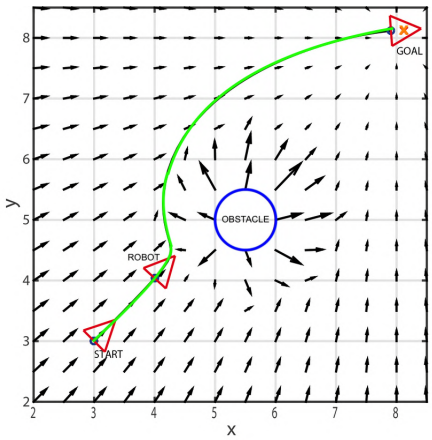


Figure 1.17: Traditional APF path planning [19].

Despite that APFs do not learn from experience, which was the main objective of the path planning for this work, they can be adapted by instead of having only a single target, having multiple targets alongside the demonstrated path.

Another work have incorporated obstacle trajectory predictors, to ensure a good performance in highly dynamic environments, where objects may move towards the robot, accounting for the future pose of the obstacles to generate the path [12]. They use a Recursive Motion Function (RMF) [5], which create a history of the trajectory of the object and fits to a particular movement type. These movements can be from a simple first-order polynomial to an ellipsis or a sinuous.

Finally, there is another approach [29] that addresses the inefficiency of avoiding obstacles close to target or dynamic obstacles by using artificial potential fields in combinations with Reinforcement Learning. The work defines Distance Reinforcement Factors (DRF) and Force Reinforcement Factors (FRF) to implement in a reward function of a Markov Decision Process (MDP) [11]. With this reward function terms, the robot can learn tho avoid obstacles when approached at a certain distance and correct movement when a collision happens, respectively.

Dynamic Movement Primitives (DMP)

One of the best alternatives that fits the task of generating a path from a demonstration without requiring a lot of time and adjustments is the Dynamic Movement Primitives (DMP) [8], which has been the base algorithm used in the previous work. This algorithm relies on the decomposition of the trajectories in simpler segments

(primitives), that allow making computations in a lower dimensional space where convergence and stability is ensured, and a stable non-linear point attraction system is used to force the movement to match the provided path example.

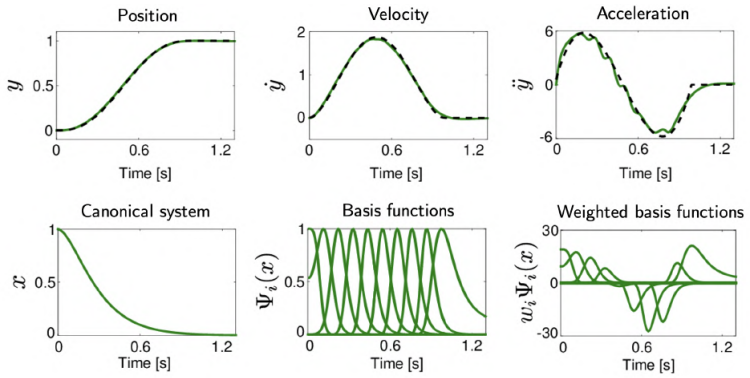


Figure 1.18: Traditional DMP path planning [30]. Example data (black dashed lines). Output (green lines).

The mathematics behind the DMP were stated by Stefan Schaal [8]. and then yeas later revised by Auke Ijspeert [13]. The latter are the ones considered for this work. The DMP algorithm start with the definition of the main dynamic, which is the one related to a damped spring model, and then the dynamic is forced to match the dynamic of a demonstration by using a forcing factor.

Despite performing well when it comes to trajectory approximation, this algorithm still needs some modifications in order to handle obstacle avoidance. Luckily, there are some works that figured out workarounds to solve this problem.

Three of these works solve this problem by directly adding a coupling term alongside the forcing term to modify the dynamics of the system in such a way that the trajectory moves away from the object while trying to follow the demonstrated path. The first approach [9] uses human behavioral dynamics of obstacle avoidance to generate the coupling term. In the work, additional modifications have also been included to improve other weakness of the DMP. The second approach, [20], defines this coupling therm as an artificial potential field. The last of these approaches, [28], implements further modifications to the second approach, where the potential filed parameters and DMP shape parameters are updated with Policy Improvement with Path Integrals (PI2) [10] method. The goal of applying PI2 is to find the policy (state-action pairs) that minimizes the cost function.

Another approach, [14], joins DMP with Model Predictive Control. They learn movement primitives by fitting the parameters of dynamical the systems. First, dynamic systems are learned from the demonstration, and then they are incorporated into the MPC to generate the optimized control commands that make the robot follow the demonstrated path. Additionally, to handle the obstacle avoidance task, spatial and temporal polyhedral constraints are used.

Actor-Critic

Nowadays, one of the most used Reinforcement Learning methods is the Actor-Critic [3], this method consists of two Neural Network (NN) models (or two parallel terminations of the same head network) Figure 1.20, the actor and the critic. The actor decides which action should be taken, and the critic inform the actor how good was the action. Thus, the actor generates policies π (action-state (a, s) pairs) and the critic generates values $V(s)$ (expected cumulative rewards) or Q-values $Q(s, a)$ (expected cumulative reward obtain by taking a particular action a in a specific state s following a certain policy π) or Advantages $A(s, a)$ (the difference between the Q-value $Q(s, a)$ and state value $V(s)$).

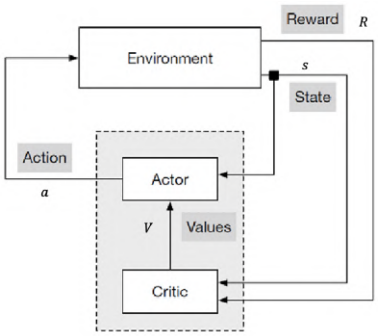


Figure 1.19: Actor-critic architecture [7].

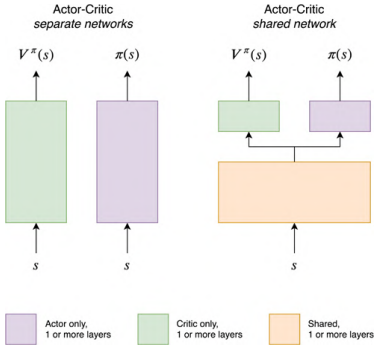


Figure 1.20: Actor/critic NN architecture [22].

The approach detailed in [39], which uses Proximal Policy Optimization with Advantage Actor-Critic, sets the neural network system input as the sensor's input, alongside the goal position and the current pose. The pose and orientation of the end-effector define the output actions. These actions are fed into a Damped Least Squares Inverse Kinematics solver [2]. It is also important to note that in this proposal, the system is only capable of avoiding obstacles that will collide with the end-effector.

Another approach [33], that uses Soft Actor-Critic with Prioritized Experience Replay, solves the collision avoiding limitation by changing the input to the obstacles' closest collision point 3D coordinates, the joint 3D coordinates and angular velocities, and the target 3D coordinates. Additionally, this method integrates the task of generating the movement command (previously computed by the IK solver) inside the actor output, making each action the rotation angular velocity of the joints.

Unfortunately, neither of these approaches is intended to generate a trajectory that follows a demonstration. Despite not having this capability to learn from demonstration, this approach can be adapted to take the demonstrated path as input.

Q-learning

Q-learning is another Reinforcement Learning off-policy technique that focuses on optimizing the robot's cumulative reward in the environment, Figure 1.21. There are two methods: one that uses tables to store Q-values (expected cumulative reward obtained by taking a particular action a in a specific state s following a particular policy π) and the other method that substitutes tables by Neural Networks, for better handling a large number of states and actions, called Deep Q-Learning.

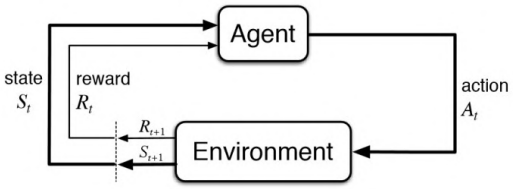


Figure 1.21: Q-Learning architecture [22].

One of the approaches found, [32], uses a gridified workspace as state input of a Q-learning algorithm. These state cells from the grid can be of type start, target, or obstacle. The system's output is a set of actions representing movements in the grid (up, down, left, right, forward, backward), encoding the path to follow. Then, this path, alongside the current (X, Y, Z) state cell, is converted to a sequence of joint angles. This conversion can be done with IK, as mentioned in the previous section, or, as the proposed work details, using a Neural Network.

Artificial Neural Networks

Artificial Neural Network (ANN or NN) [48] models are widely used in different applications due to their versatility and problem-solving capabilities. These systems are based on mathematic models that try to mimic the workings of an organic brain, which consists of neurons that activate when a certain condition is met. The fundamentals of these artificial neurons rely on the linear regression function and an activation function, which is a function that computes the output of the neuron based on the result of the regression. This fundamental neuron, called Perceptron, Figure 1.22, can solve linearly separable problems. Moreover, with the use of non-linear activation functions and concatenating multiple neurons, Figure 1.23, the approximation can virtually take any shape, allowing the system to solve non-linearly separable problems, which makes neural networks universal approximations.

In theory, a complex enough neural network, with the appropriate input, can generate the desired output after some training is carried out. This capability can also be useful for the problem tackled in this thesis, as given the robot state, obstacle position, and demonstration trajectory, the system could generate the obstacle-avoiding path.

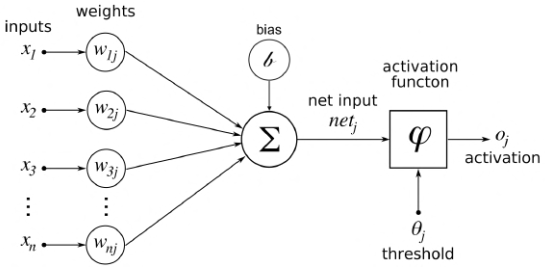


Figure 1.22: Artificial neuron [6].

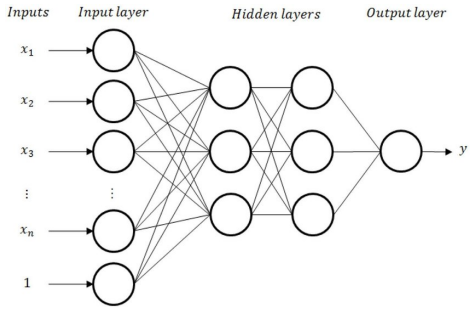


Figure 1.23: Artificial Neural Network [17].

Work [47] achieves what has been previously commented on by using a neural network called Collaborative Waypoint Planning network (CWP-net). However, the input generation is a complex method, as neural networks tend to have fixed input sizes, and a workspace can have infinite states. For this reason, the paper uses a Dirichlet Process Gaussian Mixture Model (DPGMM) [25] to generate the distributions from the angles of each joint pair (links), encoding the infinite states of the robot into a finite set of distribution parameters into submodels, then a set of $N_{k_1} \times N_{k_2} \times \ldots \times N_{k_m}$ submodel combinations are defined as states of the robot (being N_k each link's submodels, and m the total number of links), allowing to have a single numerical input for the state. The DPGMM model is trained with multiple examples of the robot positioned in the workspace.

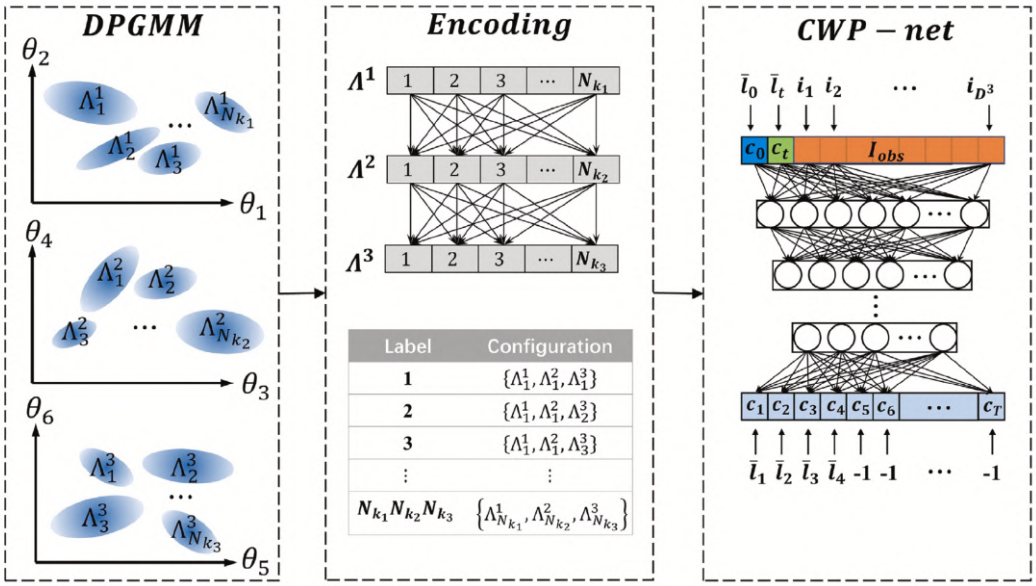


Figure 1.24: CWP-net arquitecture [47].

Even though this approach lacks the learning demonstration part, it could be added by using, for example, a DMP for trajectory generation without collision avoidance and then feeding the network X consecutive trajectory steps as targets to generate collision-free trajectory. This workaround can solve the fixed output limitations.

Chapter 2

Feedback requirements

In this chapter, the requirements of the environment feedback sensor are defined according to the requirements for the robot application setup.

2.1 Frequency

The value of working frequency can vary depending on the working speeds of the robot and the max speed of the dynamic objects of the environment.

The following facts need to be considered when setting the adequate frequency value,

- Average hand moving speed of a human is around $1.1m/s$ [4].
- Typical UR Robot tool speed $1m/s$ (individual joint speeds are close to 1m/s at max) [16].

The speed equation can be rewritten to compute the frequency of the feedback sensor by converting time t to frequency f using $t = 1/f$, as shown in Equation 2.1.

$$v = \frac{d}{t} = d \cdot f \rightarrow f = \frac{v}{d} \tag{2.1}$$

Form Equation 2.1, v is the relative speed between the robot and the fastest object in the workspace. In this case the speed of each object can be considered as a vector, $\vec{u_r}$ and $\vec{u_o}$ respectively, from which the worst case scenario of being the two vectors opposite (robot moving towards object and vice versa) is used to compute the relative speed with the formula $v = \vec{u_r} - \vec{u_o} = u_r + u_o$.

Finally, by setting a distance threshold d between the robot and the object, the threshold frequency can be computed (i.e.g, setting a threshold distance of $d = 1cm$, will mean that, if a hand is at 1cm from the moving robot, and starts moving at max speed, the system won't react before the collision).

For this case of study of this thesis, only objects moved by hand will be considered for the fastest workspace objects $u_o = 1.1m/s$, speed of robot $u_r = 1m/s$, and the distance threshold between $d = 0.05m$ and $d = 0.1m$. The distance threshold will not be fixed, as this work pretends to develop a proof of concept, and this range is not critical to obtaining results.

After using Equation 2.1 with the above values, the obtained frequency can range from $21Hz$ to $42Hz$, with the biggest frequency value being the preferred one.

2.2 Range

A key facet of the system's design is the sensor range and field of view adaptability. Recognizing the variability in the operational context, where the sensor may be positioned at different locations, this flexibility becomes instrumental in optimizing the system's performance across diverse scenarios.

Adjusting the sensor range accommodates scenarios where the robotic arm operates in confined and expansive spaces. In situations requiring close collaboration with human workers, a shorter sensor range might be preferable to ensure a more refined detection of nearby obstacles; around 1-2m should be enough to reach all workspace points.

The sensor placement must ensure that every corner of the workspace falls within its field of view (FoV) or coverage area.

2.3 Resolution

In this case of study, the system is required to detect objects with a minimum size of $1 - 2cm^2$, a stringent criterion for precision in object detection. This requirement is particularly noteworthy in industrial collaboration, where the robot operates near human workers and intricate machinery. The ability to identify objects at such a granular level ensures that even minute obstacles or potential collision points, such as a human finger, are not overlooked.

2.4 Sensor positioning

When it comes to feedback sensor positioning, there are two main alternatives, and the combination ob both Figure 2.1:

- Robot mount.
- External mount.
- Robot and external (multiple sensors).

Two main things must be considered when selecting the best feedback alternative: sensor type and robot task. For the sensor type case, if the sensor has a short range or small coverage area, it should be attached to the robot directly. On the other hand, if the sensor has a greater range and coverage, it can be mounted on the robot or in the workspace.

Positioning the sensor in a particular place is crucial to fulfilling a particular task. For example, tasks that require the robot to pick a certain (small) object will obtain better results with a sensor mounted on the robot facing the gripper. On the contrary, this setup won't be very useful if the task is to avoid obstacles, as the sensors may have a limited coverage; they would not cover the entire workspace despite using multiple units, making the detection of obstacles in the environment harder.

It is also important to denote that having a single or few sensors positioned in the workspace can also lead to poor performance in situations where the robot or other objects overlap. For this reason, a location from where occlusions can occur less frequently is preferable.

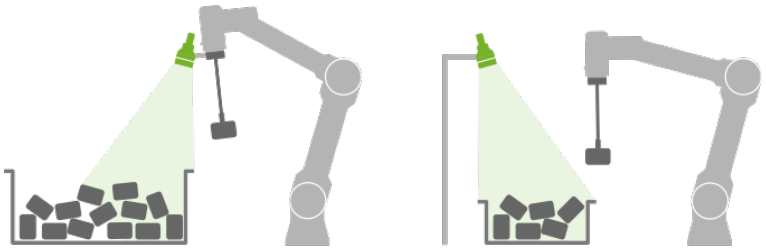


Figure 2.1: Camera mounting alternatives[44].
(right) External mount. (left) Robot mount.

2.5 Output data format

With prior research, it has been seen that utilizing a 3D volume as the input for the update of the virtual workspace model is beneficial, as it can fully capture the intricate spatial dynamics of the industrial environment. The 3D volume serves as a nuanced representation, transcending traditional 2D models by encapsulating the surface information and the volumetric aspects of the surroundings.

Directly outputting or converting sensor data into a 3D volume offers a holistic snapshot of the environment, allowing the collision check algorithm to operate with heightened precision. This depth-aware representation in robotic arm obstacle avoidance becomes indispensable for accurately assessing the proximity and potential collision points within the robot's operational space.

This 3D volume is simplified in some cases with its bounding box. This step can help reduce some algorithms' computation complexity and time.

Chapter 3

Feedback implementation

After considering all the requirements and the available sensor alternatives, the setup for the system will consist of an Intel RealSense Depth Camera D435 mounted in the workspace, and the output will be a 3D volume. This sensor has been chosen despite having less frequency because it was the one on hand, and the drop from 42Hz to 30Hz will not mean a significant change in the system performance.

3.1 Feedback pipeline

In order to generate a 3D volume from a cloud of points, several things have to be considered. First, as the sensor also detects the robot, the points corresponding to the robot must be filtered out to avoid detecting the robot as a collision object. Secondly, the system needs to handle multiple objects; if the system cannot discriminate which points correspond to each object in the environment, the constructed volume might be broken.

During the system's first tests, the robot filter's processing frequency dropped drastically, from $\sim 30Hz$ to $\sim 1Hz$. After some research, it was found that the robot filtering system was unable to cope with the high number of input points of the camera, as it needs to compute the intersection between each of the robot parts and the input point cloud, which requires many geometrical transformations.

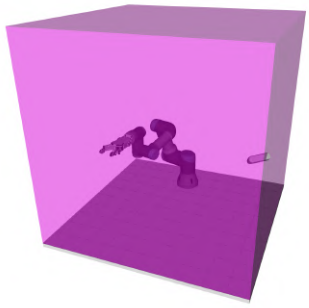


Figure 3.1: Robot Gazebo environment. Workspace (magenta volume).

A prior workspace filter was developed with Python to reduce the stress of the robot filter. The main task of this node is to filter out points outside the workspace Figure 3.1, as they are irrelevant to the collision avoidance application. This step reduced the system's performance even more to $\sim 0.1Hz$. After revising the code

execution time, it was clear that the problem reside in the programming language itself, as Python is a not compiled language, iterative operations, and ROS callbacks are slow. After changing the code from Python to C++, the output frequency of the robot filter increased to $\sim 15Hz$.

Finally, a point cloud sampling filter was used before the workspace filtering step to reduce the processing time. The used filter is a *VoxelGrid*, which is a class of the PCL library [54]. The filter divides the space into voxels of a particular defined size, then, each voxel computes the centroid of the points inside it, which become the points from the filter output cloud Figure 3.2.

This addition allowed the system to work at $\sim 30Hz$ without issues. With the clean cloud, the next task is handling the single/multiple objects reconstruction, which will be detailed in the next section.

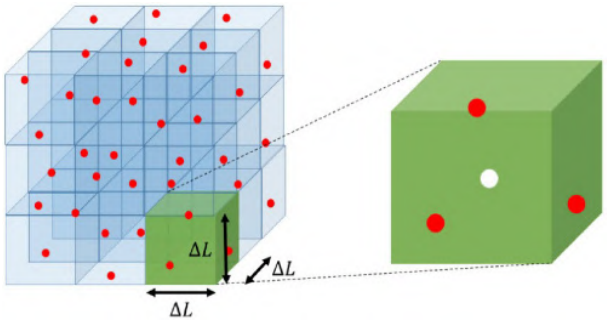
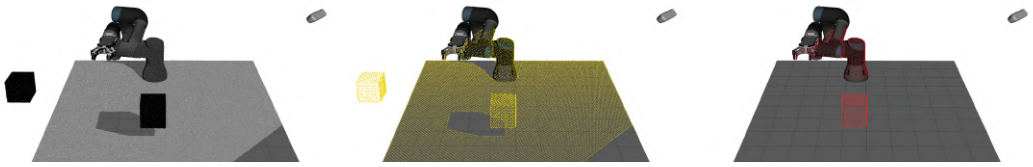
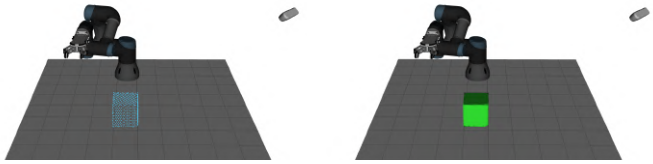


Figure 3.2: Voxel filtering diagram [23]. Voxel grid (blue squares). Voxel (green square). ΔL (Voxel size). Red dots (Point cloud). White point (centroid).

A visual representation of the data output at each stage is provided in Figure 3.3



(a) Camera point cloud (b) Sampled point cloud (c) Bounded point cloud



(d) Filtered point cloud (e) Collision volume

Figure 3.3: Feedback pipeline visual data representation.

The final workflow diagram for the feedback pipeline remains as shown in Figure 3.4.

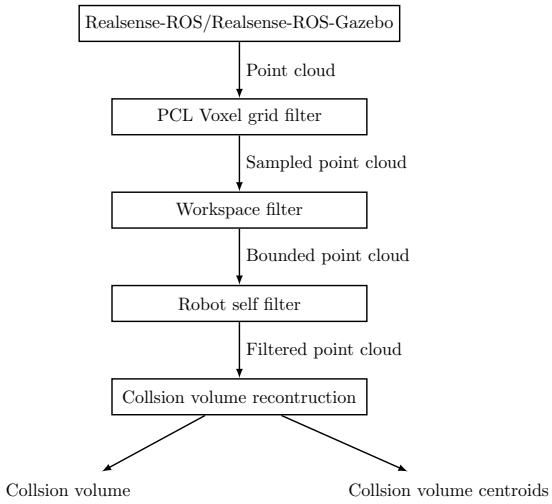


Figure 3.4: Environment feedback ROS pipeline. Boxed text (processing nodes), text (output data).

3.2 Collision volume reconstruction

The pivotal part of the feedback system is the 3D volume reconstruction, which ultimately generates the data needed for the obstacle avoidance system to detect and avoid collision objects.

This part of the work was initially developed in Python, but when the speed issues were found, it was translated to C++. The collision object reconstruction takes as input the filtered cloud and processes it in two main stages: object cloud clustering and object mesh reconstruction. Additionally, as a side product of the clusters, the centroids of the objects are obtained, which will be used for later tests.

As the point cloud is formed mostly by even distributed points, the clustering algorithm needs to consider this feature. In this case, the object will be separated when a different distance gap is detected between neighboring points. Additionally, the number of collision objects (clusters) is unknown beforehand; thus, algorithms that require this as an input parameter are not considered. The first test was carried out with Density-based clustering [66]; however, when the code was adapted to C++, another clustering algorithm with similar properties was found.

In the end, the clustering part has been implemented using the Point Cloud Library (PCL). This library has a *EuclideanClusterExtraction* [55] class, which can be used to easily segment a point cloud using Euclidean distances between neighboring points and a distance threshold d_{th}. The pseudocode of the clustering algorithm is detailed in Algorithm 1.

The *EuclideanClusterExtraction* class has the distance threshold parameter (d_{th}) defined as *cluster_tolerance*. In addition to this parameter, the class has parameters to constrain the minimum and maximum cluster sizes, which can help eliminate noise from the cloud.

Algorithm 1 Euclidean Cluster Extraction

Create a Kd-tree [50] representation for the input point cloud dataset P
Set up an empty list of clusters C and a queue of the points that need to be checked Q
for every point $\boldsymbol{p}_i \in P$ **do**
  Add $\boldsymbol{p}_i$ to the current queue Q
  for every point $\boldsymbol{p}_i \in Q$ **do**
    Search for the set P_i^k of point neighbors of $\boldsymbol{p}_i$ in a sphere with radius $r < d_{th}$
    for every neighbor $\boldsymbol{p}_i^k \in P_i^k$ **do**
      if the point has not been processed **then**
        Add it to Q
      end if
    end for
  end for
  Add Q to the list of clusters C, and reset Q to an empty list
end for

The algorithm terminates when all points $\boldsymbol{p}_i \in P$ have been processed and are now part of the list of point clusters C.

Once the clusters are extracted, the code iterates over each cluster independently to extract their centroids and compute the 3D volume.

In 3D modeling, 3D volumes are a solid representation of 3D spatial information. This spatial information can come in a wide range of formats, but it is essential to define the data correctly when a specific format is used. One of the most used 3D volume data structure methods is a polygon mesh consisting of vertices, edges, and faces. These faces are defined by subgroups of edges and vertices that form simple polygons and are the building blocks for defining the surface of the 3D volume Figure 3.5.

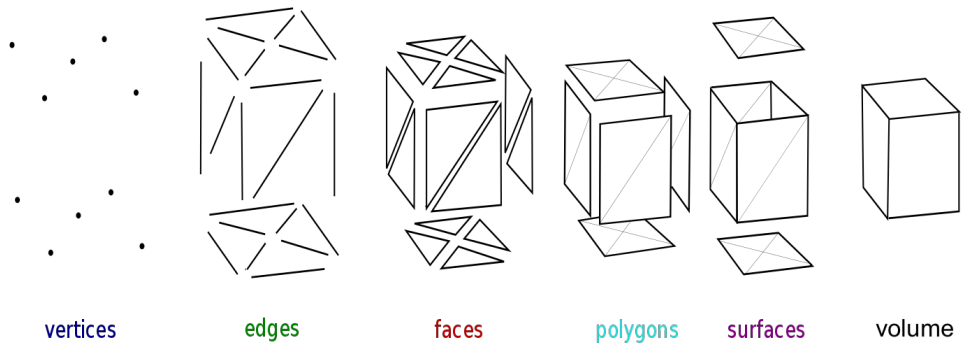


Figure 3.5: Polygon mesh insight [52].

Despite having a cloud of points that can be used as vertices for the mesh, there is still no definition for the vertices (union between points) and the faces (groups of vertices). Luckily, some algorithms reconstruct a Mesh with a given point cloud.

The PCL library offers some functions to make mesh reconstructions [68] from a point cloud, such as Greedy Projection Triangulation [56] or Concave/Convex Hull [53]. However, after some tests, the results obtained were not consistent. These results will be further analyzed later.

The other alternative for 3D mesh reconstruction is the Visualization Toolkit (VTK) [71]. This library also has some mesh reconstruction functions [69]:

- ***Delaunay3D***: Create a solid mesh from Unorganized Points.
- ***Delaunay2D***: Create a surface mesh from Unorganized Points.
- ***ExtractSurface***: Create a surface from Unorganized Points using Point filters.
- ***GaussianSplat***: Create a surface from Unorganized Points (Gaussian Splat).
- ***SurfaceFromUnorganizedPoints*** Create a surface from Unorganized Points.
- ***PoissonExtractSurface***: Create a surface from Unorganized Points using the ***PoissonReconstruction*** algorithm.
- ***ConvexHull***: Create a Convex hull mesh using ***vtkHull***.
- ***ConvexHullShrinkWrap***: Approximate Convex hull mesh using shrink wrapping.

For this case of study, the ***Delaunay3D***, ***Delaunay2D***, ***GaussianSplat***, ***SurfaceFromUnorganizedPoints*** and ***ConvexHull*** have been implemented in C++ and tested, with great results, which will be further analyzed later. The output of these functions is then cleaned and converted from ***vtkPolyData*** to ROS standard mesh format. Inside the ROS communication messages, a package called Shape_msgs [64] can be found, which offers a Mesh [65] data structure. This format is the one chosen to send the output collision mesh via ROS topic and add the collision mesh to the ***MoveIt Planning Interface***.

A launch file has also been created to make the feedback pipeline execution easier, with default parameters to ensure reproducibility. This file is saved under the *obstacle_avoidance* ROS package, inside the launch folder, with the name of *obstacle_avoidance.launch*. Note: This launch file does not bring up the *realsense-ros* camera node. Thus, prior execution is needed.

Chapter 4

Obstacle Avoidance Requirements

4.1 Computation time

The system must detect a collision with an object before the robot reaches the object to ensure safety. The time to detect collision has been computed when defining the sensor frequency in chapter 2, where the minimum workspace refresh rate has been computed.

Supposing the workspaces' information is updated at $30Hz$, as detailed in the used sensor characteristics. In that case, it means that the object at worst is changing position every $0.033s$, leaving this time as a margin to detect if the robot will collide with the object.

There are different scenarios to consider for the obstacle avoidance computation time. In the simplest obstacle avoidance task, where the object is assumed to be static, there are no computation time limitations. The obstacle will remain there after the computations are performed, and the new avoidance path will remain valid.

In another case, objects move and stop blocking the path before the collision avoidance is triggered. For this, it can be assumed the same is with the static case.

Finally, in the case where the obstacle is moving continuously, periodic trajectory corrections would need to be done, leaving the available time for computing the obstacle-avoiding path to less than the refresh rate of the workspace environment with the collision check because the object can change position between the two updates. In some cases, a new trajectory would need to be computed as the object moves to a place where it blocks the newly computed trajectory.

4.2 Collision check

The collision check code needs to be able to detect any collision between the robot and the objects inside the workspace through the whole future positions that the robot will take and determine the remaining distance to collision, the part of the robot that is colliding, and with which object is colliding.

Determining the distance is crucial in stopping the robot before the collision, as it won't be efficient to stop the robot because some object obstructed the goal while the robot was still close to the start, and it won't be good to stop the robot once it has come in contact with the object. This collision threshold would be set between $0.2 - 0.3m$ to have a generous margin for avoidance computations.

On the other hand, determining the colliding pairs is useful to trigger different actions during robot execution. Stopping with avoidance in the case of an end-effector collision trigger, while execution stop with auto-resume is triggered when another part of the robot is collision detected, always ensuring safety.

4.3 Avoidance constrains

The obstacle avoidance system will only contemplate collisions with the robot's end-effector for the new trajectory generation, as it is one of the main objectives of this thesis, and full robot collision avoidance is out of scope.

The objective of the previous work was to generate robot movement that followed a user demonstration. This objective has been kept in this thesis, which means that the generated avoidance trajectory should be as close as possible to the one generated by the DMP imitated trajectory (originally planned trajectory).

The ideal case would be modifying the trajectory just the needed amount to avoid contact between the obstacle and the end-effector. However, a safety distance will be present to account for camera error and reconstruction imprecision. This distance can be around $2 - 5cm$.

Chapter 5

Obstacle avoidance implementation

After considering all the requirements and the available algorithm alternatives for the obstacle avoidance task, the system will consist of a base Cartesian Dynamics Movement Primitives in combination with an Artificial Potential Field forcing term.

5.1 Avoidance pipeline

The first step to generate a successful obstacle-avoiding trajectory that at the same time follows a demonstrated trajectory is to monitor the collisions with the workspace to have a trigger of the avoidance path computation to reduce the computation load.

Once the collision is detected, the system waits until a certain collision distance is reached, the robot movement is stopped, and the current robot position in the trajectory is checked to forward only the remaining path to the avoidance computation step, as the previous trajectory points are now irrelevant.
It is important to denote that the initial approach of sending the trajectory pointwise as individual robot goals failed due to significant execution delays, making the robot slower than expected; thus, to achieve the expected movement speeds, the trajectory had to be sent as a single goal. Then, the current position is checked by computing the distance between the end-effector position of the current state and the end-effector position of all the robot states during the trajectory and getting the trajectory index corresponding to the least distance. This end effector position has been computed using Forward Kinematics.

Then, the system evaluates if the collision is with the end-effector or not to decide if the obstacle avoidance is triggered or if it remains stopped until the collision object moves because the collision can't be avoided as it is not contemplated for the avoidance capabilities of this work.

When the collision is with the end-efector, the system sends the current remaining trajectory as a demo for the Cartesian DMP, to generate the corresponding forcing term weights. These weights are then used to generate the new trajectory, taking as input the current robot Cartesian pose, the goal pose, and the obstacle (centroid/mesh vertices).

Finally, as the robot works in the joint state-space and the return of the obstacle avoidance DMP is in Cartesian space, the trajectory inverse kinematics are computed, and the movement is executed again. The problems encountered during this step will be detailed in the next sections.

The final workflow diagram for obstacle avoidance remains as shown in Figure 5.1.

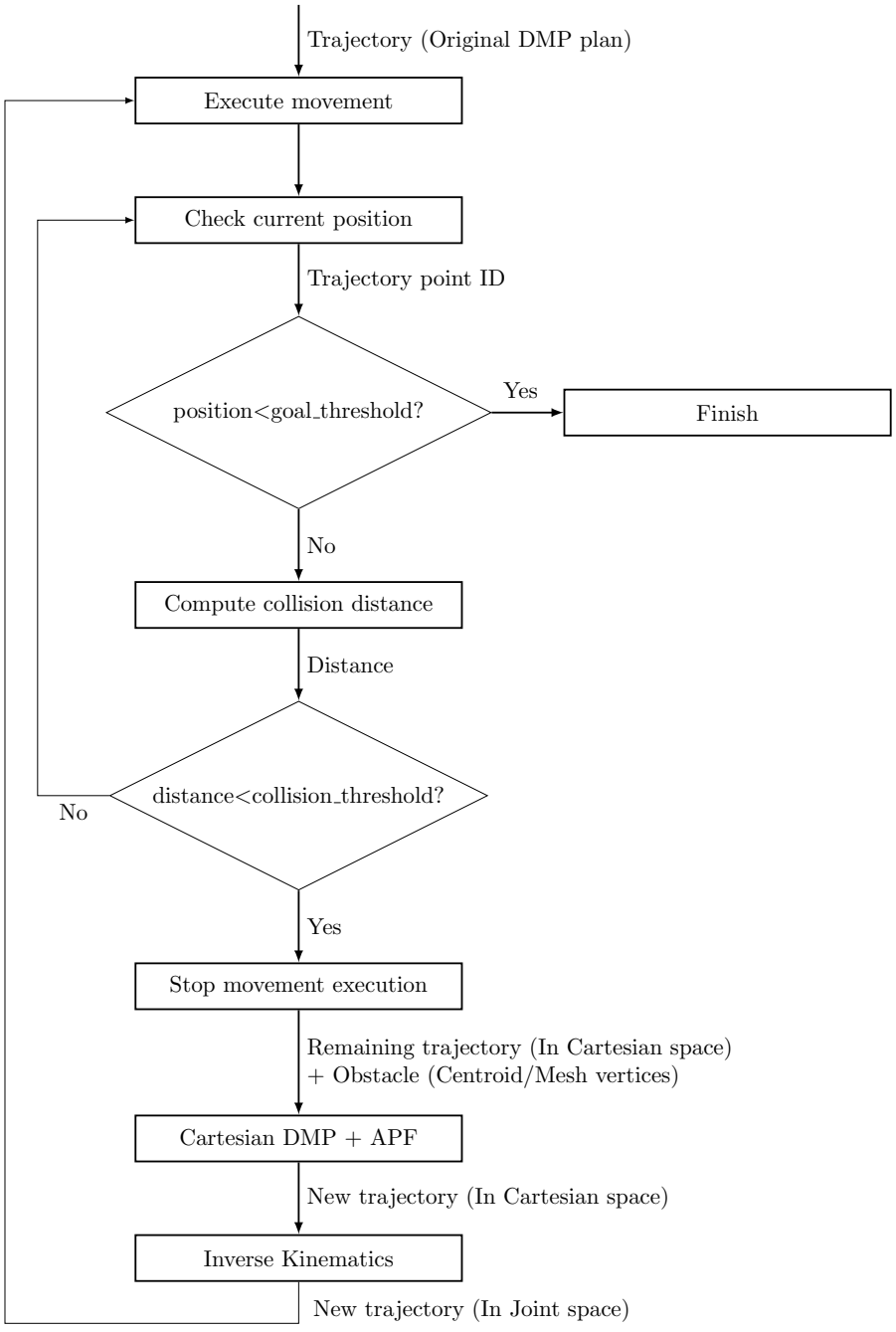


Figure 5.1: Obstacle avoidance pipeline.

5.2 Collision check

The initial approach to get the collisions has been to use the ***MoveIt planning interface*** tools. These tools check if a certain robot state collides with the planning scene environment. As the planning scene is loaded with the collision objects with the code generated for the feedback implementation, it is only left to move a virtual version of the robot to the pose corresponding to each path point and check the collisions with the ***checkCollision*** request function. This function takes a robot state and the planning scene (workspace) state and returns if a collision occurred, the objects involved in the collision, and the collision point in the robot reference fame (***base_link*** in the work setup). Figure 5.2 shows how the robot is in state T and every future state $T + i$ is checked for collision. In this case, the collision is detected at $T+2$, which means that the collision check will stop computing to fasten the computations, and will send that the end-effector is colliding with an object at a certain point.

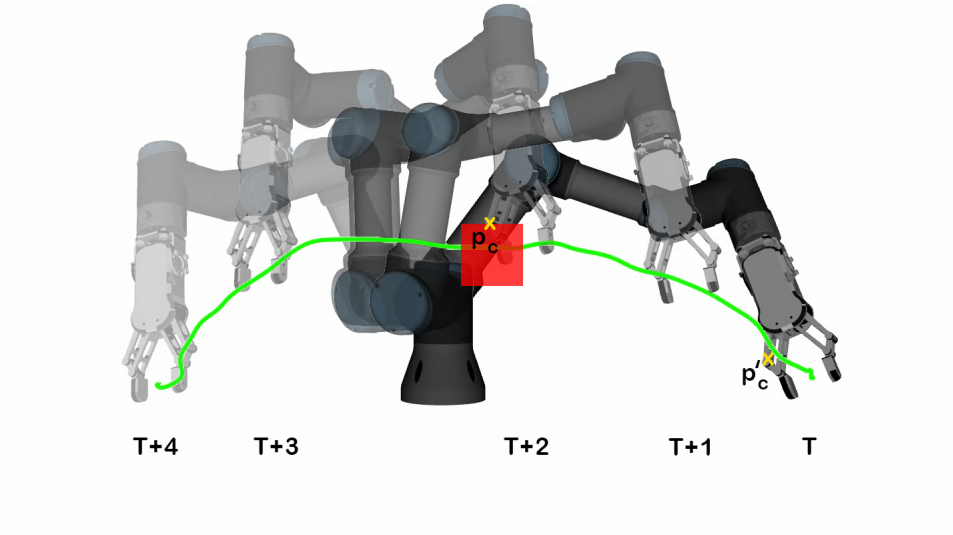


Figure 5.2: Collision check diagram. Trajectory (green). Obstacle (red). Collision point (yellow)

In addition, the distance between the current state and the collision must be computed. MoveIt provides some alternatives to do so; however, none of them provides consistent returns. A code has been implemented to compute this collision distance by finding the same collision point of the robot position in the robot reference frame of the initial state T. The collision point p_c frame of reference is changed from the robot reference frame (***base_link***) to the frame of the robot colliding part (***rg2_body_link***) using the inverse of the homogeneous transformation matrix in the collision state $T+2$, FT^{-1}_{T+2}. And then the collision point in the robot part frame is then converted again to the robot reference frame using the homogeneous transformation matrix in the current state T, (FT_T). Equation 5.2 shows the transformation steps. Generating a new point p'_c that has the same relative position as the robot part but has some displacement between the obtained collision point p_c, which can

be used to compute the collision distance using the L2 norm. The homogeneous transformation matrices, Equation 5.1, account for the rotation (R) and transformation (t) between two frames of reference. The matrices can be obtained with the MoveIt function ***getGlobalLinkTransform*** and inverted with the ***inverse()*** function.

$$TF = \begin{pmatrix} R & : & t \\ \cdots & \cdots & \cdots \\ 0 \;\; 0 & 0 & : \; 1 \end{pmatrix} \implies TF^{-1} = \begin{pmatrix} R^T & : & -R^T \cdot t \\ \cdots & \cdots & \cdots \\ 0 \;\; 0 & 0 & : \quad 1 \end{pmatrix} \tag{5.1}$$

$$p'_c = TF_T \cdot (TF^{-1}_{T+i} \cdot p_c) \tag{5.2}$$

5.3 Avoidance computation

As it has been mentioned at the start of this chapter, the computation of the obstacle-avoiding path has been implemented using a Cartesian DMP, which has been modified to have an additional forcing term, called coupling term, that will follow the dynamics of an Artificial Potential Field.

It is important to mention that the DMP from the previous reference thesis worked in the joint state-space; however, for this work, it has been adapted to a Cartesian DMP, because the computation of this coupling term requires the system to be in Cartesian state-space.

The code makes a learn-from-demo request with the remaining original trajectory in Cartesian space to generate the obstacle avoidance trajectory. Then, the system computes the weight of the DMP, and finally, the initial pose and goal are sent alongside the obstacle centroid of the collision object and some DMP+APF parameters. The return is a trajectory in Cartesian space that avoids the obstacle.

Trajectory conversion from joint space to Cartesian space is done using MoveIt ***getGlobalLinkTransform*** function, which can extract a robot frame transformation (pose of a robot link axis) by computing Forward Kinematics from a robot state. In this case, the end effector frame (***rg2_eef_link***) pose is computed using a virtual state of the robot set using joints.

The obstacle avoidance has two alternatives for obstacle input. One is the centroid of the obstacle, which assumes that the obstacle has negligible size and shape and relies on well-tuned APF parameters to generate a trajectory that does not collide. This tuning can be tricky when obstacle sizes vary, as big obstacles require more repulsion power while small objects require less. For this reason, the second alternative accounts for the object's size by having the whole obstacle mesh vertices as input; with this, the closest vertex to the evaluated trajectory point can be used to add an extra boost when the object is bigger. Further explanation of the Cartesian DMP with APF is provided in the next section.

After the trajectory is returned, it is converted from Cartesian space to joint space with an inverse kinematics solver. Allowing it to be fed into the execution loop again. This step has been detailed deeply in the section below due to the numerous issues encountered during this step,

The system can handle multiple obstacles by iteratively recomputing the new obstacle avoidance trajectory, one at a time.

5.3.1 Cartesian Dynamic Movement Primitives

For the implementation of the Cartesian Dynamic Movement Primitives trajectory generation, the Auke Ijspeert [13] Dynamic Movement Primitives algorithm version has been used.

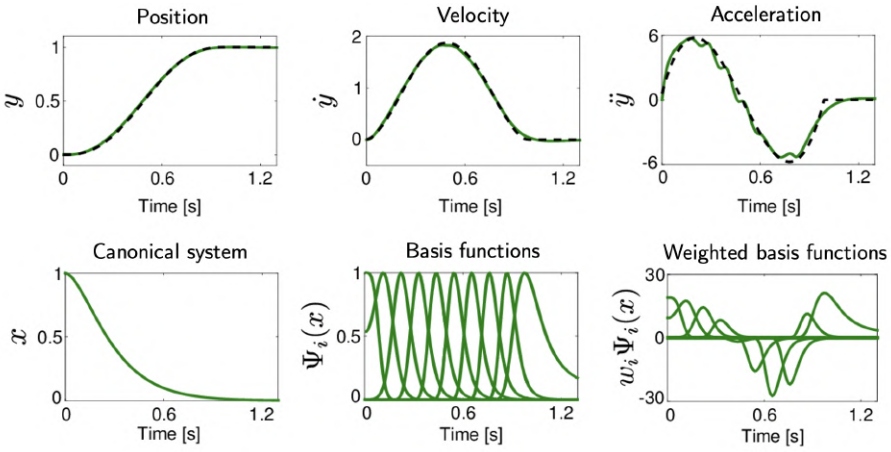


Figure 5.3: Traditional DMP path planning [30]. Example data (black dashed lines). Output (green lines).

The standard DMP algorithm starts with defining the main dynamic, which is related to a damped spring model, Equation 5.3. Where y is the desired system state, g is the goal, α_y and β_y are time constants, τ is a temporal scaling factor and f a forcing term. α_y and β_y parameters need to be chosen (with $\beta_y = \alpha_y/4$) to ensure that the system is critically damped, which means that y converges to g monotonically.

$$\tau\,\ddot{\mathbf{y}} = \alpha_y(\beta_y(\mathbf{g}_y - \mathbf{y}) - \dot{\mathbf{y}}) + f \tag{5.3}$$

This forcing term is in charge of modifying the trajectory to fit the given example. It needs to be nonlinear to allow the reproducibility of more complex trajectories by changing the exponential convergence of the second-order dynamic. However, this makes the system enter the domain of nonlinear dynamics, which can increase its complexity; thus, setting f smartly is a must.

In this approach, time (t) is replaced by a phase (x) that follows the dynamics of Equation 5.4. This term is called canonical system, and it exponentially decays proportional to α_x and converges to 0 monotonically, modeling the generic behavior of the model equations.

$$\tau \dot{x} = -\alpha_x x, \quad x_0 = 1 \tag{5.4}$$

The forcing term (f) defined in Equation 5.5 as a normalized weighted sum of Gaussian basis functions ψ_i, with the phase x and a spatial scaling term $(g - y_0)$ to modulate movement amplitude. By using the phase (x), it can be assured that the forcing term effects vanish when g is reached.

$$f(x, \mathbf{g}) = \frac{\sum_{i=1}^{N} \psi_i \omega_i}{\sum_{i=1}^{N} \psi_i} x(\mathbf{g} - \mathbf{y}_0) \tag{5.5}$$

The Gaussian basis functions ψ_i are centered at c_i and have a spread of h_i, as detailed in Equation 5.6.

$$\psi_i = \exp\big(h_i(x - c_i)^2\big) \tag{5.6}$$

The solutions for this system can be approached using optimization techniques, such as locally weighted regression. The obtained solution is detailed in Equation 5.7

$$w_i = \frac{\mathrm{s}^T \psi_i f_d}{\mathrm{s}^T \psi_i s}, \quad s = \begin{pmatrix} x_{t_0}(g - y_0) \\ \vdots \\ x_{t_N}(g - y_0) \end{pmatrix}, \quad \psi_i = \begin{pmatrix} \psi_i(t_0) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \psi_i(t_N) \end{pmatrix} \tag{5.7}$$

This DMP version implementation has been borrowed from Scott Niekum [15], as had been done in the previous work. Using the Fourier approximator for the solver.

By adapting the input of the request, this code can be used in the Cartesian space. This adaptation can be done because the code input dimensions are flexible, and the space representation won't affect the output demo approximation. Thus, the input format of the learn-from-demo request and the initial and target states have been defined as a pose vector that contains $[X, Y, Z, Y, P, R]$, being X, Y, Z the end-effector point in Cartesian space and Y, P, R the Euler angles of rotation of the end-effector link. The system's output will be the same as the input format, and the output of the derivatives will represent linear and angular velocities, respectively.

5.3.2 Artificial Potential Field

Recalling what has been commented on in the introduction section, DMP main dynamics, Equation 5.3, can be modified by adding a coupling term (p), that follows the dynamics of artificial potential fields to enable obstacle avoidance, Equation 5.8.

$$\tau \ddot{\mathbf{y}} = \alpha_y(\beta_y(\mathbf{g}_y - \mathbf{y}) - \dot{\mathbf{y}}) + f + p \tag{5.8}$$

This term needs to be computed at every step in Cartesian space, and its effect is also in this space; that is why it was essential to have the DMP working in this space. Additionally, it has to be taken into account that only end effector position is vital during obstacle avoidance for this particular work, which means that the computed coupling term (p) is going to have the format $[X'', Y'', Z'', 0, 0, 0]$, where X'', Y'', Z'' are the Cartesian accelerations that will force the avoidance dynamics and the 0 values are set to ensure that the rotations are not affected. All the system states (y) used inside the coupling term computations will only account for the Cartesian position $[X, Y, Z]$.

The first implementation of the coupling term has been implemented from [9], which uses human behavioral dynamics of obstacle avoidance to generate the coupling term Equation 5.9, where γ and β are constant, φ is the angle between the direction of the closest point towards the obstacle, Equation 5.10, and R is a rotational matrix with axis $r = (y - y_{obs}) \times \dot{y}$ and angle $\pi/2$. By looking at Equation 5.8, it can be seen that this coupling term acts as an acceleration that will modify the velocity during the integration step and, consequently, affect the position. Figure 5.4 provides a more graphical representation of the abovementioned concepts.

$$p(y, y_{obs}) = \gamma \, R \, \dot{y} \, \varphi \exp(-\beta \, \varphi) \tag{5.9}$$

$$\varphi = \arccos \left[\frac{(y_{obs} - y)^T \cdot \dot{y}}{|y_{obs} - y||\dot{y}|} \right] \tag{5.10}$$

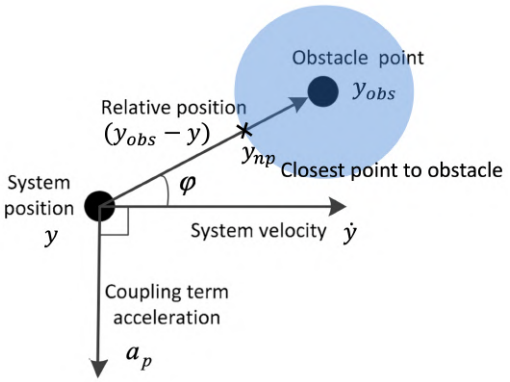


Figure 5.4: Motion analysis diagram of the system and obstacle [20].

It is important to note that despite no potential fields mentioned in this approach, the dynamics of this coupling term follow the dynamics of a repulsive potential field of the end-effector velocity. End-effector states with a velocity that points to the object (small φ) will generate a greater repulsion (greater absolute values of p) perpendicular to the velocity. States with a velocity that points away from the obstacle (big φ) won't be affected, keeping the same velocity direction ($p \sim 0$). The constants γ and β will be adjusted to determine the field's intensity and the angle's influence, respectively. This effect can be seen in Figure 5.5. It is also noticeable that this system won't change the coupling term result when φ is the same and the end-effector is closer, meaning that either the collision stop threshold would need to be big enough to give sufficient time to change the velocity or a big value γ would need to be set to apply a greater acceleration, changing the velocity direction faster.

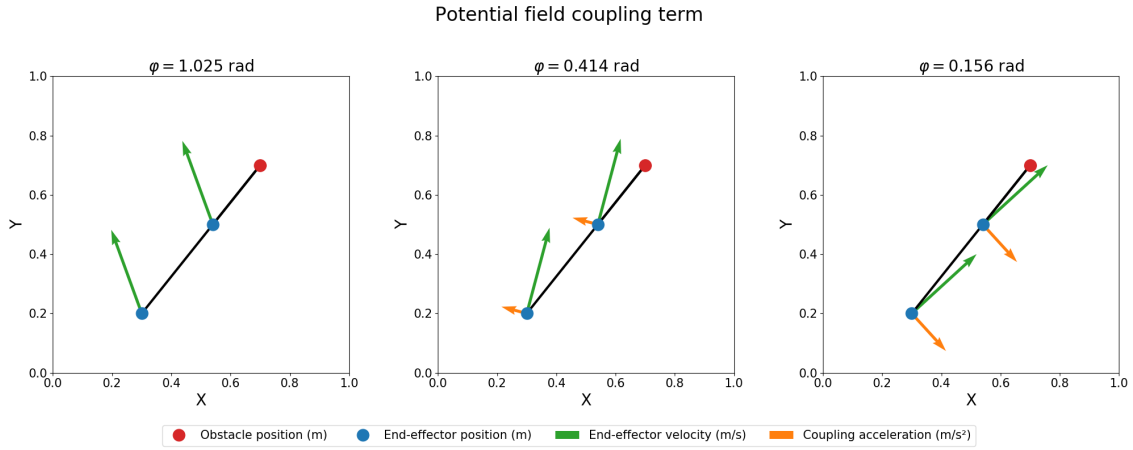


Figure 5.5: Coupling term effect in base algorithm.

An improved approach [20], modifies Equation 5.9 to allow taking into account the relative position $(y_{obs} - y)$ between the end-effector and the obstacle, in addition to the angle φ. Generating Equation 5.11, with the same parameters as before, plus the relative distance $(y_{obs} - y)$ and k that adjusts the influence of relative distance.

$$p(y, y_{obs}) = \gamma \, R \, \dot{y} \, \varphi \exp(-\beta \, \varphi) \exp(-k(y_{obs} - y)) \tag{5.11}$$

The effects of the new term can be observed in Figure 5.6. As the end-effector approaches the obstacle, the coupling term absolute value increases, ensuring that the system can avoid the obstacle even when the end-effector is very close to the obstacle. However, both of these approaches only consider the centroid of the obstacle to generate the avoidance, which means that to ensure that the obstacle volume is not hit, the intensity of the field (*gamma*) needs to be adjusted accordingly.

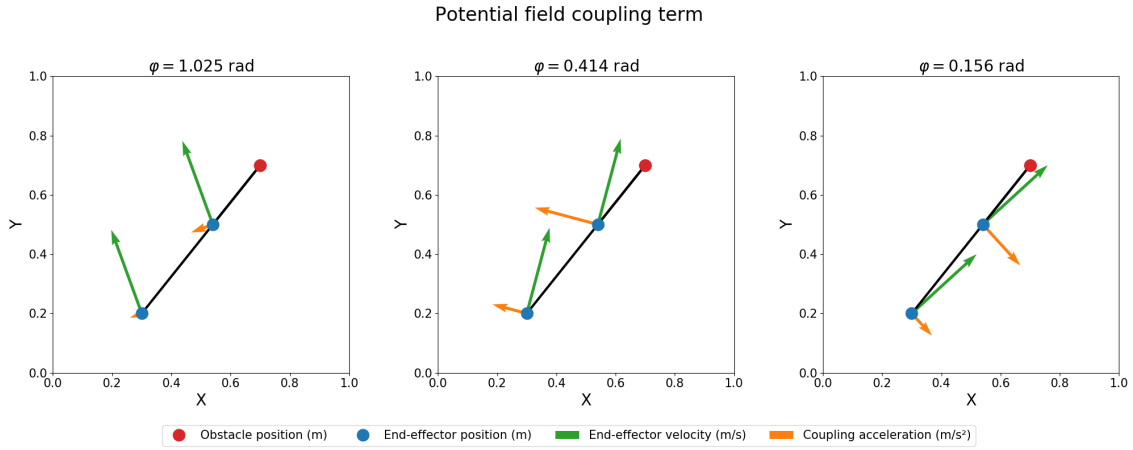


Figure 5.6: Coupling term effect in algorithm with relative distance.

To better handle the avoidance of the obstacles with non-negligible volume, the same paper as before [20], proposes to add another term to account for the relative position between the end-effector and the closest point to the obstacle, Equation 5.12. Different values for γ, β, and k are used for each of the terms to tune individually the repulsive field contribution.

$$p(y, y_{obs}) = \gamma_{np} Rij\psi_{np} + \gamma_{obs} Rij\psi_{obs} \tag{5.12}$$

$$\psi_{np} = \varphi \exp(-\beta_{np}\varphi) \exp(-k_{np}\ (y_{np} - y)) \tag{5.13}$$

$$\psi_{obs} = \varphi \exp(-\beta_{obs}\varphi) \exp(-k_{obs}\ (y_{obs} - y)) \tag{5.14}$$

The effects of this new term can be seen in Figure 5.7, whereas the closest point of the obstacle is closer to the end-effector, the generated coupling absolute value increases.

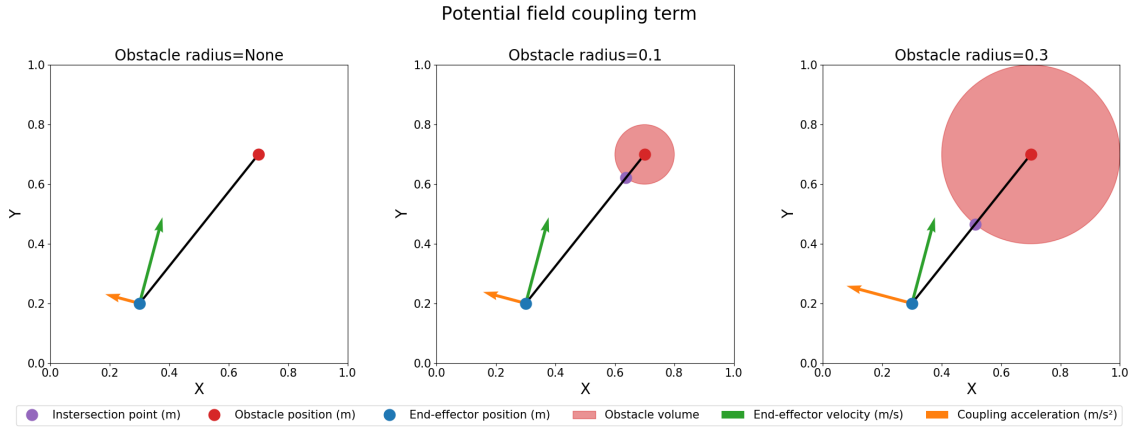


Figure 5.7: Coupling term effect in algorithm with non-negligible volume.

Several approaches have been considered to compute the closest point. The first was to directly find the intersection between the obstacle mesh and the line that joins the obstacle centroid and the end-effector position. However, the algorithms required to solve this (e.g. ray-triangle intersection) are often slow as they make intersection checks iteratively. Luckily, as the only purpose of this closest point is to scale the coupling response, it is not strictly required that the point is collinear with the line that joins the centroid and the end-effector, which means that a minimum distance computation between the vertices and the end-effector will be sufficient to obtain a significant approximation. The resolution of the mesh or number of vertices is a parameter that can be adjusted in the reconstruction step. If the mesh maximum resolution is insufficient, it is still possible to adapt the code to work directly with the clustered point cloud of the object.

The previous work, [20], also proposes the addition of another term, Equation 5.15, to generate coupling only dependent on the relative distance between the nearest point and the end-effector, without taking into account the system velocity direction. This term can ensure the system has sufficient acceleration to avoid an obstacle when the end-effector is very close, and the velocity direction is pointing away from the obstacle centroid. Ensuring the generated acceleration is enough to deviate the movement in the remaining distance to the collision.

$$p(y, y_{obs}, y_{np}) = \gamma_{np} R\dot{y}\psi_{np} + \gamma_{obs} R\dot{y}\psi_{obs} + \gamma_d R\dot{y} \exp(-k_d(y_{obs} - y)) \tag{5.15}$$

A comparative is shown in Figure 5.8 to illustrate the effects of each iteration under the same conditions.

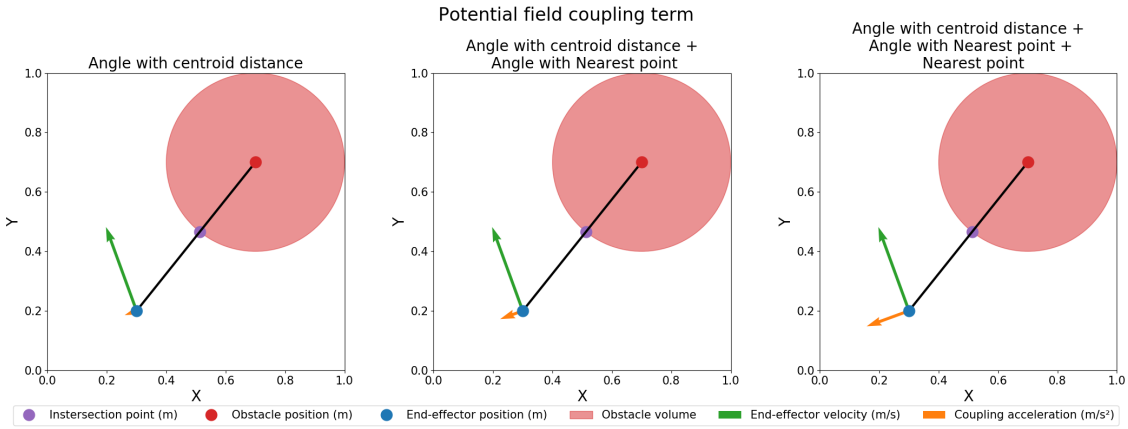


Figure 5.8: DMP+APF iterations comparison.

Finally, after some tests, it was observed that the coupling term generation produced a big impact in small objects if the same parameters were tuned to avoid big and small objects simultaneously, generating an avoidance trajectory that differed significantly from the optimal. A custom-made modification has been applied to the final coupling term value, Equation 5.16, reducing this coupling term effect.

The idea has been to multiply the coupling term effect by some factor depending on each dimension's obstacle bounding box size. This term needed to be close to 1.0 for big obstacles (even surpass one if obstacles are very big) as the full effect is needed, and it will need to be reduced as the size becomes smaller and smaller, and less effect is needed. To match this behavior, a line equation has been used to directly translate between the size and the multiplying factor, Equation 5.16, where m is the slope of the line, and n is the y-intercept.

$$p'(y, y_{obs}, y_{np}) = p(y, y_{obs}, y_{np}) \cdot s(size), \quad s(size) = m \cdot size + n \tag{5.16}$$

By changing m and n, the decreasing rate and value of the scaling factor can be adjusted to reduce the coupling when a small obstacle is present. In the case of irregular obstacles, with only some small sizes in some dimensions, the coupling in each dimension is scaled accordingly.

The addition of the potential field can prevent the DMP from converging into the target point when this point is inside the obstacle volume. To avoid generating a nonconverging path that can lead to a spaghetti-like output, a system has been implemented to check if the obstacle volume overlaps the target, and if so, the movement is stopped and the avoidance execution waits until the obstacle is moved.

5.3.3 Inverse Kinematics solver

The final part of this work is the conversion between Cartesian space and joint space to send the trajectory goals to the robot.

This conversion can often lead to multiple solutions, making the conversion of a whole trajectory tricky, as joint position steps should be bounded between two time steps to ensure smooth movement.

The ***planning_interface::MoveGroupInterface*** from MoveIt provides a function called ***computeCartesianPath***. It takes the path point's pose and computes the inverse kinematics while keeping the joint trajectory smooth. Unfortunately, this method does not ensure that the whole path is converted because the computation will end if no IK solution is found at one point. Additionally, if a big jump is detected between two consecutive joint states, the path is trimmed to the point before the jump, which, in some cases, reduces the returned trajectory even more.

In order to allow testing with the robot, the code has been modified to ignore the points with no IK solution without terminating path computation, as the provided Cartesian path has sufficient points to keep the computation consistent. An end-effector velocity filter has been added to ensure that the system does not accelerate suddenly. However, there is still the possibility of having small joint jumps.

5.4 GUI integration

One of the main objectives of this thesis has been the continuation of a previous work that laid the basis of a user-friendly control framework for the robot. This framework consisted of a Graphical User Interface with different screens that provided different features to enable controlling the robot via learning from demonstration. To see how the GUI fully works, look at the previous work thesis [27]

The obstacle avoidance has been incorporated in the execution part. As the avoidance will trigger during execution.

The first feature that has been added is the workspace movement detection. This feature has been the first step to ensuring that the system has real-time workspace feedback, and it can be used as a safety measure in applications where the robot is intended to work alone. However, humans/objects can enter the environment while the robot moves. If any movement external to the robot is detected, a stopping signal will be sent, and the robot's execution will stop. In this application, manual steps are required to re-launch the robot movement. To enable this feature, a checkbox with the text ***"Movement detection safe stop"*** has been added. If this feature is checked, the obstacle avoidance checkbox is disabled until this feature is not checked.

The second implemented feature has been obstacle avoidance. This feature provides access to the obstacle avoidance pipeline developed for this thesis. The checkbox with the text ***"Real-time obstacle avoidance"*** is activated, and the execute plan button calls the obstacle avoidance code instead of the original execution code. It is still possible to move the robot using the original execution code without using obstacle avoidance.

Modifications to the existing collision check code of the application have been made to ensure that only self-collisions are checked during plan generation when obstacle avoidance is activated. As a side product, the system can now account for collisions with the workspace objects (updated with the feedback pipeline) during collision check when the generate plan is pressed and the obstacle avoidance is not active.

To execute the movement with collision avoidance, select the desired DMP weights of the demo, set the initial and goal position, and enable the obstacle avoidance checkbox; then, the plan can be generated and previewed. Finally, the plan can be executed by clicking the execute button. The execute button will send the computed trajectory to the obstacle avoidance code. As before, if this feature is checked, the safe stop checkbox is disabled until this feature is not checked.

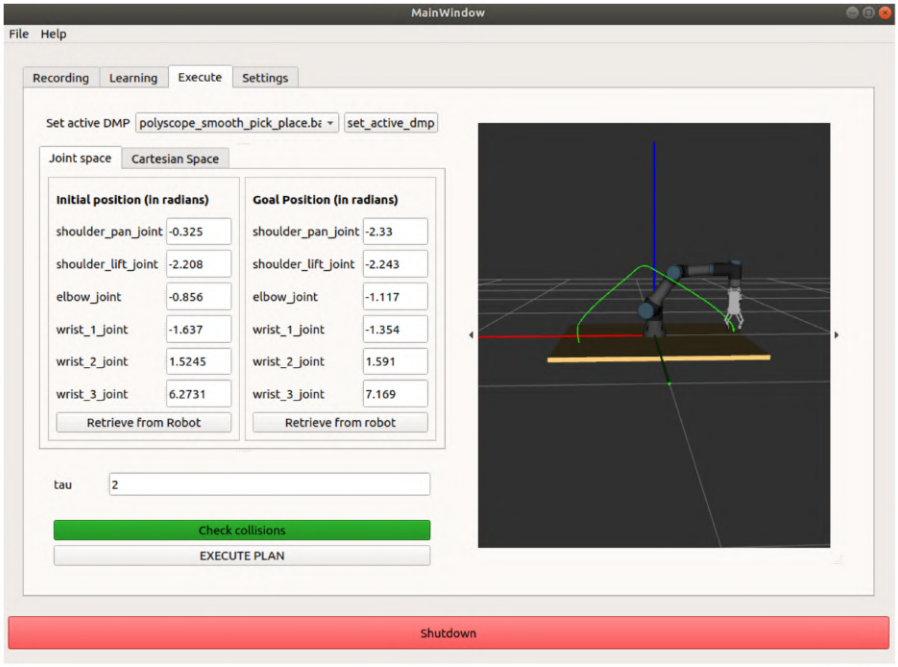


Figure 5.9: Original GUI execution tab.

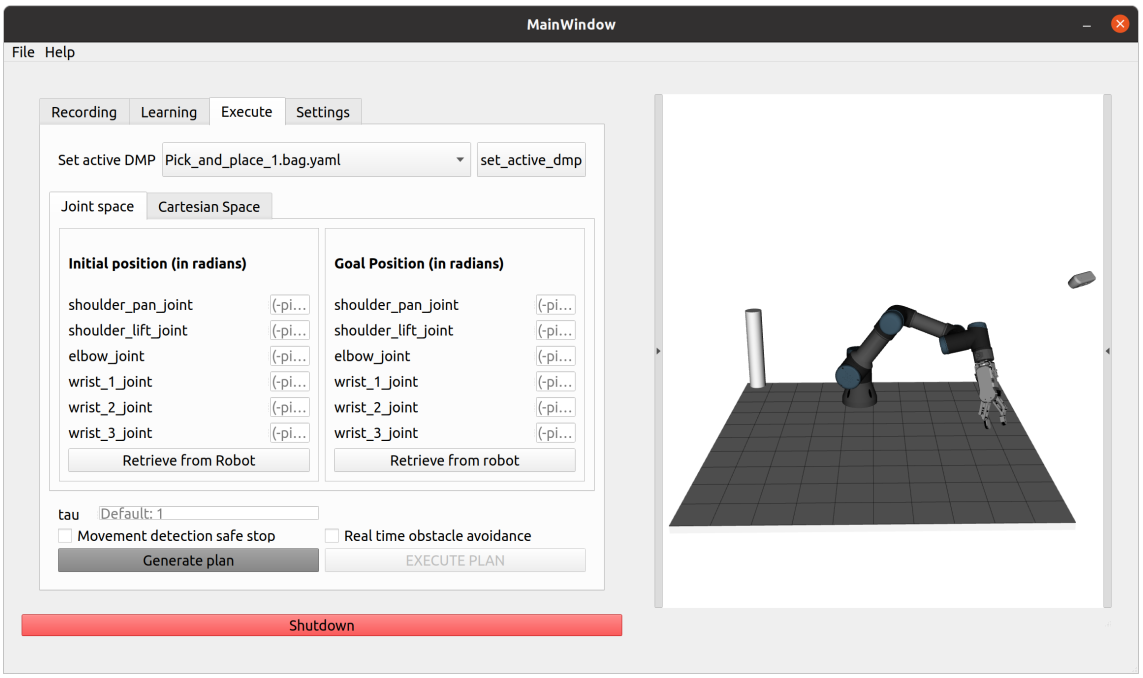


Figure 5.10: New GUI execution tab.

An additional minor feature has been added, to allow the enabling and disabling of the RViz plugin inside the GUI, the ***"Enable RViz viwe"*** checkbox. This is useful to avoid GUI crashes when running the system in computers with limited resources.

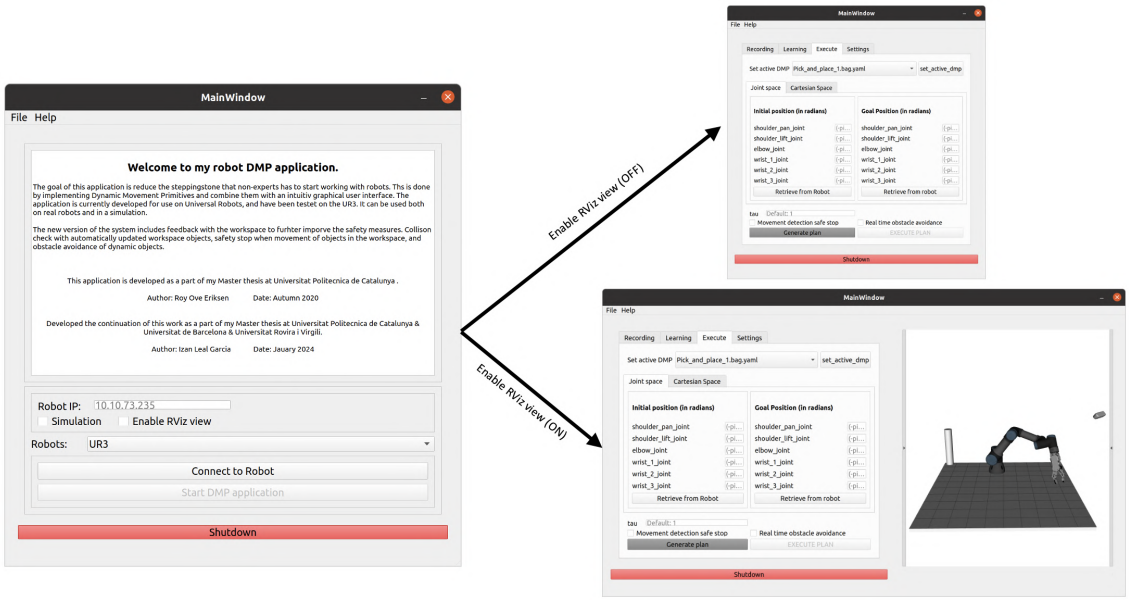


Figure 5.11: Enable RViz view alternatives.

Chapter 6

Results

6.1 Evaluation of the workspace feedback

Several steps will be carried out to evaluate the feedback system. First, the update frequency of the virtual workspace will be checked. Then, the clustering algorithm will be tested with different object configurations, and finally, the consistency of the mesh reconstruction of different objects will be inspected visually.

The following setups with three different objects and relative positions are used for evaluation. Figure 6.1 and Figure 6.2.

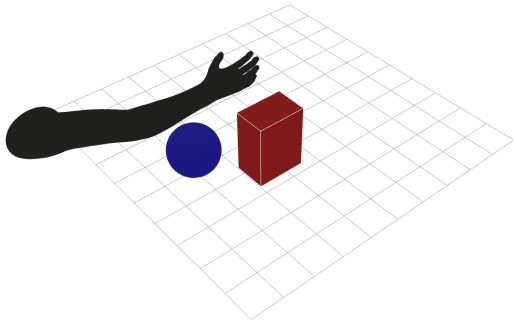


Figure 6.1: Clustering with close objects.

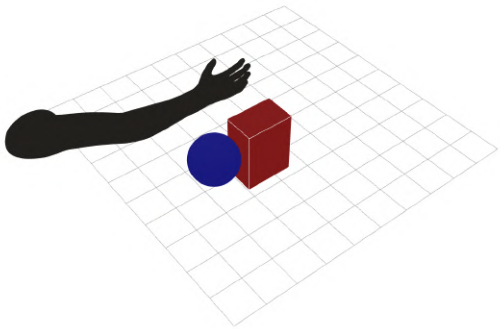


Figure 6.2: Clustering with contact objects.

In Figure 6.1, objects are close, with about $\sim 0.05m$ of spacing between them. In Figure 6.2, two of the object surfaces are in contact.

6.1.1 Refresh frequency

The evaluation of the workspace refresh rate can be determined with the help of the RQT topic monitor ROS plugin. With this tool, it has been observed that the system is updating the collision objects in the workspace at a rate of $\sim 30Hz$, which fulfills the requirements that were set for the system.

6.1.2 Clustering

In clustering evaluation, multiple objects have been added into the workspace with different relative positions to check if the object differentiation is correct.

By recalling the feedback pipeline, it is known that the camera point cloud is downsampled with a voxel filter of a specific voxel size. This size has been set to $0.03m$, to ensure the pipeline is not slowed down. With this, we can assure that the points that are part of an object will have the same point density spacing of $0.03m$. This parameter determines that the value threshold of the Euclidean cluster can be set close to $0.03m$. Different tests with $0.02m$, $0.03m$, $0.04m$, and $0.05m$ have been carried out to determine the best alternative. Additionally, the minimum cluster size has been set to 30, to ensure no noise from the real sensor is captured.

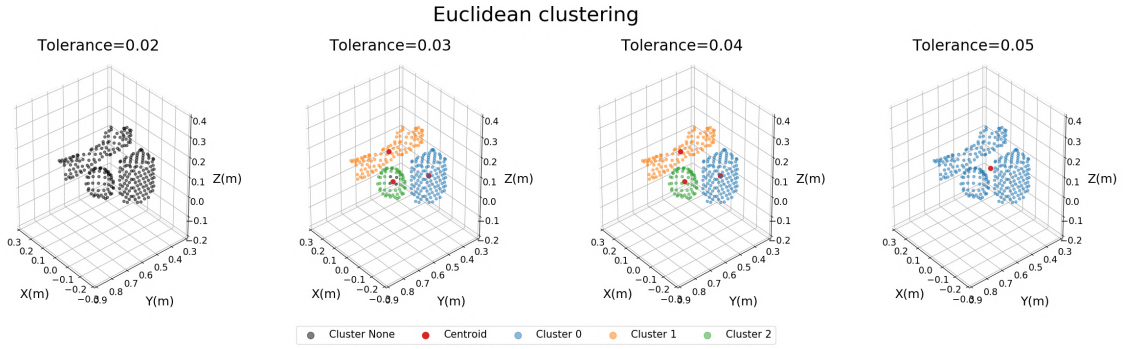


Figure 6.3: Clustering with close objects.

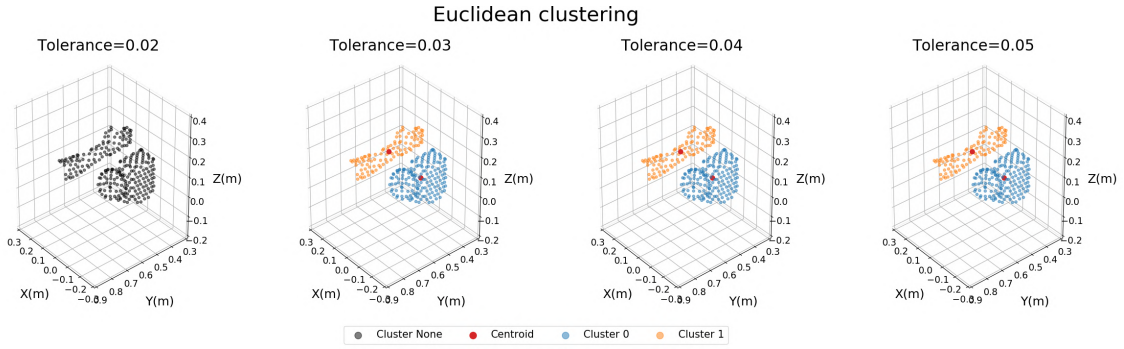


Figure 6.4: Clustering with contact objects.

By looking at the results in Figure 6.3, it can be observed that values lower to the voxel size produce no clusters (or neither of them reaches the minimum size), the value of 0.03, left some point from the objects unclustered (this effect is intensified with the real sensor, as the noise generates more room for uneven separation with bigger gaps than the voxel size), then the 0.04 value works fine and finally the 0.05 value groups objects which are close. Results from and Figure 6.4 are also very similar, except for the objects in contact, which are considered one by the algorithm with all the tested tolerances.

This grouping does not affect the avoidance performance, as the avoidance algorithm not only accounts for the centroid but also for the closest point to the trajectory; thus, by making a group and relocating the centroid, it is just creating a new obstacle with a different shape.

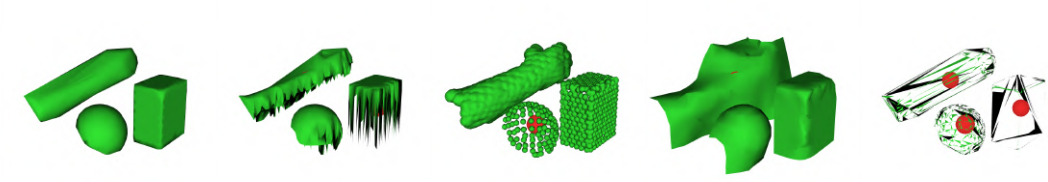
Finally, during real and simulated tests, different objects of different shapes, materials, and colors have been used to ensure that the system can provide robust feedback on any object that enters the workspace.

6.1.3 Mesh reconstruction

This final part of the algorithm is key for adequately integrating the real-world object into the virtual workspace. The different algorithms mentioned during the implementation have been evaluated following these features:

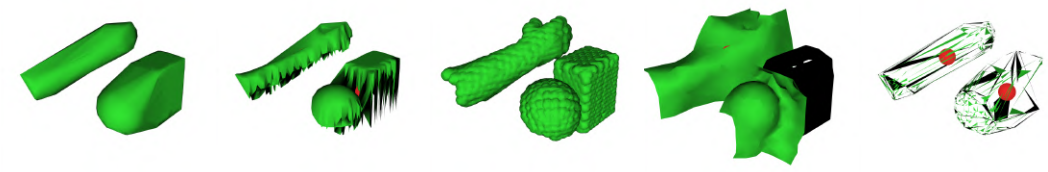
- Speed.
- Mesh resolution.
- Mesh holes/unconnected faces.

The mesh computations' starting point is the voxelized clusters with a voxel size of 0.03, and the obtained meshes of the evaluation setup are Figure 6.5 and Figure 6.6.



(a) Delaunay3D (b) Delaunay2D (c) GaussianSplat (d) SFUP (e) ConvexHull

Figure 6.5: Mesh reconstruction, close objects.



(a) Delaunay3D (b) Delaunay2D (c) GaussianSplat (d) SFUP (e) ConvexHull

Figure 6.6: Mesh reconstruction, contact objects.

In addition, the average computation time of each algorithm in a 10-sample test is detailed in Table 6.1.

Algorithm	Time [s]
Delaunay3D	0.02
Delaunay2D	0.015
GaussianSplat	0.15
ConvexHull	0.3
SurfaceFromUnorganizedPoints (SFUP)	0.015

Table 6.1: Mesh reconstruction execution time.

The mesh results show that the best alternatives are the Delaunay3D and the Gaussian Splat, as they reconstruct all the points of the cloud without adding/losing much information. However, in the case of the Gaussian Splat, it can be observed that the system cannot create a closed mesh for some object shapes. Moreover, the Delaunay3D is one order of magnitude faster between the two algorithms. Thus, despite not having the best resolution, it will be used in the avoidance pipeline, as it is the priority to achieve faster responses than having the exact object shape reconstructed. It is important to note that despite not having the exact object shape, all the cluster points are kept inside the mesh, ensuring that no object parts are missed when collisions are computed.

6.2 Evaluation of the obstacle avoidance

To thoroughly evaluate the obstacle avoidance capabilities, each of the steps in the pipeline has been evaluated independently and in conjunction.

6.2.1 Collision check

The collision check algorithm evaluation has only focused on checking the detection response time, as it has few parameters to adjust.

This algorithm takes, on average, $0.1s$ at the start, but as the robot moves and the current trajectory point increases, the number of future points to collision check is reduced, and so is the computation time. However, these results are far from ideal, as the system needs to match the workspace refresh rate of $30Hz$ at least. A sampling code has been added to only collision check states where the end effector position has changed a certain amount. This code allows to check collisions at $30Hz$ when the distance step is set to 0.02

The collision threshold will be adjusted in the trajectory generation section, as the algorithm's capabilities will determine this distance. It is important to note that there is some robot state drift between the collision detection, robot stop signal sending, and the actual robot stop position. This drift will require adding a sleep before reading the current robot state and the addition of it into the collision threshold.

6.2.2 Trajectory generation

In order to evaluate the obstacle avoidance capabilities, a simpler virtual experimental setup has been created with a single object positioned in a collision course with the end-effector. Obstacle dimensions and positions are changed during the evaluation.

For the first test of the avoidance system, a simple demo trajectory that emulates a pick-and-place task is fed alongside cubic obstacles of different sizes with the same centroid position. As stated by the previous work, the number of bases can be fixed to 50, and D is computed a $2\sqrt{K}$ to ensure a critically damped system. With obstacle sizes of $0.1 \times 0.1 \times 0.1m$, $0.2 \times 0.2 \times 0.2m$ and $0.3 \times 0.3 \times 0.3m$, and the centroid placed at $[-0.1, 0.35, 0.25]m$.

First, the system parameters are adjusted to avoid the $0.1 \times 0.1 \times 0.1m$ obstacle by only applying the original relative centroid distance and angle coupling. The parameters are detailed in Table 6.2.

dt	K	γ_{obs}	β_{obs}	k_{obs}	γ_{np}	β_{np}	k_{np}	γ_d	k_d
0.01	100	400.0	4.0	2.0	0	0	0	0	0

Table 6.2: Cartesian DMP parameters. Test 1

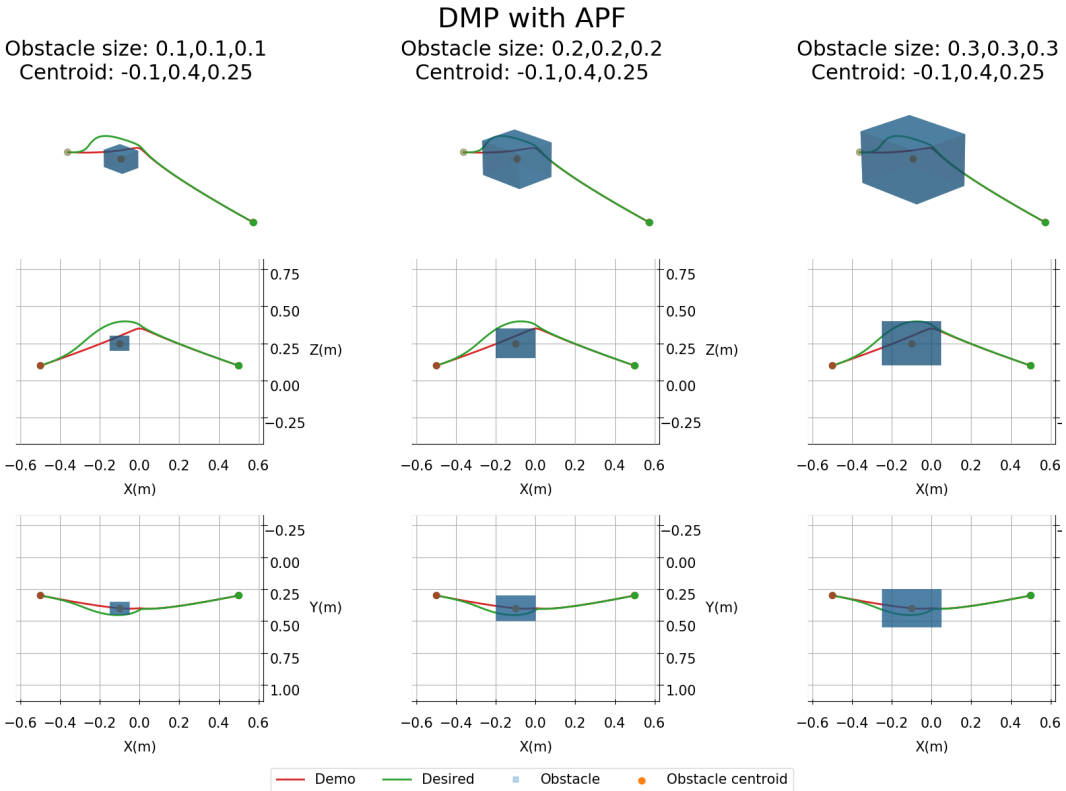


Figure 6.7: Avoidance trajectory. Test 1.

The obtained results, Figure 6.7, for all the size combinations, show that the system cannot scale the coupling correctly when the object size is changed, and the generated trajectory does not avoid the collision. Also, it is important to denote that K is in control of the DMP attraction force to the demo and that after some tests, it has been set to 100, ensuring that the system returns to the demo trajectory as soon as the obstacle is avoided.

The nearest point distance with angle coupling is introduced and adjusted to increase the scalability, allowing the generation of an avoiding trajectory for all the cases. Additionally, the previously defined parameters have been modified to keep the results as optimal as possible while adding the new coupling term. The new parameters are Table 6.3.

dt	K	γ_{obs}	β_{obs}	k_{obs}	γ_{np}	β_{np}	k_{np}	γ_d	k_d
0.01	100	200.0	4.0	2.0	300.0	2.5	5.0	0	0

Table 6.3: Cartesian DMP parameters. Test 2

In this case, as seen in Figure 6.8, the system can generate an avoiding trajectory in all cases. However, the margin between the end effector and the obstacle is reduced as the obstacle size increases. Moreover, the trajectory overshoots the target when avoiding the bigger obstacle and then returns. This effect should not be a problem for most cases, but as the trajectory approaches robot physical limitations, it can produce a non-valid result.

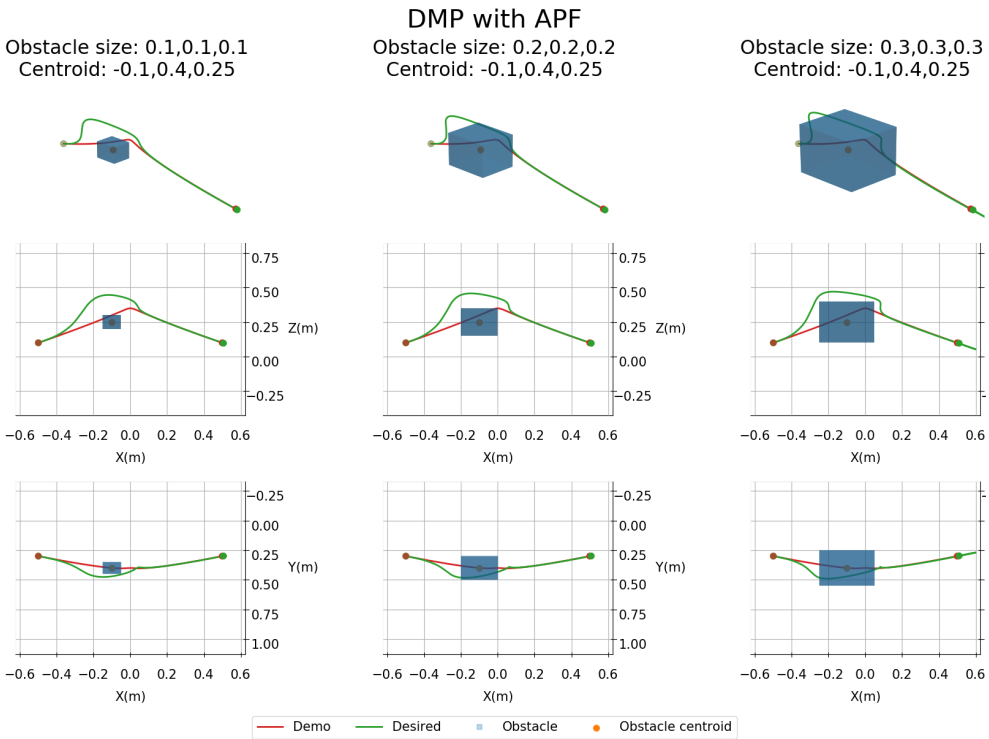


Figure 6.8: Avoidance trajectory. Test 2.

The relative nearest point distance factor that does not account for the velocity angle is added to the coupling term to mitigate the margin reduction effect. The final parameters are Table 6.4.

dt	K	γ_{obs}	β_{obs}	k_{obs}	γ_{np}	β_{np}	k_{np}	γ_d	k_d
0.01	100	200.0	4.0	2.0	300.0	2.5	5.0	30.0	20.0

Table 6.4: Cartesian DMP parameters. Test 3

The obtained results, Figure 6.9, for this parameter configuration generates an avoiding trajectory with enough collision margin for all the obstacle sizes. But due to the addition and tuning of all the coupling terms, it can be observed that the trajectory generated for the smallest obstacle is less optimal than the one obtained in the first test.

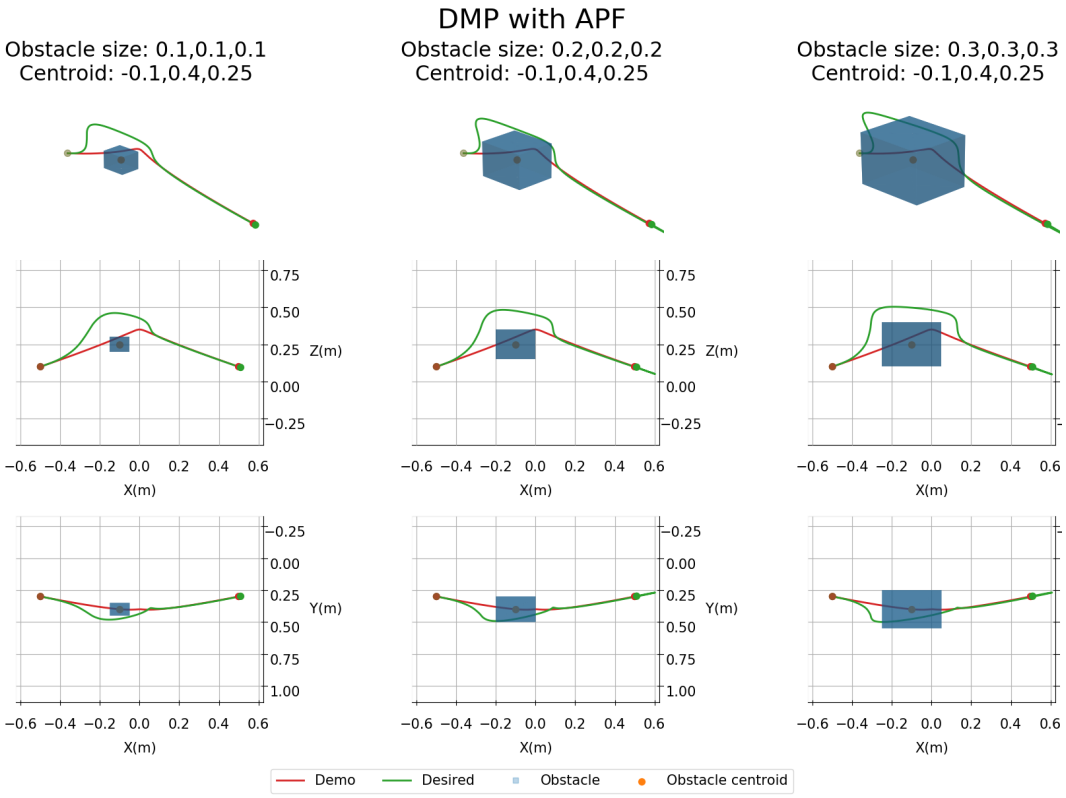


Figure 6.9: Avoidance trajectory. Test 3.

Finally, to reduce the coupling term effect directly, the proportional size scaling has been added, keeping the same configuration as before, Table 6.4, and the scaling parameters m and n, are set to 2.0 and 0.4 respectively.

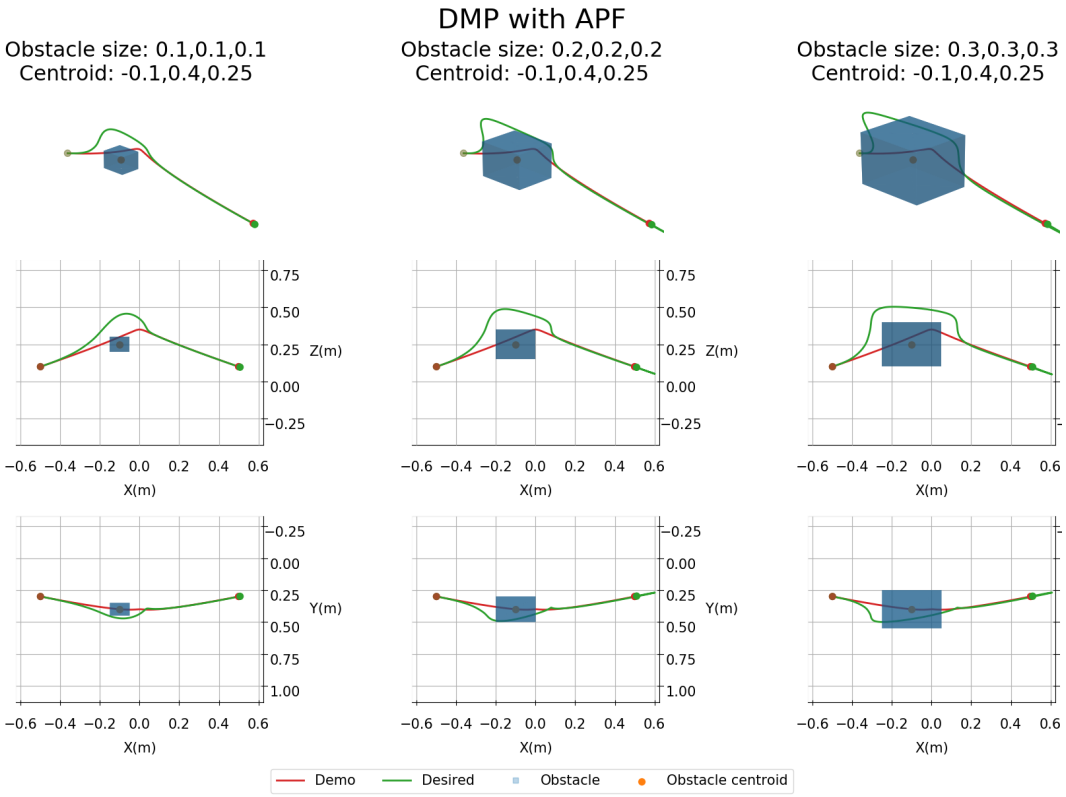


Figure 6.10: Avoidance trajectory. Test 4.

From Figure 6.10, it can be seen that the system deviates less from the demo trajectory with the small object, making it more optimal in the demo trajectory following the task.

Another trajectory has been used to test the system further. The obstacle's position has also been changed to $[-0.1, 0.45, 0.35]m$. This change does not seem to be much, but it moves the object from below the trajectory to above, which produces a big change in the coupling term computation. All the rest of the configurations have been kept the same.

The obtained results, Figure 6.11, show that the system can still handle the generation of a non-colliding trajectory. In this case, the trajectory produced is below the object, which is the robot and starts from a certain joint state, which can trigger collision detection with a robot part different from the end-effector. However, as handling full robot avoidance is not in the scope of this thesis, the robot will be driven into a safe-stop wait state until the obstacle is moved.

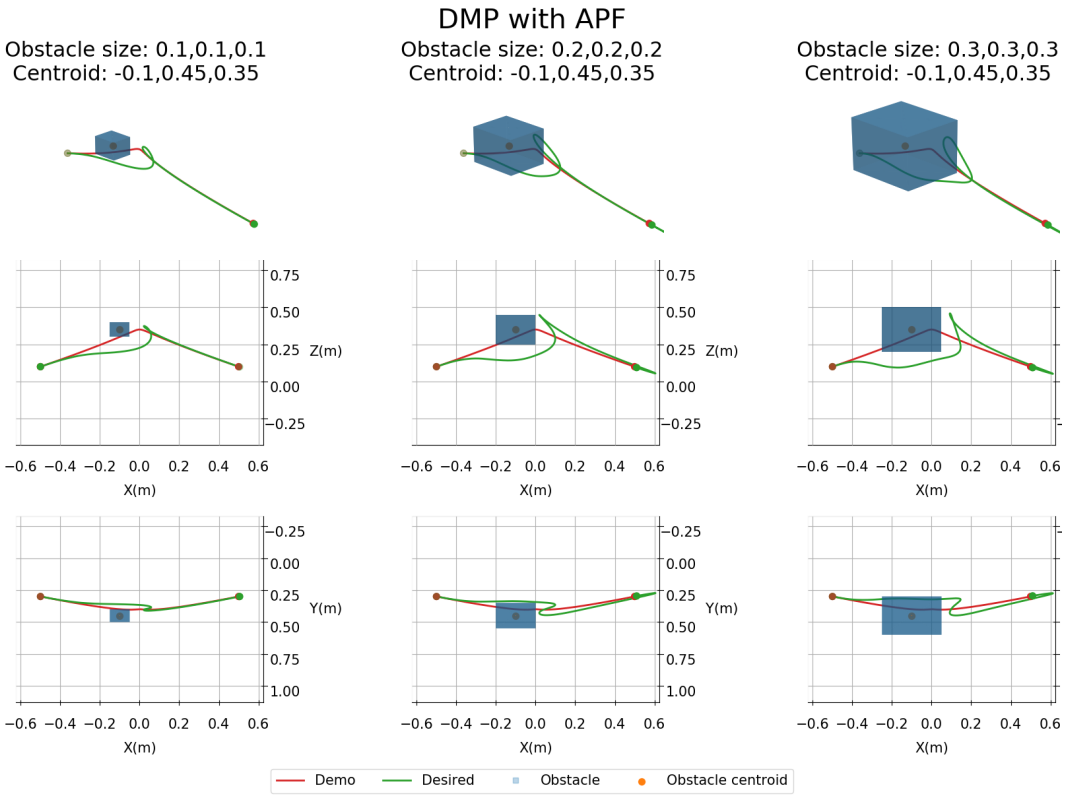


Figure 6.11: Avoidance trajectory. Test 5.

Finally, some edge cases have been tested to evaluate the system's limitations. The first case is set so that the obstacle is close to the starting point. In the second case, the obstacle is very close to the target point, and in the last case, the target point will be inside the obstacle volume.

By recalling the definition of the equations of the potential field, it can be expected that the system will not be able to generate sufficient repulsion in the given collision distance margin in the first case. Similarly, the generated repulsion will overshoot the target in the second case. Finally, if the obstacle volume overlaps the target, the system cannot converge to the point.

All previous hypotheses have been confirmed with Figure 6.12. Where the obstacle is avoided by moving the end-effector towards the base of the robot

The case where the obstacle overlaps with the target can be handled in the full application, as the system searches if the overlap is present and it triggers a safe-stop wait state, preventing the obstacle avoidance trajectory computation until the obstacle is moved, as it considers the goal to be unreachable.

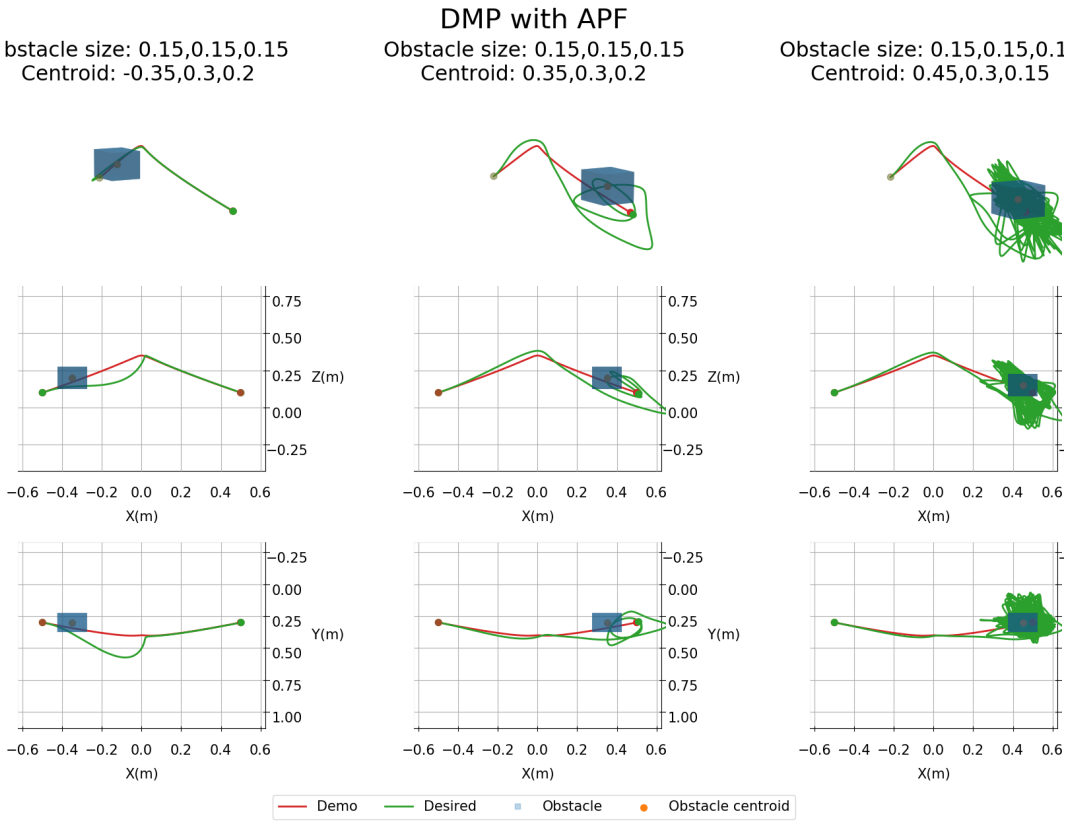


Figure 6.12: Avoidance trajectory edge cases. Test 6.

The last test has been carried out with a different trajectory to validate that the parameters work across trajectories. This new trajectory is like a slope in which the robot will move top-down.

Results can be seen in Figure 6.13. It has been confirmed that the system can handle different trajectories. However, it is essential to note that these parameters will still need some adjustments to achieve optimal performance (closest avoidance to demo trajectory) in some cases where the combination demo trajectory-obstacle is unfavorable.

For the cases where the generated trajectory still collides with the object due to parameters tuning or object movement, the system will try to move again after the first avoidance computation, it will detect the new collision, and it will generate a new avoidance trajectory starting from the last avoidance trajectory as a demo. This effect will be shown in section 6.3.

So it can be concluded that the parameters from Table 6.4, $m = 2.0$ and $n = 0.4$, will work for most cases.

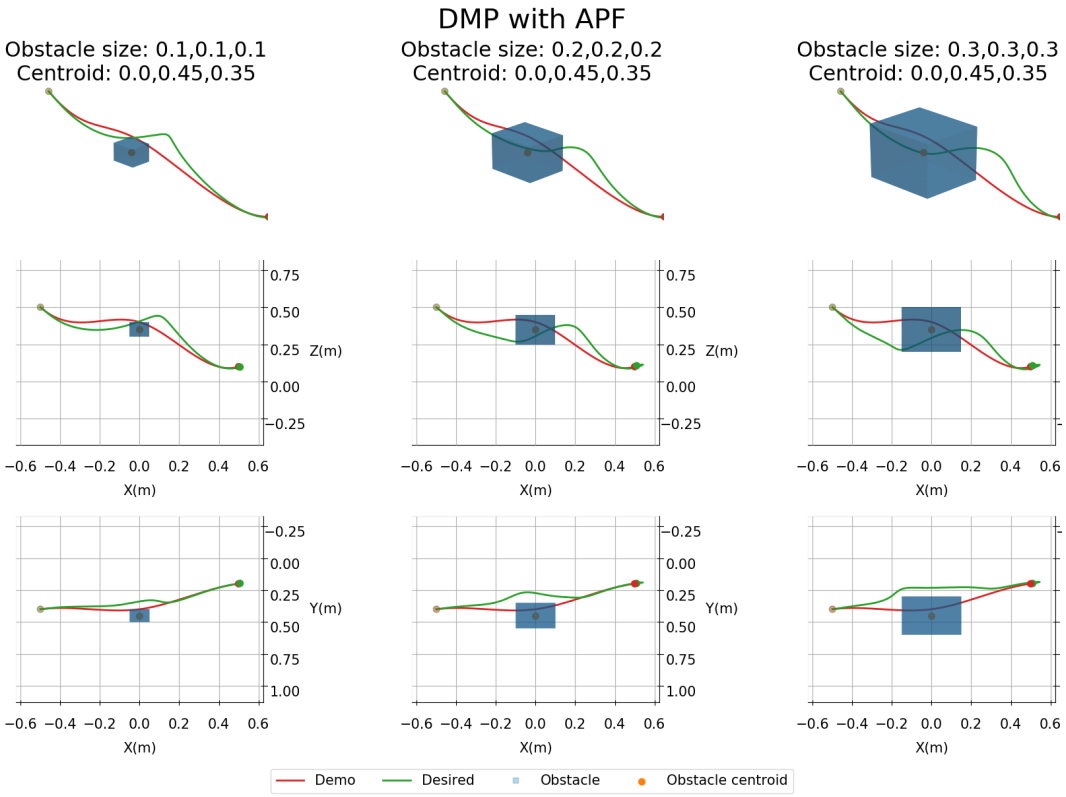


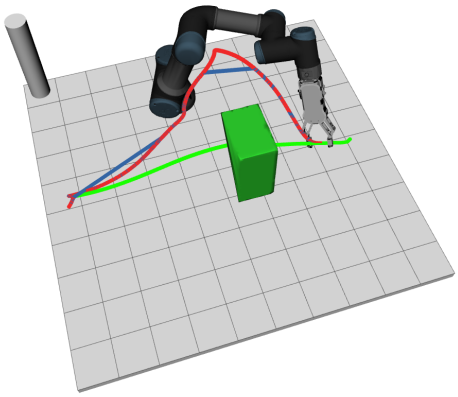
Figure 6.13: Avoidance trajectory. Test 7.

Finally, the generation time has been measured to conclude the trajectory generation evaluation. The current configuration takes an average of $0.06s$ to generate a response, which is still slower than the defined 30Hz in the requirements. However, this time can be reduced by increasing the DMP dt parameter from 0.01 to 0.1, which reduces the request time to $0.01s$, fulfilling the requirements. This change in the dt laid almost identical trajectory results as the ones detailed above.

6.2.3 Evaluation of inverse kinematics

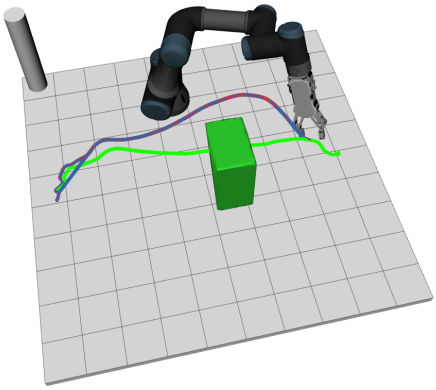
The conversion between Cartesian and Joint space takes, on average, 1s, which means that the system won't be able to generate a real-time response, as stated in the requirements.

When evaluating the conversion from the Cartesian trajectory to the Joint space, it has been observed that in some cases where the avoidance trajectory is further from the original demo trajectory, the system cannot find a solution in certain regions. This error occurs because the end-effector poses are out of reach when starting from the current joint configuration.



Demo (Green)
Cartesian DMP
Cartesian DMP converted with IK (Blue)

Figure 6.14: IK limits reached



Demo (Green)
Cartesian DMP
Cartesian DMP converted with IK (Blue)

Figure 6.15: IK no limits reached.

6.3 Full application results

Multiple videos have been created to showcase the obstacle avoidance system working with the real robot setup.

The speed and wait times of the robot have been adjusted to ensure safety during execution. However, the system is still capable of moving faster and without much delay between collision detection and avoidance.

Figure 6.16 video shows the collision avoidance with a cardboard box.

Figure 6.17 video shows collision avoidance with a hand/arm.

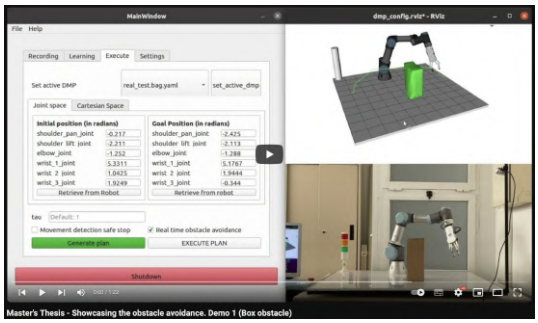


Figure 6.16: Video demo 1 (https://youtu.be/nNWK6TriPfU).

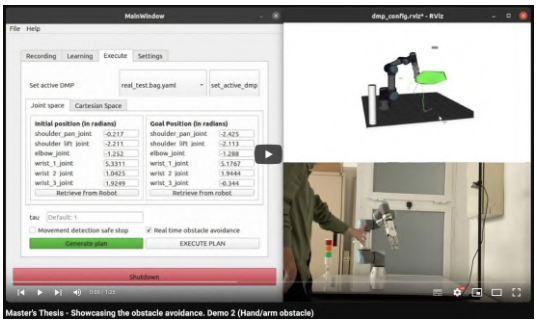


Figure 6.17: Video demo 2 (https://youtu.be/xKeDVJG8ucA).

Figure 6.18 video shows collision avoidance with a bottle. In this video, the reaching of the limits of the IK solver can be seen.

Figure 6.19 video shows the collision avoidance with a bottle in the same configuration as before, but the DMP parameters were adjusted to avoid the limit reaching.

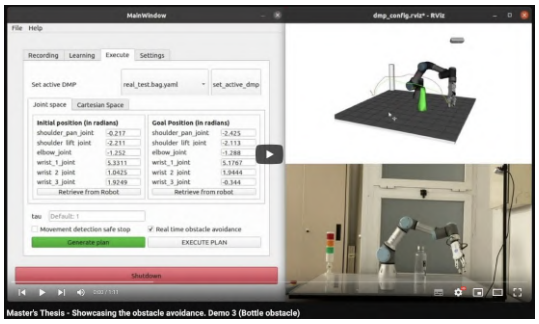


Figure 6.18: Video demo 3 (https://youtu.be/MBp5vqZ0TR8).

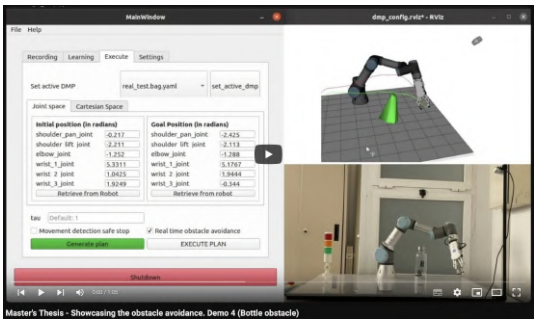


Figure 6.19: Video demo 4 (https://youtu.be/m7B9dGOFJkk).

Figure 6.20 video shows collision avoidance with the hand/arm. In this video, it can be seen how the system makes a second collision avoidance trajectory computation due to the hand moving slightly toward the new trajectory, triggering a new collision detection.

Figure 6.21 sixth video shows the feedback in the original app collision check and safe stop feature. Where the collision check prevents execution when the obstacle is in a collision course with the robot and a moving hand is detected, and the execution stops.

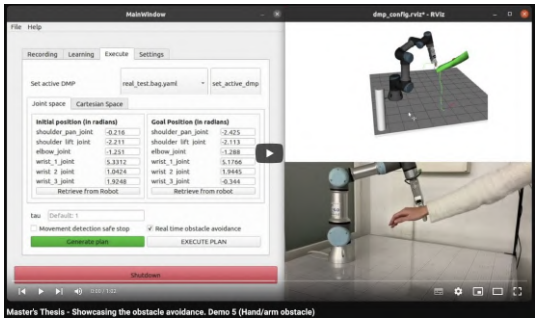


Figure 6.20: Video demo 5 (https://youtu.be/ZJFb8pUomdQ).

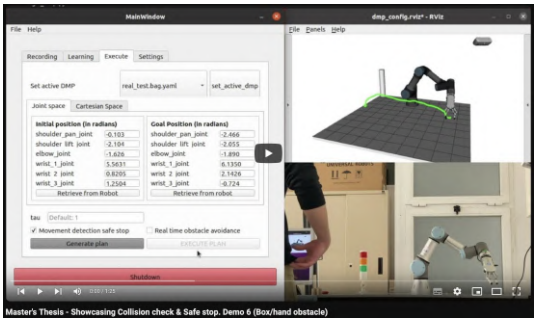


Figure 6.21: Video demo 6 (https://youtu.be/SP4JaBKoEi8).

Figure 6.22 video shows collision avoidance with a moving hand/arm.

Figure 6.23 video shows collision avoidance with two contacting objects. This video validates the previous reasoning that two objects acting as one does not affect the avoidance result.

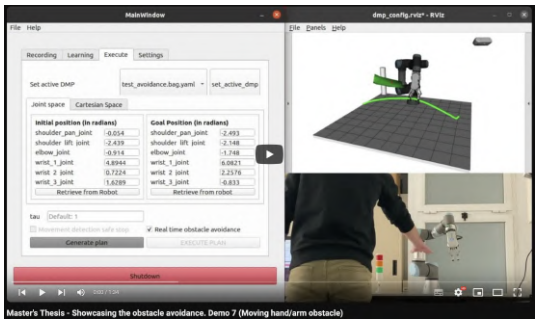


Figure 6.22: Video demo 7 (https://youtu.be/mYGpvaMRMZg).

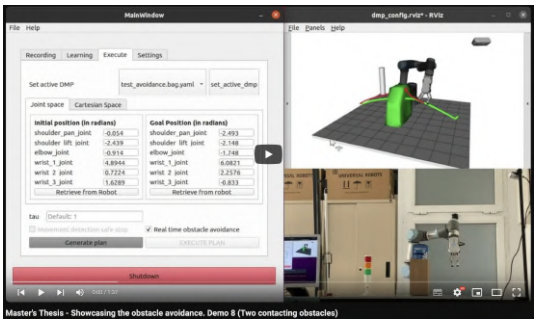


Figure 6.23: Video demo 8 (https://youtu.be/HnJg-f-7IJE).

Chapter 7

Final considerations

7.1 Planning and Scheduling

The tasks of this thesis have been distributed according to the diagram presented in Figure 7.1.

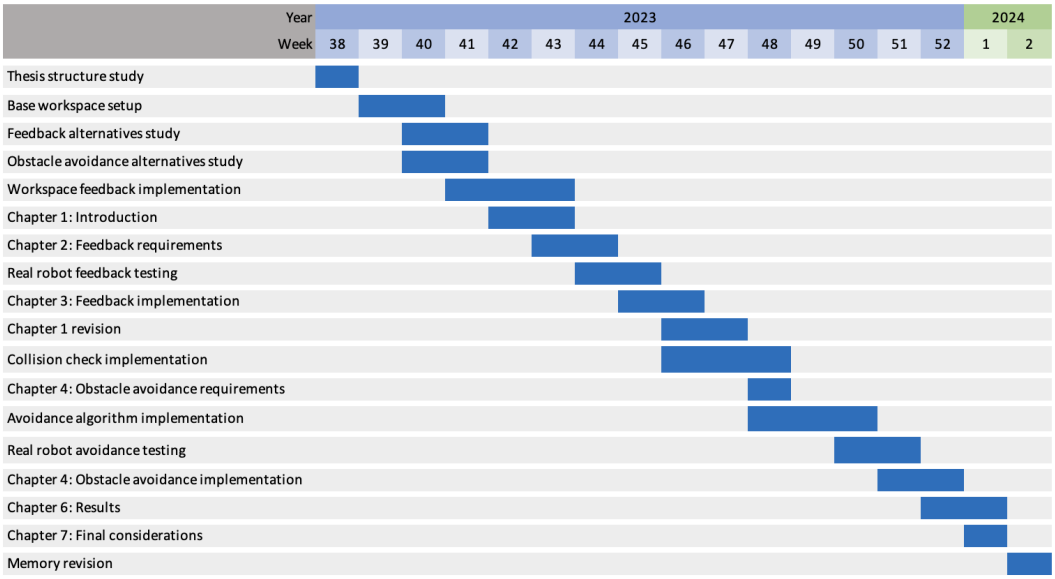


Figure 7.1: Gantt diagram.

For this thesis, a total of 388 hours have been used, divided into the following sections:

Sections	Hours
Research	50
Software development	138
Testing	50
Report	150

Table 7.1: Hours distribution.

7.2 Budget

This section will show an overview of the study's most relevant costs, representing only the direct costs of the parts needed to reproduce the setup.

Part	Price [€]
UR3 CB	23,000
HP EliteDesk 700 G1	208
Intel Realsense D345	520
Custom-made camera support	20
Total cost	23,748

Table 7.2: Estimated costs.

7.3 Conclusions

The results of this work show that the developed system can detect collisions with any part of the robot and avoid collisions with the end-effector while following a demonstrated motion, both in a virtual environment and on the real robot.

All the experiments have been carried out to ensure that the system can detect collision with any object independently of its type and to adjust the parameters of the implemented obstacle avoidance code implemented on top of the DMP library of Scott Niekum to work with the UR3 robot. The tune parameters are set as default values of the application inside the ***obstacle_avoidance.launch*** file.

The main goals of this work were to generate workspace feedback and implement obstacle avoidance capabilities to the existing work, [27]. Both of these objectives have been fulfilled by using a depth camera and creating a collision detection pipeline, which allowed the use of this workspace feedback to generate a new trajectory with a collision avoidance pipeline. As a result, the secondary objective of making the system safer for the human operator has also been achieved. Additionally, the base work GUI has been extended to accommodate the new obstacle avoidance capabilities, and the workspace feedback has been added to the existing collision check pipeline for the non-avoiding run case.

The performance requirements have been achieved by updating the virtual workspace, detecting collisions at a rate of $\sim 30Hz$, and generating an avoidance trajectory for objects that block the end-effector path. Other non-end-effector collisions are detected, and the robot is stopped, but no avoidance is triggered, as this was out of the scope of this thesis.

This work continues with the open source policy stated in the base work to encourage further development of cooperative robot applications in industrial environments.

7.4 Comments and further development

The system has been tested and performs well in most cases with the real UR3 robot. However, improvements can be made to enhance the system's performance. These improvements, in combination with some comments, are detailed in this section.

First, the IK conversion part of the code has been a significant limitation when developing the work, as the available UR implementation of ***MoveIt*** had many result generation issues, and the real test depended on it. A revision of this part of the code is required, and maybe the use of another solution should be implemented in the future.

For the rest of the pipeline, there is still room for optimization of the code execution times to increase the workspace update rate and, in consonance, allow working in more complex environments with faster-moving objects.

The tuning of obstacle avoidance parameters has been a time-consuming task that undoubtedly led to a suboptimal solution. In future work, this adjustment can be made using a reinforcement learning system or even changing the coupling term generator to a neural network. This modification can also help with the addition of complete robot body obstacle avoidance.

Finally, despite not being a major issue, the object detection pipeline lacks discrimination capabilities when two objects are in contact. This misclassification does not affect the system performance, but it could be good to make this differentiation for some particular applications where the object is relevant to the task.

Bibliography

[1] Oussama Khatib. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots". In: *The International Journal of Robotics Research* 5.1 (1986), pp. 90–98. DOI: 10.1177/027836498600500106. eprint: https://doi.org/10.1177/027836498600500106. URL: https://doi.org/10.1177/027836498600500106. (accessed: 11.11.2023).

[2] Arati S. Deo and Ian D. Walker. "Overview of damped least-squares methods for inverse kinematics of robot manipulators". In: *Journal of Intelligent and Robotic Systems* 14.1 (Sept. 1995), pp. 43–68. ISSN: 1573-0409. DOI: 10.1007/BF01254007. URL: https://doi.org/10.1007/BF01254007. (accessed: 12.11.2023).

[3] Vijay Konda and John Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf. (accessed: 12.11.2023).

[4] Kurt M. DeGoede et al. "How Quickly Can Healthy Adults Move Their Hands to Intercept an Approaching Object? Age and Gender Effects". In: *The Journals of Gerontology: Series A* 56.9 (Sept. 2001), pp. M584–M588. ISSN: 1079-5006. DOI: 10.1093/gerona/56.9.M584. eprint: https://academic.oup.com/biomedgerontology/article-pdf/56/9/M584/9732755/M584.pdf. URL: https://doi.org/10.1093/gerona/56.9.M584. (accessed: 1.11.2023).

[5] Yufei Tao et al. "Prediction and Indexing of Moving Objects with Unknown Motion Patterns". In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. SIGMOD '04. Paris, France: Association for Computing Machinery, 2004, pp. 611–622. ISBN: 1581138598. DOI: 10.1145/1007568.1007637. URL: https://doi.org/10.1145/1007568.1007637. (accessed: 11.11.2023).

[6] Chrislb. *Artificial Neuron Model*. June 2005. URL: https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png. (accessed: 14.11.2023).

[7] Mark Lee. *6.6 Actor-Critic Methods*. incompleteideas, Apr. 2005. URL: http://incompleteideas.net/book/ebook/node66.html. (accessed: 12.11.2023).

[8] Stefan Schaal. "Dynamic Movement Primitives–A Framework for Motor Control in Humans and Humanoid Robotics". In: *Adaptive Motion of Animals and Machines* (Jan. 2006). DOI: 10.1007/4-431-31381-8_23. (accessed: 11.11.2023).

[9] Peter Pastor et al. "Learning and generalization of motor skills by learning from demonstration". In: (2009), pp. 763–768. DOI: 10.1109/ROBOT.2009.5152385. (accessed: 11.11.2023).

[10] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. "Learning Policy Improvements with Path Integrals". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 828–835. URL: https://proceedings.mlr.press/v9/theodorou10a.html. (accessed: 11.11.2023).

[11] William Uther. "Markov Decision Processes". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 642–646. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_512. URL: https://doi.org/10.1007/978-0-387-30164-8_512. (accessed: 11.11.2023).

[12] Raul Acuña et al. "Dynamic Potential Field Generation Using Movement Prediction". In: July 2012. ISBN: 978-981-4415-94-1. DOI: 10.1142/9789814415958_0098. (accessed: 11.11.2023).

[13] Auke Jan Ijspeert et al. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". In: *Neural Computation* 25.2 (2013), pp. 328–373. DOI: 10.1162/NECO_a_00393. (accessed: 11.11.2023).

[14] Robert Krug and Dimitar Dimitrov. "Model Predictive Motion Control based on Generalized Dynamical Movement Primitives". In: *Journal of Intelligent & Robotic Systems* 77.1 (Jan. 2015), pp. 17–35. ISSN: 1573-0409. DOI: 10.1007/s10846-014-0100-3. URL: https://doi.org/10.1007/s10846-014-0100-3. (accessed: 11.11.2023).

[15] Scott Niekum. *dmp*. 2016. URL: https://github.com/sniekum/dmp. (accessed: 10.9.2023).

[16] *UR3 Technical specifications*. universal-robot, 2016. URL: https://www.universal-robots.com/media/240787/ur3_us.pdf. (accessed: 1.11.2023).

[17] Dominique Guerillot and Jeremie Bruyelle. "Uncertainty Assessment in Production Forecast with an Optimal Artificial Neural Network". In: Mar. 2017. DOI: 10.2118/183921-MS. (accessed: 14.11.2023).

[18] et al. Prof. Dr. Marco Hutter. *Programming for Robotics: Introduction to ROS*. ETH Züric, Feb. 2017. URL: https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/ROS2017/lecture1.pdf. (accessed: 6.11.2023).

[19] Giuseppe Fedele et al. "Obstacles Avoidance Based on Switching Potential Functions". In: *Journal of Intelligent & Robotic Systems* 90 (June 2018). DOI: 10.1007/s10846-017-0687-2. (accessed: 11.11.2023).

[20] Mingshan Chi et al. "Learning, Generalization, and Obstacle Avoidance with Dynamic Movement Primitives and Dynamic Potential Fields". In: *Applied Sciences* 9.8 (2019). ISSN: 2076-3417. DOI: 10.3390/app9081535. URL: https://www.mdpi.com/2076-3417/9/8/1535. (accessed: 6.11.2023).

[21] Bill Schweber Leave a Comment. *LiDAR and Time of Flight, Part 2: Operation*. Microcontroller tips, Dec. 2019. URL: https://www.microcontrollertips.com/lidar-and-time-of-flight-part-2-operation/. (accessed: 3.11.2023).

[22] Laura Graesser and Wah Loon Keng. "Reinforcement Learning - The Actor-Critic Algorithm". In: (Dec. 2019). URL: https://www.informit.com/articles/article.aspx?p=2995356&seqNum=5. (accessed: 12.11.2023).

[23] Satoki Hasegawa et al. "Electroholography of real scenes by RGB-D camera and the downsampling method". In: *OSA Continuum* 2.5 (May 2019), pp. 1629–1638. DOI: 10.1364/OSAC.2.001629. URL: https://opg.optica.org/osac/abstract.cfm?URI=osac-2-5-1629. (accessed: 5.11.2023).

[24] Mohammad Safeea, Pedro Neto, and Richard Bearee. "On-line collision avoidance for collaborative robot manipulators by adjusting off-line generated paths: An industrial use case". In: *Robotics and Autonomous Systems* 119 (Sept. 2019), pp. 278–288. DOI: 10.1016/j.robot.2019.07.013. URL: https://doi.org/10.1016%2Fj.robot.2019.07.013. (accessed: 2.11.2023).

[25] Leon Chlon. "tl;dr: Dirichlet Process Gaussian Mixture Models made easy." In: (Feb. 2020). URL: https://towardsdatascience.com/tl-dr-dirichlet-process-gaussian-mixture-models-made-easy-12b4d492e5f9. (accessed: 14.11.2023).

[26] Arsalan Anwar. *Part 2: 7 Simple Steps to Create and Build Your First ROS Package*. The Startup, Feb. 2021. URL: https://medium.com/swlh/7-simple-steps-to-create-and-build-our-first-ros-package-7e3080d36faa. (accessed: 6.11.2023).

[27] Roy Ove Eriksen. *Implementation and evaluation of movement primitives and a graphical user interface for a collaborative robot*. eng. Angulo Bahón, Cecilio, Jan. 2021. URL: http://hdl.handle.net/2117/340118.

[28] Ang Li et al. "Reinforcement Learning with Dynamic Movement Primitives for Obstacle Avoidance". In: *Applied Sciences* 11.23 (2021). ISSN: 2076-3417. DOI: 10.3390/app112311184. URL: https://www.mdpi.com/2076-3417/11/23/11184. (accessed: 2.11.2023).

[29] Haoxuan Li, Daoxiong Gong, and Jianjun Yu. "An obstacles avoidance method for serial manipulator based on reinforcement learning and Artificial Potential Field". In: *International Journal of Intelligent Robotics and Applications* 5.2 (June 2021), pp. 186–202. ISSN: 2366-598X. DOI: 10.1007/s41315-021-00172-5. URL: https://doi.org/10.1007/s41315-021-00172-5. (accessed: 2.11.2023).

[30] Matteo Saveriano et al. "Dynamic Movement Primitives in Robotics: A Tutorial Survey". In: (Feb. 2021). (accessed: 11.11.2023).

[31] Innoviz Technologies. *LiDAR vs Camera*. Youtube. Apr. 2021. URL: https://youtu.be/fjo00OLBzX0. (accessed: 3.11.2023).

[32] Ali Abdi, Mohammad Hassan Ranjbar, and Ju Hong Park. "Computer Vision-Based Path Planning for Robot Arms in Three-Dimensional Workspaces Using Q-Learning and Neural Networks". In: *Sensors* 22.5 (2022). ISSN: 1424-8220. DOI: 10.3390/s22051697. URL: https://www.mdpi.com/1424-8220/22/5/1697. (accessed: 6.11.2023).

[33] Pengzhan Chen et al. "A deep reinforcement learning based method for real-time path planning and dynamic obstacle avoidance". In: *Neurocomputing* 497 (2022), pp. 64–75. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.05.006. URL: https://www.sciencedirect.com/science/article/pii/S0925231222005367. (accessed: 2.11.2023).

[34] Azmi Haider and Hagit Hel-Or. "What Can We Learn from Depth Camera Sensor Noise?" In: *Sensors* 22.14 (2022). ISSN: 1424-8220. DOI: 10.3390/s22145448. URL: https://www.mdpi.com/1424-8220/22/14/5448. (accessed: 3.11.2023).

[35] Alican Mertan, Damien Jade Duff, and Gozde Unal. "Single image depth estimation: An overview". In: *Digital Signal Processing* 123 (Apr. 2022), p. 103441. DOI: 10.1016/j.dsp.2022.103441. URL: https://doi.org/10.1016%2Fj.dsp.2022.103441. (accessed: 3.11.2023).

[36] Anis Naema Atiyah Rafai, Noraziah Adzhar, and Nor Izzati Jaini. "A Review on Path Planning and Obstacle Avoidance Algorithms for Autonomous Mobile Robots". In: *Journal of Robotics* 2022 (Dec. 2022), p. 2538220. ISSN: 1687-9600. DOI: 10.1155/2022/2538220. URL: https://doi.org/10.1155/2022/2538220. (accessed: 10.11.2023).

[37] *(Mindmap) A Hardcore Look at 9 types of LiDAR systems*. Think Autonomous, June 2023. URL: https://www.thinkautonomous.ai/blog/types-of-lidar/. (accessed: 3.11.2023).

[38] Jaafar Ahmed Abdulsaheb and Dheyaa Jasim Kadhim. "Classical and Heuristic Approaches for Mobile Robot Path Planning: A Survey". In: *Robotics* 12.4 (2023). ISSN: 2218-6581. DOI: 10.3390/robotics12040093. URL: https://www.mdpi.com/2218-6581/12/4/93. (accessed: 10.11.2023).

[39] Teham Bhuiyan et al. *Deep-Reinforcement-Learning-based Path Planning for Industrial Robots using Distance Sensors as Observation*. 2023. arXiv: 2301.05980 [cs.RO]. (accessed: 2.11.2023).

[40] Giovanni Boschetti et al. "3D collision avoidance strategy and performance evaluation for human–robot collaborative systems". In: *Computers & Industrial Engineering* 179 (2023), p. 109225. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2023.109225. URL: https://www.sciencedirect.com/science/article/pii/S0360835223002498. (accessed: 2.11.2023).

[41] Yu Chen et al. "Research on Real-Time Obstacle Avoidance Motion Planning of Industrial Robotic Arm Based on Artificial Potential Field Method in Joint Space". In: *Applied Sciences* 13.12 (2023). ISSN: 2076-3417. DOI: 10.3390/app13126973. URL: https://www.mdpi.com/2076-3417/13/12/6973. (accessed: 6.11.2023).

[42] *Different Types of Proximity Sensors & Their Applications*. Geya, Mar. 2023. URL: https://www.geya.net/different-types-of-proximity-sensors/. (accessed: 3.11.2023).

[43] Sha Luo and Lambert Schomaker. "Reinforcement learning in robotic motion planning by combined experience-based planning and self-imitation learning". In: *Robotics and Autonomous Systems* 170 (2023), p. 104545. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2023.104545. URL: https://www.sciencedirect.com/science/article/pii/S0921889023001847. (accessed: 2.11.2023).

[44] *Should the camera be fixed or robot-mounted?* Pick-It N.V, 2023. URL: https://docs.pickit3d.com/en/3.1/optimize-your-application/hardware/faq-which-camera-mount.html. (accessed: 3.11.2023).

[45] *Solara designated Certified Systems Integrator for Universal Robots*. Solara Automation, 2023. URL: https://solaraautomation.com/solara-designated-certified-systems-integrator-for-ur/. (accessed: 1.11.2023).

[46] *UR3*. universal-robot, 2023. URL: https://www.universal-robots.com/products/ur3-robot/. (accessed: 24.10.2023).

[47] Yanzhe Wang et al. "An online collision-free trajectory generation algorithm for human–robot collaboration". In: *Robotics and Computer-Integrated Manufacturing* 80 (2023), p. 102475. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2022.102475. URL: https://www.sciencedirect.com/science/article/pii/S0736584522001570. (accessed: 2.11.2023).

[48] Wikipedia. *Artificial neural network — Wikipedia, The Free Encyclopedia*. 2023. URL: http://en.wikipedia.org/w/index.php?title=Artificial%5C%20neural%5C%20network&oldid=1183756198. (accessed: 14.11.2023).

[49] Wikipedia. *Inertial measurement unit — Wikipedia, The Free Encyclopedia*. 2023. URL: http://en.wikipedia.org/w/index.php?title=Inertial%5C%20measurement%5C%20unit&oldid=1177504327. (accessed: 3.11.2023).

[50] Wikipedia. *k-d tree — Wikipedia, The Free Encyclopedia*. 2023. URL: https://en.wikipedia.org/wiki/K-d_tree. (accessed: 5.11.2023).

[51] Wikipedia. *Motion planning — Wikipedia, The Free Encyclopedia*. 2023. URL: http://en.wikipedia.org/w/index.php?title=Motion%5C%20planning%5C&oldid=1172480989. (accessed: 11.11.2023).

[52] Wikipedia. *Polygon mesh — Wikipedia, The Free Encyclopedia*. 2023. URL: https://en.wikipedia.org/wiki/Polygon_mesh. (accessed: 5.11.2023).

[53] *Construct a concave or convex hull polygon for a plane model*. Point Cloud Library. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/index.html#surface. (accessed: 5.11.2023).

[54] *Downsampling a PointCloud using a VoxelGrid filter*. Point Cloud Library. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/voxel_grid.html. (accessed: 5.11.2023).

[55] *Euclidean Cluster Extraction*. Point Cloud Library. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/cluster_extraction.html#cluster-extraction. (accessed: 5.11.2023).

[56] *Fast triangulation of unordered point clouds*. Point Cloud Library. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/greedy_projection.html#greedy-triangulation. (accessed: 5.11.2023).

[57] *Gazebo*. Open Robotics. URL: https://gazebosim.org/home. (accessed: 6.11.2023).

[58] *Intel RealSense Depth Camera D435*. Intel. URL: https://www.intelrealsense.com/depth-camera-d435/. (accessed: 4.11.2023).

[59] *Intel RealSenseTM D400 Series Product Family*. Intel. URL: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf. (accessed: 4.11.2023).

[60] *RG2 – FLEXIBLE 2 FINGER ROBOT GRIPPER WITH WIDE STROKE*. Onrobot. URL: https://onrobot.com/en/products/rg2-gripper. (accessed: 4.11.2023).

[61] *Robot Operating System*. Open Robotics. URL: https://www.ros.org/. (accessed: 6.11.2023).

[62] *ROS Noetic Ninjemys*. Open Robotics. URL: http://wiki.ros.org/noetic. (accessed: 6.11.2023).

[63] *RViz*. Open Robotics. URL: http://wiki.ros.org/rviz. (accessed: 6.11.2023).

[64] *shape_msgs*. ROS. URL: http://wiki.ros.org/shape_msgs. (accessed: 5.11.2023).

[65] *shape_msgs/Mesh Message*. ROS. URL: http://docs.ros.org/en/api/shape_msgs/html/msg/Mesh.html. (accessed: 5.11.2023).

[66] *sklearn.cluster.DBSCAN*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN. (accessed: 5.11.2023).

[67] Ioan A. Sucan and Sachin Chitta. *MoveIt*. PickNik Robotics. URL: moveit.ros.org. (accessed: 6.11.2023).

[68] *Surface*. Point Cloud Library. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/index.html#surface. (accessed: 5.11.2023).

[69] *Surface reconstruction*. Kitware. URL: https://examples.vtk.org/site/Cxx/#surface-reconstruction. (accessed: 5.11.2023).

[70] *USER MANUAL RG2 Industrial Robot Gripper*. Onrobot. URL: https://onrobot.com/sites/default/files/documents/RG2_User%20_Manual_enEN_V1.9.2.pdf. (accessed: 4.11.2023).

[71] *Visualization Toolkit*. Kitware. URL: https://vtk.org/. (accessed: 5.11.2023).

Appendix A

Code

All the code used in the development of this work, alongside with the setup and execution instructions, is available in GitHub:

https://github.com/Izan00/ur_obstacle_avoidance.git