Technical University of Crete
School of Electrical and Computer Engineering

Vision-Based Autonomous Robotic Arm for Pick-and-Stack Applications in the Gazebo-ROS Environment

Charalampos Theodorakis

Thesis Committee
Professor Michail G. Lagoudakis (ECE)
Professor Michalis Zervakis (ECE)
Associate Professor Georgios Chalkiadakis (ECE)

Chania, October 2022

Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Αυτόνομος Ρομποτικός Βραχίονας Βασισμένος σε Όραση για Εφαρμογές Pick-and-Stack στο Περιβάλλον Gazebo-ROS

Χαράλαμπος Θεοδωράκης

Εξεταστική Επιτροπή
Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)
Καθηγητής Μιχάλης Ζερβάκης (ΗΜΜΥ)
Αναπληρωτής Καθηγητής Γεώργιος Χαλκιαδάκης (ΗΜΜΥ)

Χανιά, Οκτώβριος 2022

Abstract

In recent years, the use of robots as a part of the automation process is a rapidly expanding field. The most prevalent type of robot in industrial environments is robotic arms. This type of robots is capable of performing a variety of operations, including transportation, assembly, packing and welding. The complexity of the tasks assumed for these growing disciplines is also rising. Thus, the ability of the robot to comprehend the surrounding environment is a required feature to carry out new tasks. Interacting with the environment and being aware of the changing conditions enables the robot to make autonomous decisions in more complex situations. This thesis describes the implementation of an autonomous 6-DOF (Degrees Of Freedom) robotic manipulator with visual guidance, where the objective for the robot arm is to stack cylinder blocks, recreating a gradually-presented structured tower pattern. The available blocks, initially placed randomly in the robot workspace, have unique ArUco markers displayed on them, so they can be identified. The manipulator is equipped with vacuum grippers and a 2D camera attached to its end effector. The robot acquires information about the desired tower pattern from a remote stationary camera, responsible to track the spawning of building blocks. Since the robotic arm is equipped with its own camera, it employs computer vision techniques to locate the desired cylinder and, consequently, constructs a trajectory to pick and stack that cylinder to its correct position using motion planning algorithms. The entire project has been implemented within the Robot Operating System (ROS) and Gazebo open-source 3D robotics simulator. The proposed robotic system has been tested extensively in simulations to ensure its reliability and investigate its efficiency.

Περίληψη

Τα τελευταία χρόνια, η χρήση ρομποτικών συστημάτων ως μέρος της διαδικασίας αυτοματισμού είναι ένας ταχέως αναπτυσσόμενος τομέας. Ο πιο διαδεδομένος τύπος ρομπότ σε βιομηχανικά περιβάλλοντα είναι οι ρομποτικοί βραχίονες. Αυτός ο τύπος ρομπότ είναι ικανός να εκτελεί μια ποικιλία λειτουργιών, όπως μεταφορά, συναρμολόγηση, συσκευασία και συγκόλληση. Η πολυπλοκότητα των εργασιών που αναλαμβάνονται για αυτούς τους αναπτυσσόμενους κλάδους αυξάνεται επίσης, επομένως η ικανότητα του ρομπότ να κατανοεί το περιβάλλον του είναι απαραίτητο χαρακτηριστικό για την εκτέλεση νέων εργασιών. Η αλληλεπίδραση με το περιβάλλον και η επίγνωση των μεταβαλλόμενων συνθηκών επιτρέπει στο ρομπότ να παίρνει αυτόνομες αποφάσεις σε πιο περίπλοκες καταστάσεις. Η παρούσα διπλωματική εργασία περιγράφει την υλοποίηση ενός αυτόνομου ρομποτικού χειριστηρίου 6-DOF (Degrees Of Freedom – Βαθμών Ελευθερίας) με οπτική καθοδήγηση, όπου ο στόχος είναι ο ρομποτικός βραχίονας να στοιβάζει κυλίνδρους, ανακατασκευάζοντας ένα πρότυπο δομημένου πύργου που παρουσιάζεται σταδιακά. Τα δομικά στοιχεία, αρχικά τοποθετημένα τυχαία στον χώρο εργασίας του ρομπότ, έχουν μοναδικούς δείκτες ArUco που εμφανίζονται σε αυτά, ώστε να μπορούν να αναγνωριστούν. Ο βραχίονας είναι εξοπλισμένος με άρπαγες κενού αέρος και μια κάμερα 2D συνδεδεμένη στην απόληξη του. Το ρομπότ αποκτά πληροφορίες σχετικά με το επιθυμητό πρότυπο πύργου από μια απομακρυσμένη σταθερή κάμερα, υπεύθυνη για την παρακολούθηση της διαδοχής των δομικών στοιχείων. Δεδομένου ότι ο ρομποτικός βραχίονας είναι εξοπλισμένος με δική του κάμερα, χρησιμοποιεί τεχνικές μηχανικής όρασης για να εντοπίσει τον επιθυμητό κύλινδρο και, στη συνέχεια, σχεδιάζει μια τροχιά για να σηκώσει και να στοιβάξει τον συγκεκριμένο κύλινδρο στη σωστή του θέση χρησιμοποιώντας αλγόριθμους σχεδιασμού κίνησης. Η εργασία στο σύνολό της έχει υλοποιηθεί χρησιμοποιώντας το Robot Operating System (ROS) και το περιβάλλον ρομποτικής προσομοίωσης Gazebo. Το προτεινόμενο ρομποτικό σύστημα έχει δοκιμαστεί εκτενώς σε προσομοιώσεις, για να διασφαλιστεί η αξιοπιστία του και να διερευνηθεί η αποτελεσματικότητά του.

Acknowledgements

First of all, I would like to thank my Professor Michail G. Lagoudakis for his guidance and advice throughout this project.

I would also like to thank my colleagues and friends for their assistance and support, which was essential for the completion of this work.

Last, but not least, I am grateful to my family for their continuous help and support.

Contents

1 Introduction 1
1.1 Thesis Contribution 2
1.2 Thesis Outline 3
2 Background 4
2.1 UR5 Robot Arm 4
2.2 Robot Operating System and Gazebo 7
2.3 MoveIt! 8
2.4 Kinematics 11
2.4.1 Forward Kinematics 11
2.4.2 Singularities 12
2.4.3 Inverse Kinematics 13
2.4.4 Inverse Jacobian Technique 16
2.5 Robot Vision and ArUco Markers 17
3 Problem Statement 19
3.1 Autonomous Vision Based Control For Robot Manipulator 19
3.2 Related Work 20
4 Our Approach 21
4.1 Simulation Environment 21
4.2 UR5 Robot Modeling 22
4.3 System Synchronization and management 25
4.4 ArUco Marker Detection 26
4.5 Robot Control and Motion Planning 30
5 Results 34

5.1 Simulation 35
5.2 Object Pose Estimation 41
6 Conclusions 42
6.1 Conclusion 42
6.2 Future work 43
6.2.1 Real Robot Application 43
6.2.2 Motion planners 43
7 References 44
8 Appendix A 48

List of Figures

Figure 2.1 : UR5 collaborative robot arm 4
Figure 2.2 : UR5 Technical Specifications 5
Figure 2.3 : A ROS message example 8
Figure 2.4 : Move_group Architecture 9
Figure 2.5 : Planning scene pipeline 10
Figure 2.6 : UR5 DH parameters 12
Figure 2.7 : Inverse Kinematics example of a 2-link robotic arm 14
Figure 2.8 : Comparison of some common Fiducial markers 17
Figure 2.9 : ArUco Marker with ID = 4 18
Figure 4.1 : Gazebo environment. Workbench 1 (left), Workbench 2 (right) 21
Figure 4.2 : Workspace 1, target stacks spawn 22
Figure 4.3 : UR5 Robot model equipped with the additional modules and the vacuum grippers configuration 23
Figure 4.4 : UR5 tf tree 24
Figure 4.5 : Tf tree view in RVIZ plugin 24
Figure 4.6 : FSM 25
Figure 4.7 : ArUco Marker ID=4 and the corresponding cylinder blender model 27
Figure 4.8 : Marker cells 28
Figure 4.9 : ArUco Markers' IDs detection by the UR5's 28
Figure 4.10 : Motion planning pipeline 31
Figure 4.11 : Start and Goal State 32
Figure 4.12 : Motion planning 33
Figure 5.1 : ROS Nodes Graph showing all nodes and topics 34
Figure 5.2 : Example 1, Simulation start 35
Figure 5.3 : Example 1, Iteration 1. Detection (left) Picking (center) Placing (right) 35

Figure 5.4 : Example 1, Iteration 2 35
Figure 5.5 : Example 1, Iteration 3 36
Figure 5.6 : Example 1, Iteration 4 36
Figure 5.7 : Example 1, Iteration 5 36
Figure 5.8 : Example 1, Iteration 6 36
Figure 5.9 : Example 1, Iteration 7 37
Figure 5.10 : Example 1, Iteration 8 37
Figure 5.11 : End of simulation (8 cylinders) 37
Figure 5.12 : Example 2, Iteration 1 38
Figure 5.13 : Example 2, Iteration 2 38
Figure 5.14 : Example 2, Iteration 3 38
Figure 5.15 : Example 2, Iteration 4 38
Figure 5.16 : Example 2, Iteration 5 39
Figure 5.17 : Example 2, Iteration 6 39
Figure 5.18 : Example 2, Iteration 7 39
Figure 5.19 : Example 2, Iteration 8 39
Figure 5.20 : Example 2, Iteration 9 39
Figure 5.21 : Example 2, Iteration 10 40
Figure 5.22 : End of simulation (10 cylinders) 40

Chapter 1

Introduction

In recent years, automation is one of the most effective tools of industry, thanks to the constant advancements in computation, electronics and control algorithms. It enables the completion of specific tasks quickly, efficiently and safely. The use of robots as a part of the automation process is a rapidly expanding field. The most prevalent type of robot, in industrial environments, are robotic arms. This type of robot is capable of performing a variety of operations, including transportation, assembly, packing and welding.

The complexity of the tasks assumed for these growing disciplines is also rising. Furthermore, robots are required to coexist and cooperate with humans in tasks like assisted industrial manipulation, collaborative assembly etc. Thus, the ability of the robot to comprehend the surrounding environment is a required feature to carry out the new tasks. Interacting with the environment and being aware of the changing conditions enables the robot to make autonomous decisions in more complex situations.

Obtaining knowledge about the surroundings can be accomplished by computer vision (CV). A computer vision system includes a camera that captures images, processes them and extracts information about the environment. The ability of vision provides the robot with the necessary tools to identify objects in its physical environment and select the best way to interact with them.

The development of computer vision has completely revolutionized the problem solving approach in many different industries like retail, manufacturing, warehousing and agriculture. Repetition in those types of industries is typically the key factor that makes automation easy to apply. However, when there is some variability in the task, an effort should be made to lessen the uncertainty, or a human should be recruited to make the judgments. Thanks to intelligent systems, decision making is now possible with the aid of perceptual systems, such as computer

vision. Visual servoing is the term used to describe this technique. It gathers data via visual sensors in order to control the robot.

Commercial systems provided by robot manufacturers include not only the physical robot but also the software and methodology for controlling it. Oftentimes, in the pursuit of friendly user interaction and intuitive development, these systems are mainly proprietary and focus on solving easy and generic tasks. For the implementation of more advance features developing third party programs and connecting them with the system is required. For this reason, a variety of software tools is available, regardless of the robot's manufacturer, that can integrate tasks like CV, robot control and advanced trajectory planning. Complex issues can be resolved by a later interface between the developed software and the robot.

1.1 Thesis Contribution

This thesis describes the implementation of an autonomous 6-DOF robotic manipulator with visual guidance, where the objective is for the robot arm to stack cylinder blocks, recreating randomly generated towers. The blocks have different, unique ArUco markers displayed on them so they can be identified. The robot is equipped with vacuum grippers to achieve the grasping of the objects. Also, a 2D camera is attached to its end effector, to enable the robot system to make decision and function autonomously by providing visual feedback.

In an area adjacent to the robot's workspace, cylinder blocks are being spawned, one at a time, in three different locations creating towers. A camera, placed above the aforementioned area, makes use of an object recognition algorithm to detect every newly spawned cylinder and extract information about its coordinates and identification. This information is transferred to the robot arm to proceed with its task.

The robot arm has a similar set of cylinders blocks, laid out in front of it. By exploiting the data communicated to it by the camera, it is able to determine which block it is required to stack and at what position. Since the robotic arm is equipped with a camera, it employs computer vision techniques to locate the desired cylinder. Consequently, it constructs a trajectory to position its end effector directly above the cylinder by making use of motion planning algorithms. After the pick and place task has been completed the process repeats for the next cylinder block.

The entire project has been implemented within the Robot Operating System (ROS). In the absence of a real robotic manipulator the whole environment was simulated in the Gazebo open-source 3D robotics simulator. However, the present approach could be applied to the actual robot arm, after minor adjustments, since all the packages used to describe and control the robot are provided and tested from the manufacturer.

1.2 Thesis Outline

In Chapter 2 we present all the background information needed for this thesis. We provide the characteristics and specifications of the selected robot arm and an overview of all the frameworks and software packages used for this thesis. Additionally, basic knowledge of concepts like computer vision and kinematics is discussed.

In Chapter 3 the basic problems of our approach are stated and also similar robotic projects are referenced.

In Chapter 4 we describe in detail the implementation steps of our approach. The exact environment setup is presented, as well as the methods and proposed algorithm to solve computer vision, robot control and path planning problems.

Chapter 5 contains the results of the approach in the simulated environment.

Finally, in Chapter 6 conclusion and future work to extend our approach is presented.

Chapter 2

Background

2.1 UR5 Robot Arm

The UR5 is a 6-axis robot arm developed by the Danish company Universal Robots. It is regarded as a collaborative robot, meaning that it is safe to operate alongside humans, as they are equipped with force sensors in their joints which will stop the motion as soon as they detect a collision with an object. An image of such a robotic arm can be seen at Figure 2.1 along with the dimensions of all the links.

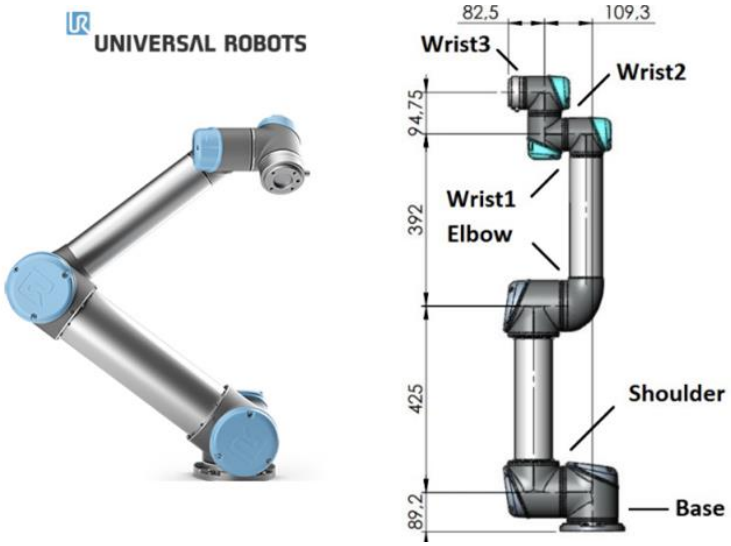

Figure 2.1 : UR5 collaborative robot arm

The UR5 a lightweight, adaptable robot that tackles medium-duty applications designed for industrial environments. It is one of the industry's most popular cobot because of its portability combined with the long-term flexibility of the UR5's higher payload and longer reach.

The specifications given by Universal Robots are provided in Figure 2.2.

UR5 Technical specifications Item no. 110105

6-axis robot arm with a working radius of 850 mm / 33.5 in

Weight:	18.4 kg / 40.6 lbs
Payload:	5 kg / 11 lbs
Reach:	850 mm / 33.5 in
Joint ranges:	+/- 360°
Speed:	All joints: 180°/s. Tool: Typical 1 m/s. / 39.4 in/s.
Repeatability:	+/- 0.1 mm / +/- 0.0039 in (4 mils)
Footprint:	Ø149 mm / 5.9 in
Degrees of freedom:	6 rotating joints
Control box size (WxHxD):	475 mm x 423 mm x 268 mm / 18.7 x 16.7 x 10.6 in
I/O ports:	Controlbox / Tool conn. Digital in 16 / 2 Digital out 16 / 2 Analog in 2 / 2 Analog out 2 / -
I/O power supply:	24 V 2A in control box and 12 V/24 V 600 mA in tool
Communication:	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP
Programming:	Polyscope graphical user interface on 12 inch touchscreen with mounting
Noise:	Comparatively noiseless
IP classification:	IP54
ISO Class Cleanroom robot arm:	5
ISO Class Cleanroom control box:	6
Power consumption:	Approx. 200 watts using a typical program
Collaboration operation:	15 Advanced Safety Functions Tested in accordance with: EN ISO 13849:2008 PL d EN ISO 10218-1:2011, Clause 5.4.3
Materials:	Aluminum, PP plastic
Temperature:	The robot can work in a temperature range of 0-50°C
Power supply:	100-240 VAC, 50-60 Hz
Cabling:	Cable between robot and control box (6 m / 236 in) Cable between touchscreen and control box (4.5 m / 177 in)

Universal Robots A/S
Energivej 25
DK-5260 Odense S
Denmark
+45 89 93 89 89

www.universal-robots.com
sales@universal-robots.com

Figure 2.2 : UR5 Technical Specifications

There are six revolving joints in the robotic arm. These joints are referred to as Base, Shoulder, Elbow, Wrist1, Wrist2, and Wrist3. For these kinds of robotic arms, the UR5 features a layout that is very typical. The Shoulder and Elbow joint are rotating perpendicular to the Base joint. The main purpose of the wrist joints is to move the Tool Center Point (TCP) into the proper orientation.

The Degree of Freedom (DOF) for a robot arm is the configuration space dimension, which in turn is the minimum number needed to describe this space. The degrees of freedom for a mechanism are calculated using Grubler's formula [22] by deducting the freedom of a joint from the number of independent constraints.

$$\text{DOF} = \sum (\text{ Freedoms of bodies }) \text{ - Number of independent constraints}$$

For a robot arm with J number of joints this formula become :

$$\text{DOF} = m(N-1) - \sum_{i=1}^{j} c_i$$

Where m is the number of freedoms for a single rigid body. For planer bodies, m equals three with one rotational and two transitional in 2D space. In our case, for spatial bodies in a 3D environment, m equals six with three rotational and three transitional. N is the total number of bodies including the ground. Six revolute joints make up UR5, and when ground is added, N becomes seven. Finally, c is the constraint between two rigid bodies. For revolute joints, c is equal to 5. The aforementioned equation can be used to obtain the UR5 robot arm's DOF by entering these numbers.

$$\text{DOF} = 6(7-1) - 6(5) = 6$$

2.2 Robot Operating System and Gazebo

Robot Operating System ROS [17] is a flexible collaborative framework for robotic software development, and consists of a large collection of tools, libraries and templates. Research organizations, labs, and individuals can use ROS as the programming platform to contribute their algorithms, also known as ROS packages, and build software by utilizing pre-existing modules.

Numerous ROS and ROS-Industrial [18] packages are being used for the purposes of this project. The latter are designed specifically for industrial robotic applications with the support of the robotics industry and research institutions. ROS-Industrial provides numerous sensor plugins, robot controller packages and planning algorithms in order to enhance agile industrial robotics for a wide range of automation tasks and industrial manufacturing.

Additionally, ROS framework comes with a wide variety of tools to help with real time environment configuration and monitoring, data visualization tools, like Rviz [23] and the transformation system tf [24].

In the ROS environment each process performing computation is called a ROS node and represents a device. ROS middleware integrates a communication infrastructure in order for the nodes to interact with each other, via a peer-to peer network. The capabilities, provided for inter -process communication, are described as follows:

- Message passing via publishing and subscribing systems. ROS nodes exchange messages in an asynchronous way through ROS Topics. A ROS node that generates and publishes data to a particular ROS Topic is referred to as a publisher. Any ROS node can subscribe to a ROS subject and receive information if it is interested in the published data. A publishing ROS node is not aware of the identities of the subscribers to its subjects in this manner of communication. Through a single topic, numerous publications and subscribers can interact.

- Remote procedure calls (RPC) in request-response systems. RPC that provide request and reply interactions are called ROS services. The RPC interactions in ROS are defined by a pair of ROS messages. A provider ROS node registers the service under a namespace (a directory of names), and a client calls the service by sending a request message and waiting for a reply.

- Distributed parameter server. A shared dictionary to store static and non-binary parameters. The parameters stored in the server can be retrieved globally by all ROS nodes and they provide and registration services to the rest of the nodes.

ROS messages are defined using a simplified message description language, which enables ROS to automatically create source code for several languages. A ROS message is a collection of constant definitions and descriptions for data fields. The built-in or self-defined description is the field type listed in the left column. The field type is followed by the field name, which provides the name of the data structure and is bounded by a space. Although it is not necessary, a data field's description can be added after the comment symbol (#), as seen in the figure below.

```
std_msgs/Header header    # Message Header
string name               # Object name in a database
std_msgs/UInt16 obj_seq   # Assigned object sequential number
time detection_time       # The time when the object was detected
geometry_msgs/Pose pose   # Pose of the object observed at the detection time
```

Figure 2.3 : A ROS message example

The workspace environment was simulated and tested using Gazebo, a robotics simulator. Gazebo uses Open Dynamic Engine (ODE) for dynamics simulation. Also, high-quality rendering is supported by the Object-Oriented Graphics Rendering Engine (OGRE). A variety of sensors and robotic models are included in the Gazebo library. Various command line tools are available to users, and customized plugins for robot, sensor and environment control can be programmed in Python or C++ and integrated into Gazebo.

2.3 MoveIt!

MoveIt! [19] Motion Planner is a set of software packages integrated with the ROS and designed specifically to provide motion planning capabilities, especially for the manipulators. It is state–of–the–art software for mobile manipulation and it provides the latest advances in motion planning, manipulation, 3D perception, robot kinematics and control. MoveIt! packages support the UR5 robot and were used as a convenient platform to implement the motion planning pipeline.

The system architecture for the primary node, move_group, is shown in Figure 2.4. It combines controllers, sensors, libraries and all the other components to provide a set of ROS

actions (red lines), services (blue lines) and topics (green lines). This node obtains three different types of information from the ROS parameter server:

- URDF which is the robot_description file on the ROS parameter server.
- SRDF which is the robot_description_semantic parameter on the ROS parameter server. The MoveIt! Setup Assistant is commonly used by a developer to generate the SRDF just once.
- MoveIt! Configuration which includes MoveIt specific settings, such as kinematics, joint limits, motion planning, perception, and other data.

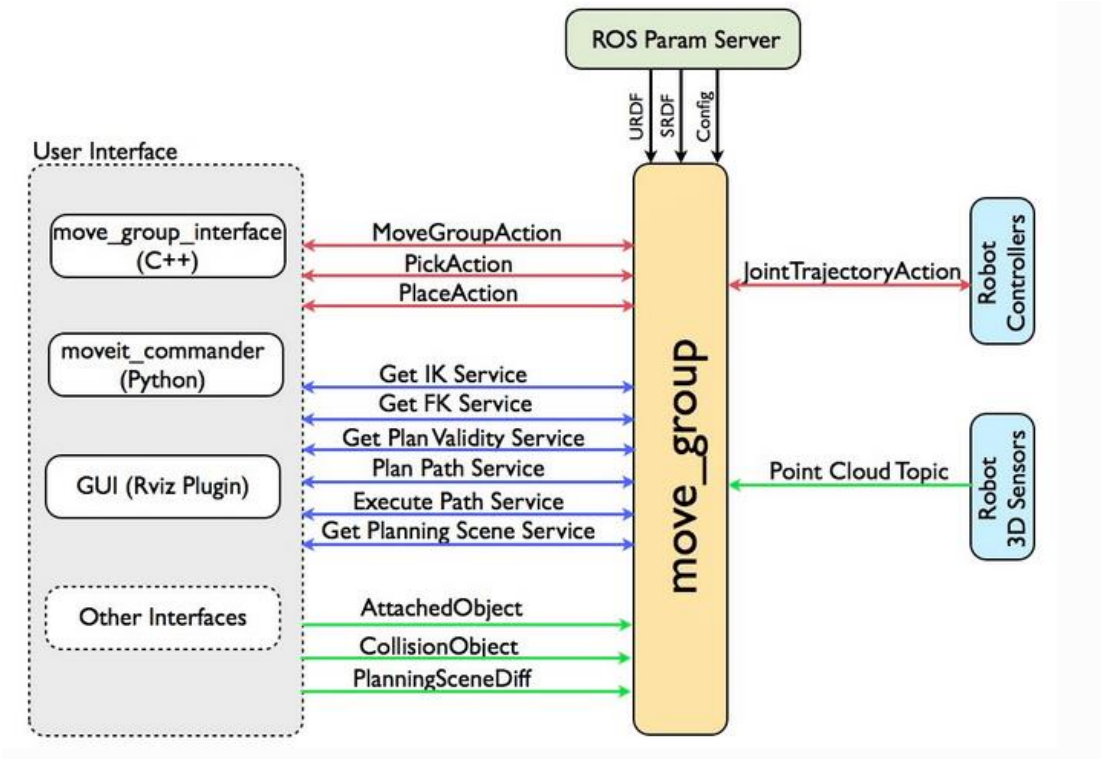

Figure 2.4 : Move_group Architecture

Topics and actions are used by the move group node to communicate with the robot. Joint state information is published on the /joint state ROS topic and transform tree data is published on the /tf topic. Information on the robot transform tree is published by the robot state publisher. The state of the robot and its surrounding environment is defined as a planning scene in a scene monitor. In order to compute the trajectory, the motion planning algorithm retrieves the data from the planning scene.

MoveIt! Includes motion planning plugins that allow the user to take advantage of multiple open-source planning libraries such as the Open Motion Planning Library (OMPL) [20]. OMPL is used as the primary or default set of planners in MoveIt. The planners in OMPL are abstract, thus they no concept of a robot. Instead, MoveIt! sets up OMPL and supplies the back-end needed for OMPL to work with robotics-related issues. The structure of OMPL can be seen in Figure 2.5.

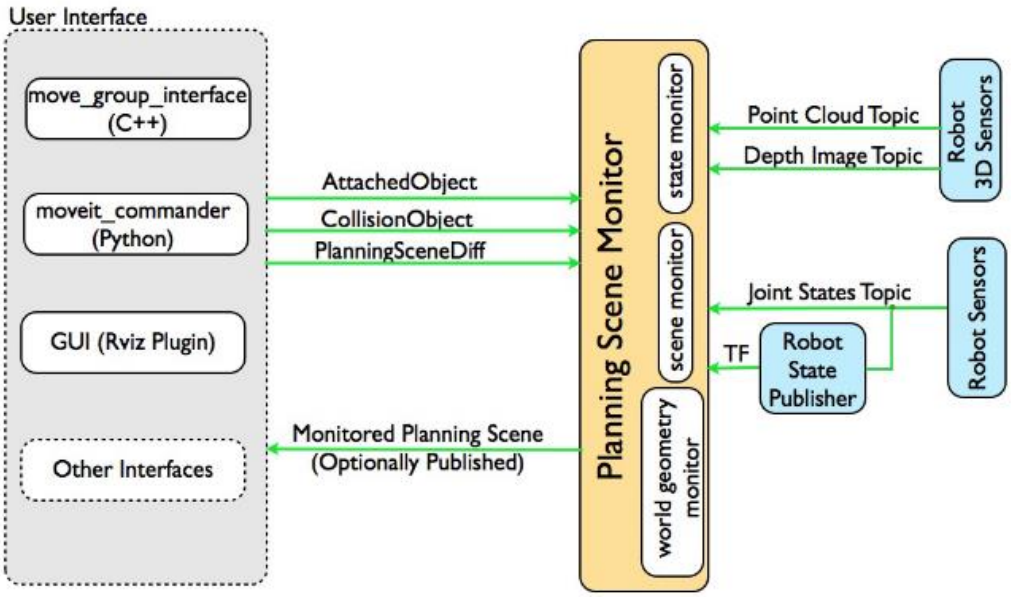

Figure 2.5 : Planning scene pipeline

2.4 Kinematics

The branch of mechanics that studies the motion of a body or a system of bodies, without consideration given to its mass or the forces acting on it, is known as kinematics. Hence, the study of the kinematics of robotic manipulators refers to all the geometrical and time-based properties of the motion. From the mechanical structure point of view, the UR5 robot is an open kinematic chain connected by 6 revolute joints, so the 6-DOF robotic manipulator's kinematic model can be developed.

2.4.1 Forward Kinematics

Forward kinematics is the method for determining the orientation and position of the end effector, given the joint angles and link lengths of the robot arm.

The transformation matrix H_j^i is the matrix mapping point p in reference frame Ψi into Ψj. For a robotic arm it is convenient to begin with the base frame Ψ0 and start from the zero vector in xyz.

$$\begin{bmatrix} p^j \\ 1 \end{bmatrix} = H_j^0 \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$$

The transformation matrices can be multiplied to conveniently map into other reference frames.

$$H_k^0 = H_j^0 H_k^j$$

The position of the joints of the UR5 are used to construct every homogeneous transformation matrix and represent the transformation from joint i−1 to joint i. Where θ_1 is the Base joint and θ_6 is the Wrist3. Eventually a mapping from the base frame to the position of the TCP can be constructed.

$$H_6^0 = \prod_{i=1}^{6} H_i^{i-1}(\theta_i)$$

The transformation matrices can be easily constructed in accordance with the Denavit-Hartenberg convention [21], which is a commonly used to define four parameters that describe how the reference frame of each link is attached to the robot manipulator.

$$H_i^{i-1}(\theta_i, d_i, a_i, \alpha_i) = R_z(\theta_i)T_z(d_i)T_x(a_i)R_x(\alpha_i)$$

These Denavit-Hartenberg parameters are known for the UR5 and listed in Figure 2.6.

	$\boldsymbol{\theta}\ [rad]$	$\boldsymbol{a}\ [m]$	$\boldsymbol{d}\ [m]$	$\boldsymbol{\alpha}\ [rad]$
Joint 1:	0	0	0.08920	$\frac{\pi}{2}$
Joint 2:	0	−0.42500	0	0
Joint 3:	0	−0.39243	0	0
Joint 4:	0	0	0.10900	$\frac{\pi}{2}$
Joint 5:	0	0	0.09300	$\frac{-\pi}{2}$
Joint 6:	0	0	0.08200	0

Figure 2.6 : UR5 DH parameters

2.4.2 Singularities

In robotics, singularity is a common problem. When a robot is in a singular configuration it cannot track the desired trajectory. How a singularity occurs and how it affects the manipulator will be discussed.

The transformation matrix from the joint velocity space to the end effector velocity space is called the Jacobian.

Geometric Jacobian $$v = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = \begin{bmatrix} J_P \\ J_\omega \end{bmatrix} \dot{\theta} = J_G \dot{\theta}$$

Analytic Jacobian $$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} J_P \\ J_\phi \end{bmatrix} \dot{\theta} = J_A \dot{\theta}$$

There is a difference between the geometric Jacobian and the analytic Jacobian. While ω represents the angular velocity $\dot{\varphi}$ represents the change of orientation of the end-effector frame φ. ω and $\dot{\varphi}$ are in general not the same.

The geometric Jacobian for a robot with only revolute joints can be constructed as follows [25]:

$$J_G = \begin{bmatrix} Z_0 \times (o_N - o_0) & \cdots & Z_{i-1} \times (o_N - o_{i-1}) & \cdots & Z_{N-1} \times (o_N - o_{N-1}) \\ Z_0 & \cdots & Z_{i-1} & \cdots & Z_{N-1} \end{bmatrix}$$

Where Z_i are the first three elements of the third column of transformation matrix H_i^0 and o_i are the first three elements of the fourth column of H_i^0

Singularity happens when a rank is dropped in the Jacobian. There are two types of singular configurations for robotic arms, boundary singularities and internal singularities. If the robot is requested to move outside its workspace, a boundary singularity will occur. Typically, this takes place when the arm is fully extended. Internal singularities are usually caused by the alignment of two or more of the robot's axes. As a result, the action of one joint can be cancelled by another joint. In this situation, there are infinite possibilities for the movement, leaving the action undetermined.

Near-singularities pose also a problem. In the case, when the determinant of the Jacobian becomes small, the inverted Jacobian will become large. This means that some joints must perform a considerable movement for a small end-effector movement. That can lead to the joint's velocity operating outside its physical range, making it impossible for the end-effector to move according to the intended trajectory.

2.4.3 Inverse Kinematics

In section 2.4.1 the mapping from the joint space to the cartesian space for the robotic manipulator was discussed. This is a fairly easy problem since the transformation matrix has only one solution and only depends on the known joint angles.

Finding the joint variables for given end–effector positions is a much more complicated issue known as inverse kinematics. The joint angles, which are unknown in this instance, are also a factor in the transformation matrix. Additionally, given a single Cartesian coordinate, there are often eight distinct configurations that are feasible. The two main types of inverse kinematics techniques are iterative techniques and closed form solutions.

The robotic arm's geometrical properties are utilized by the closed-form solution, which can directly solve the inverse kinematics problem. To illustrate this concept, a two DOF robotic arm example [26] is explained below.

A 2-link arm is shown in figure 2.7. Link 1 and 2 are respectively l_1 and l_2. The angle from link 1 to the x-axis is expressed as θ_1. The angle between link 2 and link 1 is expressed as θ_2. The position of the end effector in the Cartesian space is expressed in p.

The Forward Kinematics for the position of the end effector of this robotic arm are straightforward.

$$p_x = cos(\theta_1)l_1 + cos(\theta_1 - \theta_2)l_2$$
$$p_y = sin(\theta_1)l_1 + sin(\theta_1 - \theta_2)l_2$$

Where p_x is the x-coordinate and p_y is the y-coordinate of the end effector.

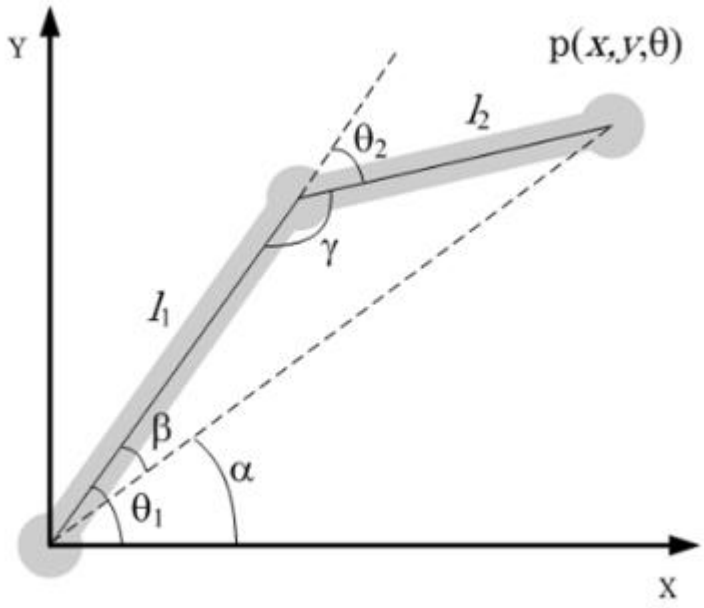

Figure 2.7 : Inverse Kinematics example of a 2-link robotic arm

The Inverse Kinematics is more complicated. The challenge is to find the angles θ_1 and θ_2 , knowing the desired position of the end effector, p.

The Rule of Cosine can be used to determine the angle of θ_2, where $\cos(\gamma) = -\cos(\theta_2)$. There will be two solutions for θ_2 .because of the arccos, meaning $-\theta_2$ will also be a solution.

$$(p_x^2 + p_y^2) = l_1^2 + l_2^2 + 2l_1 l_2 cos(\theta_2)$$

$$\theta_2 = arccos\left(\frac{(p_1^2 + p_2^2) - l_1^2 - l_2^2}{2l_1 l_2}\right)$$

The Rule of Sine can be used to determine the angle of θ_1. Angle θ_1 is dependent on θ_2, but there are no different solutions for a same θ_2.

As a result, there will typically be two alternative configurations for a given point in the Cartesian plane.

$$\frac{sin(\beta)}{l_2} = \frac{sin(\theta_2)}{\sqrt{p_x^2 + p_y^2}}$$

$$\alpha = arctan\left(\frac{p_y}{p_x}\right)$$

$$\theta_1 = \alpha + \beta = arctan\left(\frac{p_y}{p_x}\right) + arcsin\left(\frac{l_2 sin(\theta_2)}{\sqrt{p_x^2 + p_y^2}}\right)$$

This was a simplified example with only two links. Finding a closed form solution for a 6 DOF robotic arm is significantly more complicated.

Using iterative techniques is another method to perform the inverse kinematics. Iterative numerical approaches are one of those techniques. A search algorithm like the Newton-Raphson can be used. The transformation matrix H_6^0 can be used to create the cost function for this optimization algorithm. The cost functions derived from the DH-parameters of the UR5, that can be used for various minimization algorithms, are described in Appendix A.

2.4.4 Inverse Jacobian Technique

The method to calculate the inverse Kinematics of the UR5 is the inverse Jacobian technique. Although it is impossible to reverse the joint space to Cartesian space mapping for position, it is possible to do so for rates. As we mentioned in section 2.4.2, this mapping is known as the Jacobian and it is only dependent on the angles of the joints, thus in can be inverted.

The plan is to differentiate the 6D position and rotation vector. The resulting vector, which contains the TCP's velocities, is multiplied by the Jacobian's inverse. The present angles of the joints are used to calculate this Jacobian as described on section 2.4.2. The resulting vector represents the joints' velocities and it is finally summed to produce the vector with the joint angles. The whole process is briefly presented below. Here p_i is the position and rotation vector for instant i. JG is the geometric Jacobian and θ_i is the vector for the angles of all six joints for instance i.

$$p_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ R_{xi} \\ R_{yi} \\ R_{zi} \end{bmatrix}$$

$$\Delta p_i = p_i - p_{i-1}$$

$$\Delta \theta_i = J_G^{-1}(\theta_{i-1}) \Delta p_i$$

$$\theta_i = \theta_{i-1} + \Delta \theta_i$$

It should be noted that the order in which the rotation matrices are multiplied affects the angle. There is no way to compute the difference without introducing an error. Furthermore, the present position affects the Jacobian used to determine the following position. This will, also, produce an error since θ is generally not constant. In order to avoid problems, the steps must be very small so the errors are not significant.

2.5 Robot Vision and ArUco Markers

Computer vision is a field that enables computers and systems to perceive the world around them and derive meaningful information by analysing digital images, videos and other visual inputs. Robot vision is a related branch where the host computer not only processes the environment data collected from cameras or sensors but also uses the information to control the client robot.

The most used software to handle computer vision problems is the OpenCV [27], an open source cross-platform programming language. This library implements many functions useful to handle cameras, intrinsic calibrate them and to manage and parse images.

A very useful tool for computer vision systems are fiducial markers. These markers are image like objects, which are designed to be detectable and typically contain an interpretable meaning. They come in a variety of sizes and shapes, ranging from tiny dots to intricate bar-code images.

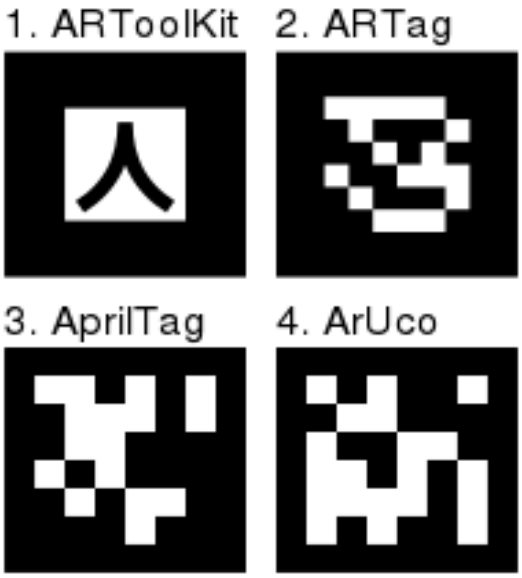

Figure 2.8 : Comparison of some common Fiducial markers

For the purposes of this thesis the ArUco markers [11] are best suited. These markers are comprised by an external black border and an inner region that encodes a binary pattern. There are several dictionaries for ArUco markers which differ in the number of markers they contain

and the number of inner squares that encode the binary pattern. In this application, the marker is a 7x7 grid with the outer rows and columns used as a black border, leaving a 5x5 grid for the encoding. Only two squares on each row are utilized for the actual code, with the remaining used for error checking.

This evaluates to $1111111111_2 = 1023_{10}$, resulting to 1024 unique IDs.

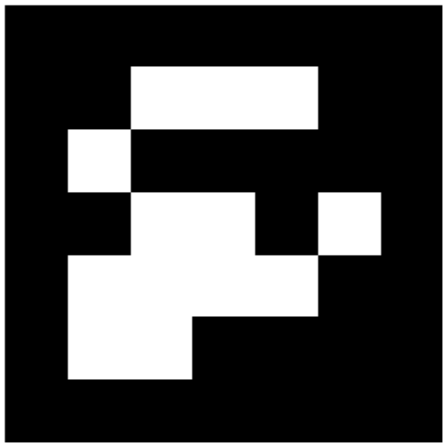

Figure 2.9 : ArUco Marker with ID = 4

The large squares in this design makes it easy and fast to detect, compared to something like a QR-code, while it is still able to contain an encoded ID. Therefore, we can preserve all the needed information while any unnecessary complexity is eliminated. Another advantage of these markers is that each of them provides a 4-point vector representing the pixel coordinates of the corners. When the corners are known, the position of the center can be calculated and because of the asymmetry of the binary code, the orientation of the marker can be determined.

Chapter 3

Problem Statement

3.1 Autonomous Vision Based Control For Robot Manipulator

The use of robot manipulators is a continuously developing area, especially in industrial and manufacturing environments. As a result, there is a growing need for agile and adaptable autonomous robotic systems. In order for the robots to be considered autonomous, they must be able to perceive the surrounding environment and make decisions about the order of actions.

This thesis is focused on developing autonomous control of the UR5 robot arm, by the means of computer vision, recognition and tracking, in order to complete picking, handling and stacking tasks. The robot must be able to operate autonomously, without user input. Furthermore, it should be able to interact with its environment and recognize objects of interest. This is crucial in order for the robot arm to adapt to potential changes of its surroundings, and operate under random conditions of the workspace.

The introduction of extra features like, robot vision and grasping, calls for the integration of the appropriate modules. Additionally, the tools for the trajectory planning, robot control and object recognition must be implemented. Due to the need of performing the tasks accurately and reliably, synchronization and efficient algorithm development is a key factor for the success of the system.

3.2 Related Work

Autonomous robotic manipulators have been an area of increasing research interest, due to their extensive use in industrial and manufacturing applications. Robot arms with 6DOF are complex enough to take the place of humans and perform complicated tasks. Many approaches focus on the integration of robotic arms in industrial environments [1], [2], [5]. These approaches utilize vision systems to enable the robot to perceive its surroundings. The Amazon Picking Challenge (APC) [3] is a well-known competition, organized every year by the amazon company, in order to promote shared and open solutions to some of the big problems in unstructured automation. This challenge focuses on pick-and-stow operations where a robot recognizes target items, picks items from shelves and places them in shipping boxes.

Moreover, visual guidance systems are examined in [4], [7], [6]. In those approaches different camera systems have been used, from simple 2D USB camera to the more advanced Kinect depth camera. Also, the recognition tasks vary, as some approaches focus on color recognition, while others deal with shapes and object detection. In our approach, we used ArUco markers [11] for the object detection. These markers are commonly used in robotics, not only for object tracking but also for localization and navigation [8], [9]. Since there are ways to extract information about the object's distance from the camera, if you possess knowledge about the markers' dimensions, the use of a depth camera is not necessary.

Robot arm kinematics and motion planning are also topics with great interest in robotics. Specifically, for the UR5 manipulator the kinematic and dynamic modeling is explained in [10],[12]. Different approaches for the motion control of the robot have been investigated in the past, including the Proportional Integral Derivative method and the Fuzzy Logic Control [13],[14]. These techniques are especially useful for controlling the hardware of the real UR5 robot. Finally, the motion planning strategy for the 6-DoF robot has been studied in several publications [15],[16] which contain the conditions and parameters involved in robot trajectory planning.

Chapter 4

Our Approach

4.1 Simulation Environment

The first step was to create the environment for the simulation in the Gazebo platform. The environment consists of two workbenches. The first one, that will be referred to as workbench 1, is the place where the towers, which the robotic arm is called to recreate, are being spawned. A stationary 2D camera is placed directly above, so that its field of view covers the entirety of the area. The second workbench (workbench 2) includes the UR5 manipulator and 8 cylinders with unique ArUco markers displayed on top of them. This is the operation area of the robot arm. The goal is for the robotic manipulator to handle the cylinder blocks on its workbench to recreate the spawning of stacks that takes place in workbench 1.

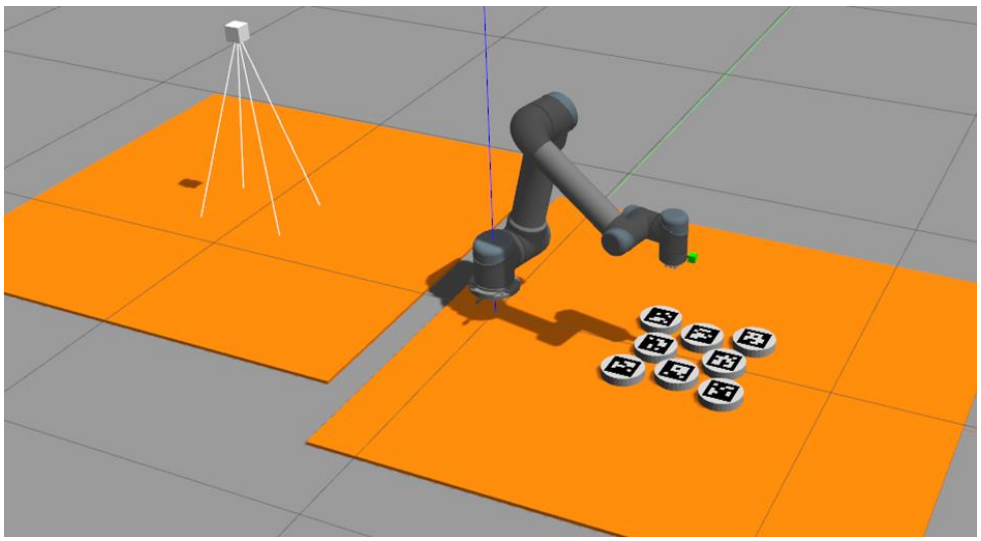

Figure 4.1 : Gazebo environment. Workbench 1 (left), Workbench 2 (right)

The process by which the stacks, that we are called to copy, are being spawned is described as follows. In each iteration a single cylinder, from a set identical to the one described above, is being spawned randomly in one of three positions, on workbench 1. The order by which the IDs spawn is random. The stationary camera is responsible for detecting the cylinder and identify its ArUco ID and position. The robotic manipulator is able to acquire this data via the ROS Service /latest_aruco_id, in order to continue with the pick and place task.

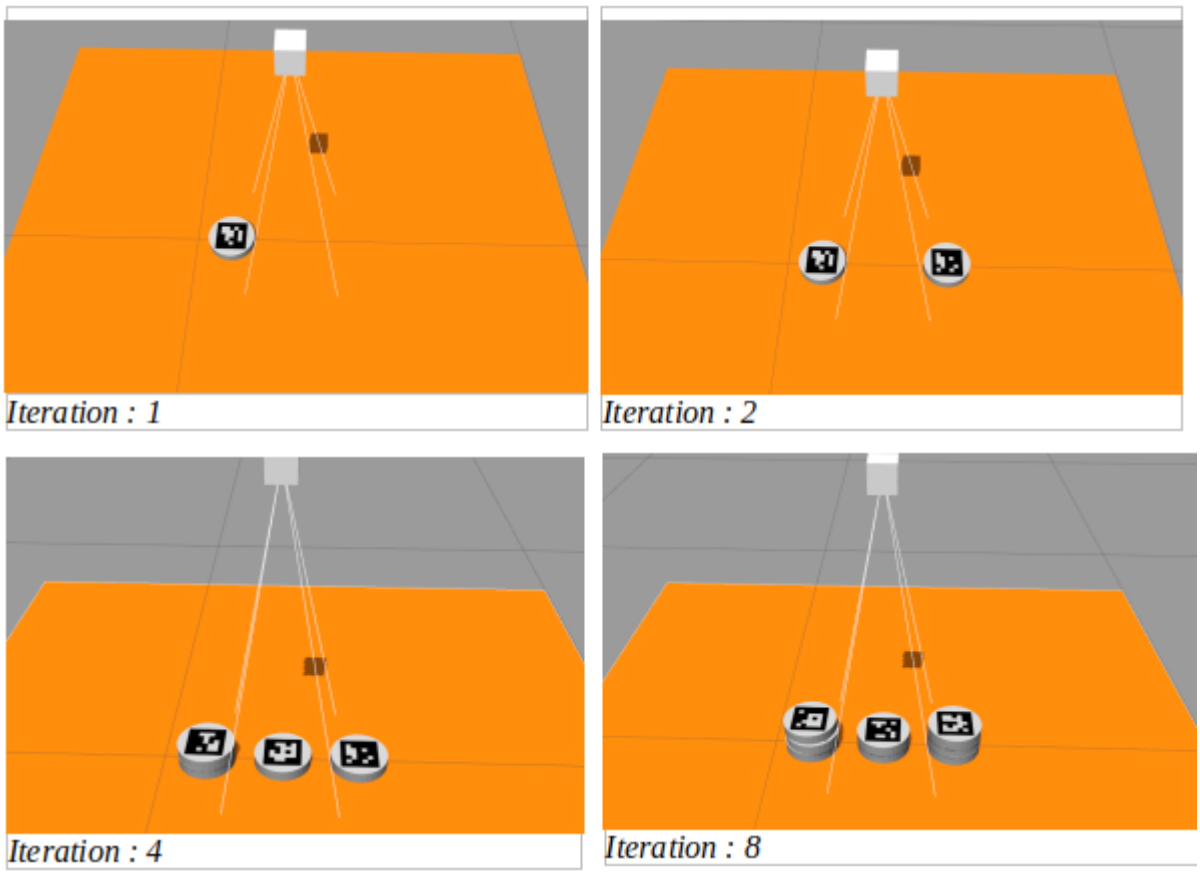

Figure 4.2 : Workspace 1, target stacks spawn

4.2 UR5 Robot Modeling

For the UR5 robotic arm ROS Industrial universal_robot package was used. Universal robot descriptions, drivers, gazebo resources, kinematic, ROS messages and MoveIt! packages are included. Robots in ROS are modelled in The Universal Robotic Description Format (URDF) which is XML file format. For speeding up of tedious writing in URDF language serves XML

macro (Xacro). Xacro enables the developer to define repeating properties and create own macros. Several addition had to be introduced in the original ur5.urdf.xacro file in order to add the desired functionalities to the robot.

First, vacuum grippers were attached to the manipulator's end effector. The grippers are available as a Gazebo ROS dynamic plugin. They operate by collecting data from a ROS topic and applying wrench to a body accordingly. Since the vacuum gripper only provides limited force, nine identical grippers were added in order to lift the object. An extra link and a joint are added to the URDF file for every gripper.

The robot was also equipped with a camera module attached to its end effector, available as a gazebo sensor plugin, for the detection and tracking of the cylinders. The camera is displayed in the simulation as a green cube. All camera's image raw data are published at the /ur5/usbcam/image_raw topic.

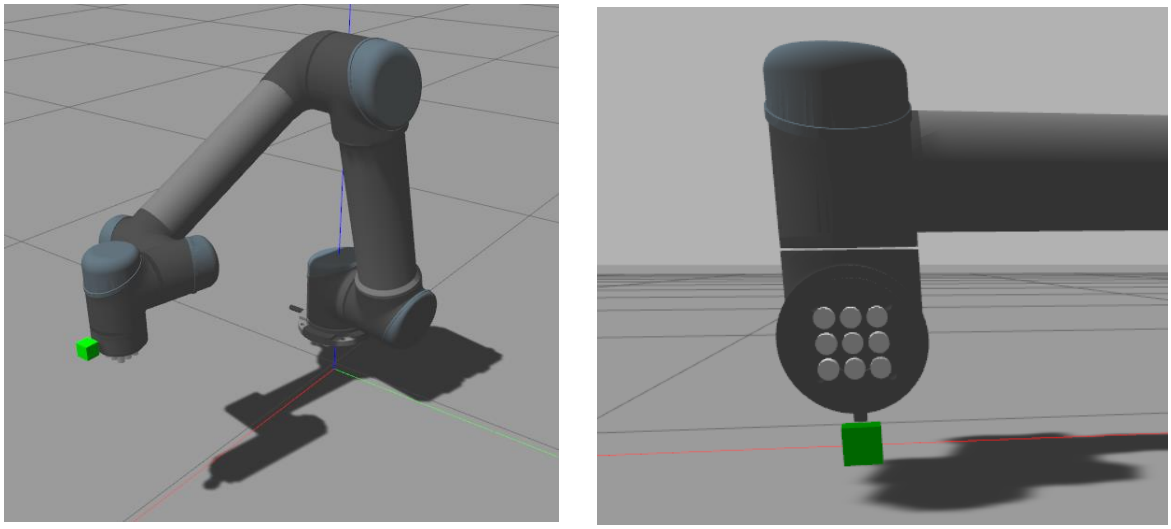

Figure 4.3 : UR5 Robot model equipped with the additional modules and the vacuum grippers configuration

In the discipline of robotics, each component of a robot is described by its coordinate system and its origin, which are provided by a position and a rotation vector relative to another reference system. Additionally, for moving robots the relation between two subsystem parts is dependent to time. ROS has an integrated transform library, called tf [24], which is utilized as the main method of tracking positional data. This information is very useful to determine robot poses and objects' position at a given timestamp. The tf tree of the complete UR5 with the additions of the extra modules and views in Rviz plugin are presented in Figure 4.4 and Figure 4.5 respectively.

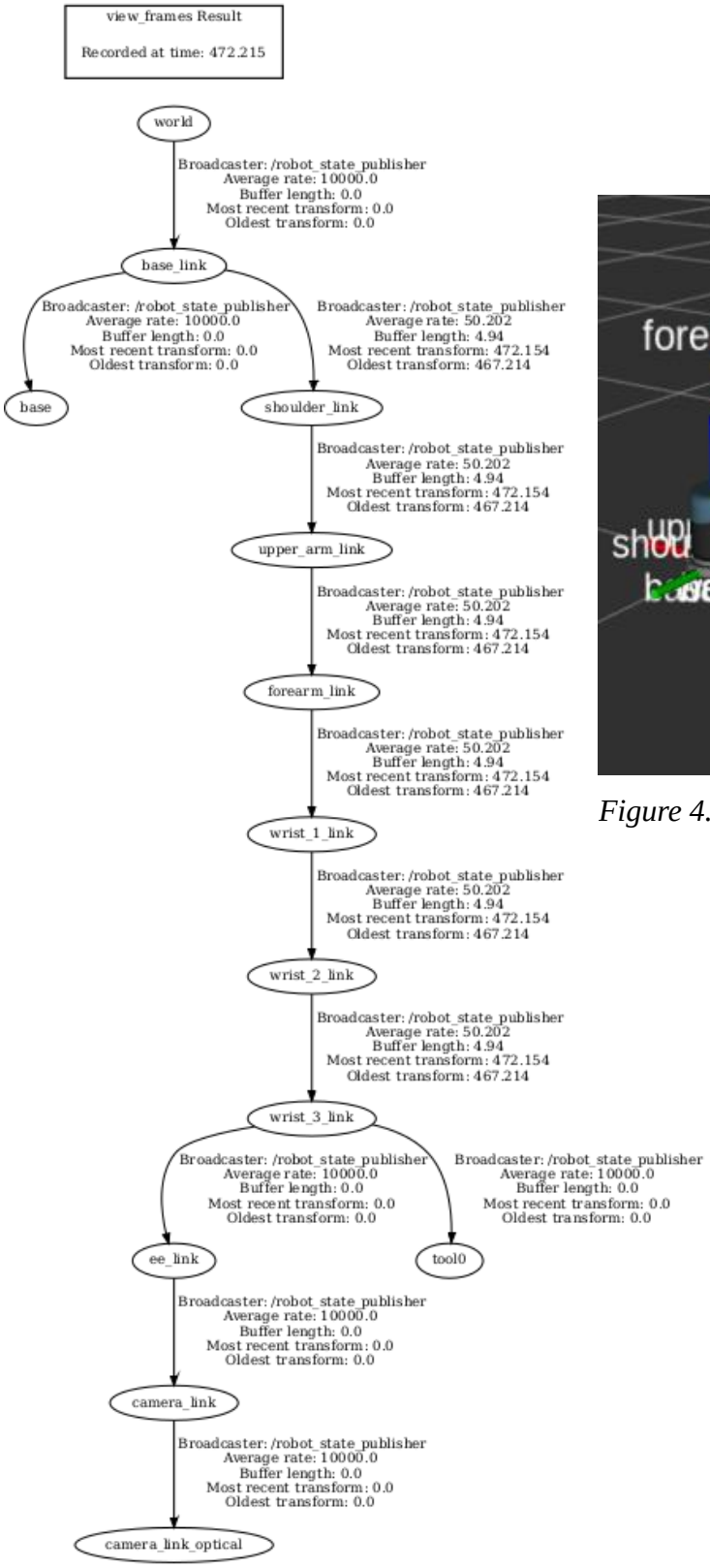

Figure 4.4 : UR5 tf tree

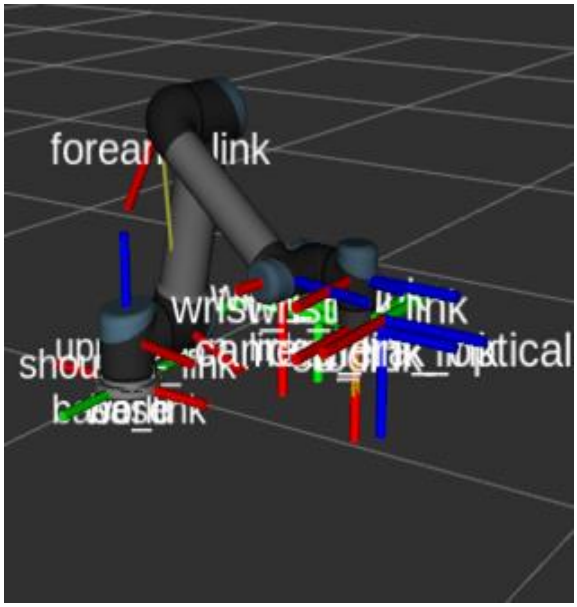

Figure 4.5 : Tf tree view in RVIZ plugin

4.3 System Synchronization and management

Synchronizing every subtask is crucial to ensure the smooth operation of the system. Delays in the simulation can cause loss of data, during the communication of different nodes, resulting in errors. To solve this problem a simple finite state machine (FSM) was implemented which controls the robot's behavior and the function of all the other modules. The complete finite state machine is shown in figure 4.6.

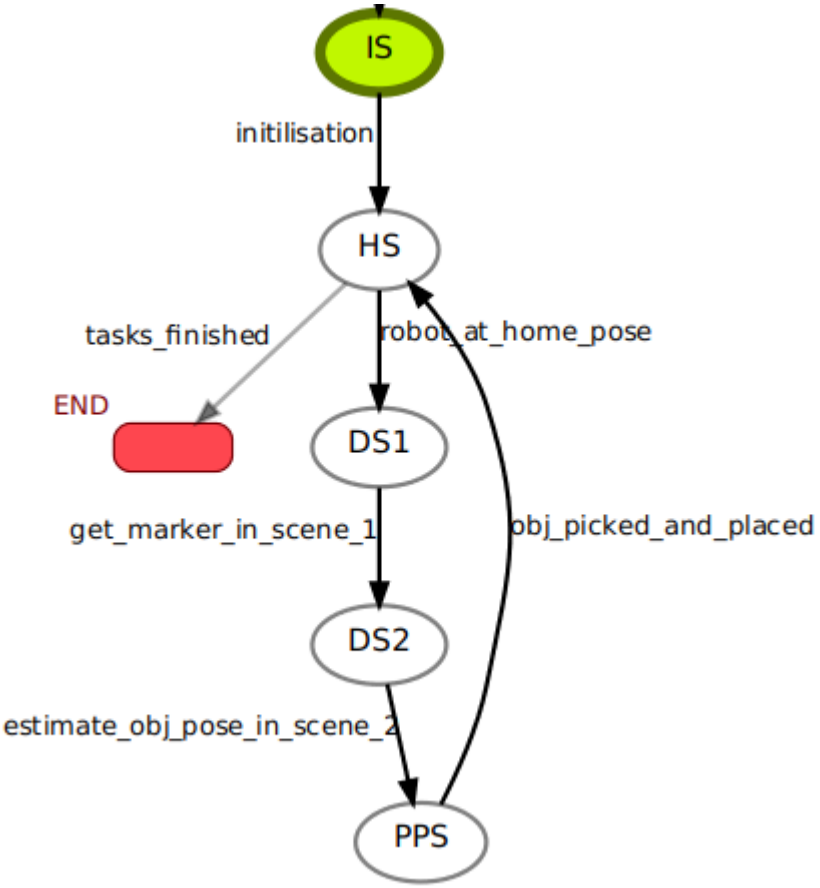

Figure 4.6 : FSM

The starting point is the Initialization State, IS. The FSM will remain in this state until the simulation environment is built. When the robot, the workbenches, the scene 1 camera and the cylinders are spawned, we transition to the next state.

HS stands for Home State, during this state the arm goes to its home pose. In this pose, the robot's camera is aimed at the cylinder blocks that must be stacked. During this state, a cylinder, whose placement the robot must recreate, spawns on workbench 1.

The next state is Detection Scene 1, DS1. The 2D camera located above workbench 1 recognizes the newly spawned ArUco ID and based on its coordinates it determines in which of the three towers it was stacked. This information is communicated to the robot via ROS Services request and reply messages. The camera node stores information about all the ArUco IDs it has previously encountered, that way we ensure that only the information about the most recently spawned cylinder will get transferred to the robot.

The next transition in to the Detection Scene 2 state, DS2. Now the robot arm in called to detect the cylinder with the same ArUco ID, from the block set located in front of it, using the attached camera on its end effector. Furthermore, the cylinders position must be estimated for the next step.

It is finally time for the pick and place task, PPS. The robot arm has knowledge about the coordinates of the desired cylinder and the stack on which it should be placed, thus the motion planning can be implemented. When the placement is completed, the FSM transitions to the Home State for the loop to repeat. When we run out of cylinder blocks the program will progress to the End State.

4.4 ArUco Marker Detection

The detection algorithms were implemented using OpenCV, since it provides a large set of image processing algorithms. Additionally, a library specifically for ArUco markers detection, developed by Rafael Muñoz and Sergio Garrido [11] is available for OpenCV.

The script from Github repository [28] was used, under MIT License, to create the ArUco markers. Then, using Blender open source 3D creation suite, COLLADA files were created describing the mesh, material and texture of each cylinder based on the ArUco image files. An example of a generated ArUco marker and the corresponding blender model is shown in Figure 4.7.

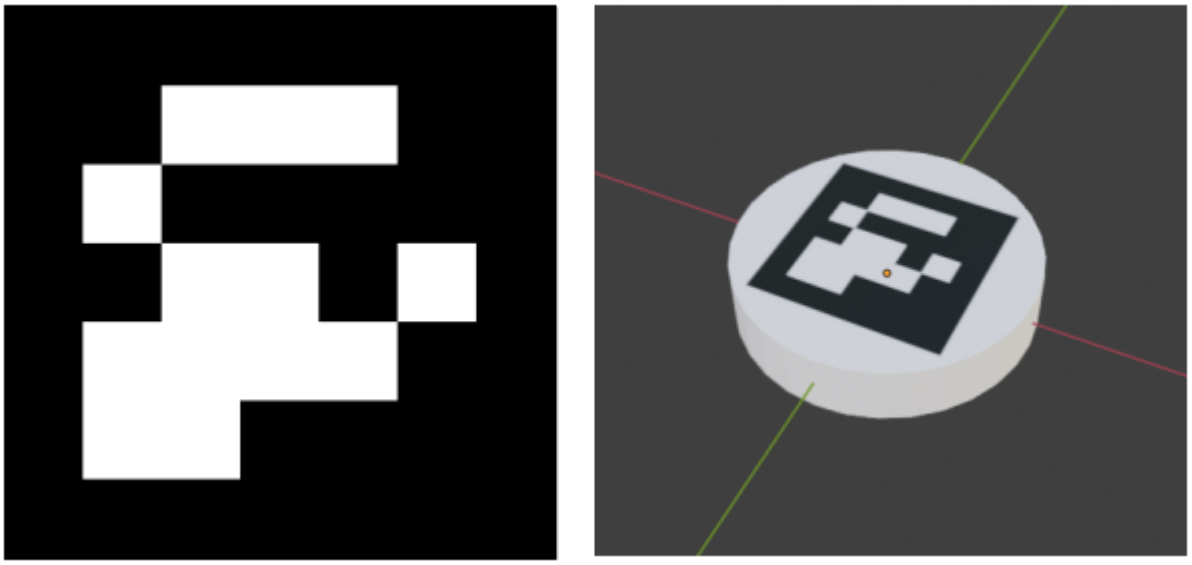

Figure 4.7 : ArUco Marker ID=4 and the corresponding cylinder blender model

Before we proceed with the detection, it is necessary to ensure that we are accepting the camera's latest frame. For that reason, we discard the first 20 frames the camera provided in each detection state, which is equivalent to 2 seconds since the camera operates at 10 frames per second. This is enough time to make sure that delays in the simulation or in the publishing of the camera's recorded image on the ROS Topic, will not cause any issues.

The first step of the detection process is finding marker candidates. In order to do that, an adaptive threshold [29] is applied to the image. Window sliding is used in adaptive thresholding to obtain the best greyscale value for each window. Values below the calculated value will be black and above is white. Contours can now be extracted from the binary image [30]. If they are not convex or close to a square shape, they will be discarded. The size of the edges or the separations between them, among other filters, are used to define these conditions [31].

Now that the square shaped candidates have been selected, the next step in to determine if they are in fact ArUco markers with a valid ID. The candidates will be subjected to perspective transform. This form is known as the canonical form. Otsu thresholding [32] will then be applied. The goal of Otsu thresholding is finding an optimum point of the histogram of the image so that the disparity between the black and white (foreground and background) distributions is minimum.

For the purpose of this thesis the 5x5 ArUco dictionary was used, so the last version of the images will be divided into 5x5 sub-images (with the border, 7x7). Since they are already thresholded, it is easy to convert the binary images to a binary matrix. These matrices will be searched in the dictionary and if they match with the markers their IDs will be determined.

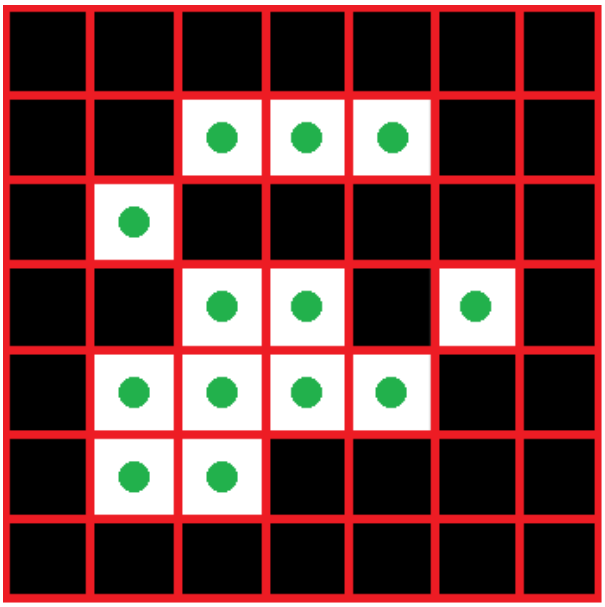

Figure 4.8 : Marker cells

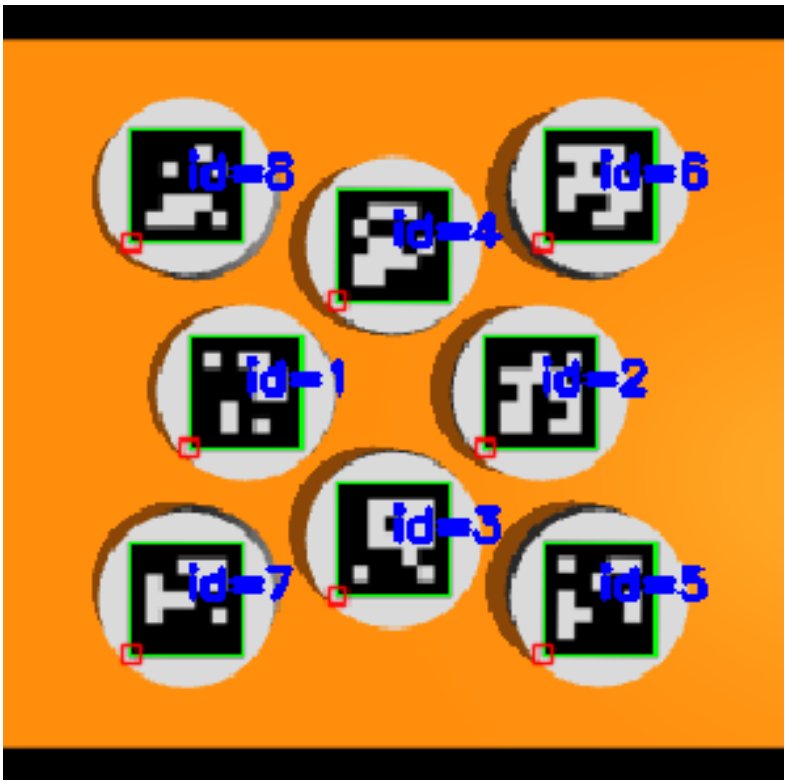

Figure 4.9 : ArUco Markers' IDs detection by the UR5's end effector camera

The final step is to estimate the positions of the markers. ArUco detect function returns the coordinates of the corners for each marker, in pixel units, in the camera field of view. The depth can be calculated if we take into account the exact dimensions of the marker. In our case, the square's side length is 7.81 centimeters. If we pass this data, along with the camera's distance coefficient and position, as parameters in the function estimatePoseSingleMarkers(), which is included in the ArUco library, it will return 2 vectors, the translation (position) and rotation of the marker. The rotation vector is not relevant for its application since cylinder blocks are used, thus their orientation in not significant. The translation vector is the coordinates of the ArUco marker corners translated to the camera coordinate system. Using simple geometry, we can deduce the center point of the square marker by calculating the means of the corners' x and y coordinates.

In order to command the robot to move its end effector, the position must be relevant to the robot's base coordinate system. To do that we use the homogeneous transformation matrix H, that represents position and orientation of one coordinate frame relative to another coordinate frame.

$$p^{ROB} = H^{ROB}_{CAM} \cdot p^{CAM}$$

$$\begin{bmatrix} x^r \\ y^r \\ z^r \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}^r_c & \boldsymbol{t}^r_c \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix}$$

To find the homogeneous transformation matrix we need the rotation matrix, R and the translation vector, t.

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \boldsymbol{t} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The camera pose relative to the robot base can be acquired using the get_current_pose function. In ROS the preferred representation for orientation are quaternions, a 4-tuple representation (x, y, z, w). A unit quaternion q can be converted to a rotation matrix R as follows:

$$\boldsymbol{R} = \begin{bmatrix} 1 - 2{q_y}^2 - 2{q_z}^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z - 2q_yq_w \\ 2q_xq_y - 2q_zq_w & 1 - 2{q_x}^2 - 2{q_z}^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z - 2q_xq_w & 1 - 2{q_x}^2 - 2{q_y}^2 \end{bmatrix}$$

The same transformation has to be performed to get the vector that describes the pose of the marker in the camera coordinate system. By multiplying the two terms together we get the position of the object in the robotic arm reference frame.

4.5 Robot Control and Motion Planning

For the robot motion MoveIt! was used to define the required control groups both for the arm and the vacuum grippers, perform the motion planning and solve the inverse kinematics. Motion plan creation by MoveIt is advantageous since only the goal pose needs to be defined if the motion path is irrelevant.

What we want from the motion planner to perform is specified in the motion plan request. Typically, the motion planner is asked to move the arm to a new pose or the end effector to a different location. By default, collisions, including self-collisions, are checked. The desired trajectory will be generated to move the arm according to the plan request. It should be noted that the result will be a trajectory, not just a path, meaning that velocity and acceleration constraints at the joint level must be taken into consideration.

A motion planner is a part of the entire motion planning pipeline, which also includes planning request adapters. Pre-processing motion plan requests and post-processing motion plan responses are both possible with planning request adapters. Pre-processing is helpful in a variety of circumstances, such as when a robot's initial state is just barely outside of its defined joint limits. Other processes, such converting paths generated for a robot into time-parameterized trajectories, require post-processing.

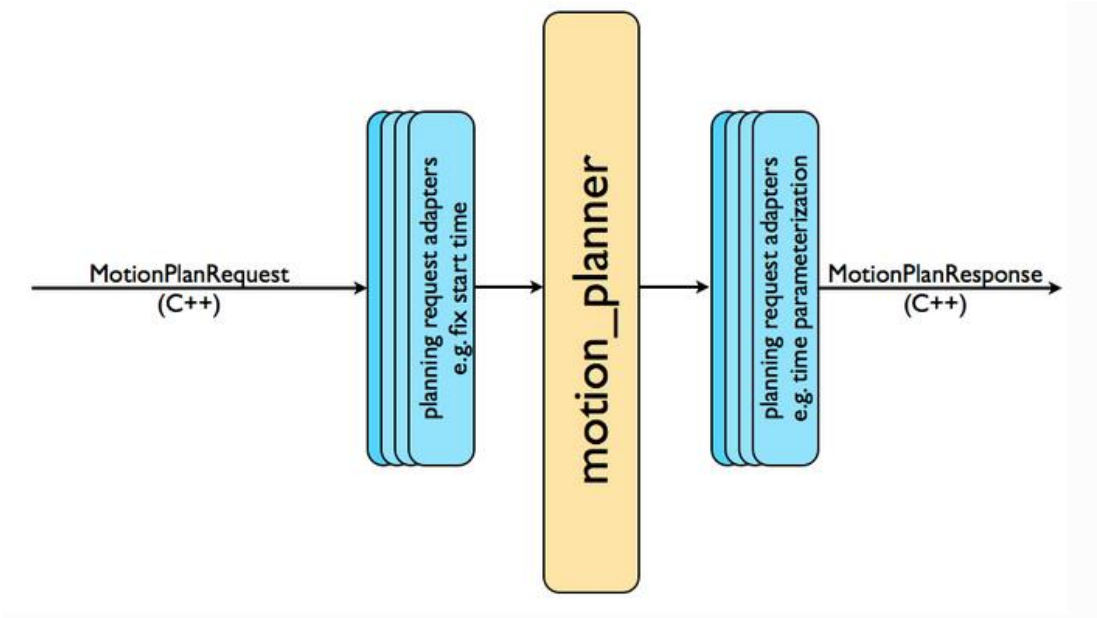

Figure 4.10 : Motion planning pipeline

To generate the MoveIt configuration package the setup assistant tool GUI can be used. The URDF/xacro file that defines the robot is the only file needed to execute this tool. Also, the ROS Industrial package provides all the necessary MoveIt configuration YAML files for the UR5 manipulator, including controllers, joint limits, kinematic solvers and OMPL planning.

One of the simplest and most useful MoveIt user interfaces is through the Python-based Move Group Interface. The functionality for all the needed operations can be provided by these wrappers. To use the Python MoveIt interfaces, the moveit_commander namespace must be imported in the python script responsible for robot movement. This namespace provides us with a MoveGroupCommander class, which is an interface to a planning group or group of joints, and a RobotCommander class, which has information about the robot’s kinematic model and the current joint states. Additionally, a GripperCommander class was created in order to turn the grippers on and off.

In order to plan a motion for the robot arm, such as picking or placing an object and moving to home pose, we require a start and a goal state. For the starting state, we can acquire the robot's pose using the get_current_pose() function. For the goal state we can just define the x,y,z coordinates the arm's end effector should end up after the motion is completed. These coordinates are predefined if the robot needs to move to the home position, or if it is required to place a cylinder in one of the three stacks. Of course, in the placing case, the z-coordinate adjusts depending on the height of the tower. In the picking scenario the coordinates are provided from the ArUco detection and they correspond to the cylinders position in the world.

The Cartesian space path planning is implemented by the use of compute_cartesian_path() function. This function computes a sequence of waypoints to make the end effector move in straight line segments in order to move from the start to the goal state. A value is requested, as a parameter, to determine the distance that configurations are computed, in our case that distance is 2 centimeters. The system computes the joint positions using inverse kinematics for each interpolated waypoint and the planned path is, finally, executed. In Figure 4.10 the start state (green) and the goal state (orange) of a simple motion are depicted. The Cartesian path and the waypoints of the computed motion plan are shown in Firure 4.11.

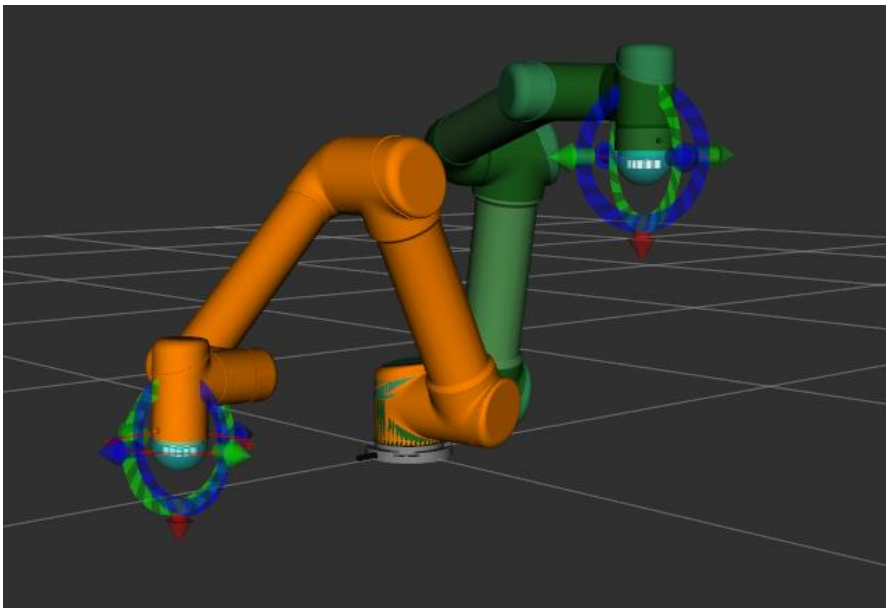

Figure 4.11 : Start and Goal State

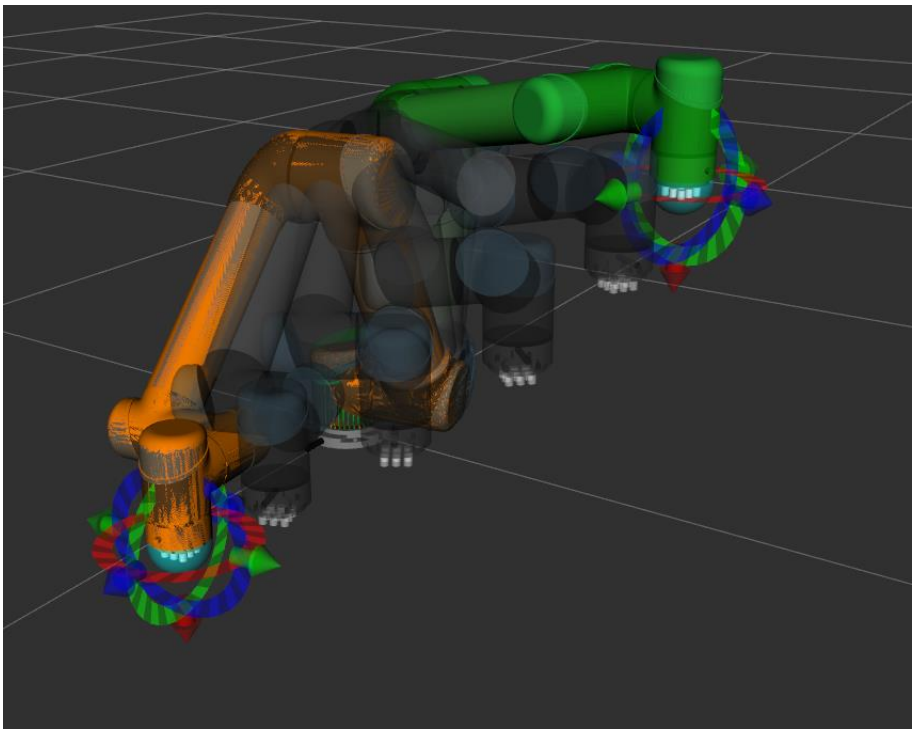

Figure 4.12 : Motion planning

Chapter 5

Results

An overview of the whole approach can be viewed using rqt_graph, a tool that offers graphical representation of all the nodes and the topics used for communication between them.

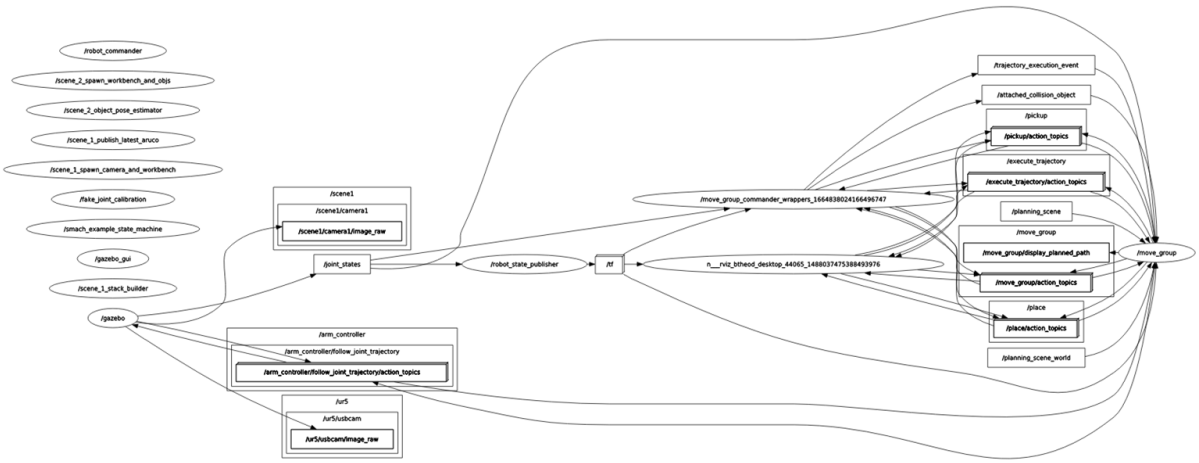

Figure 5.1 : ROS Nodes Graph showing all nodes and topics

5.1 Simulation

Complete simulations of the robot stacking all the cylinders, in order to recreate the desired towers, is presented step by step in the figures below. First, an example with 8 cylinders:

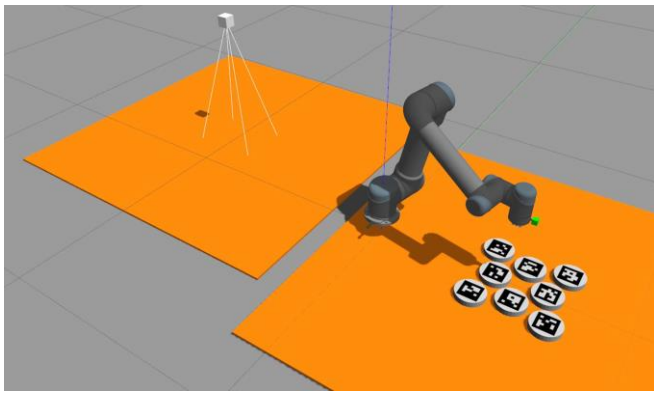

Figure 5.2 : Example 1, Simulation start

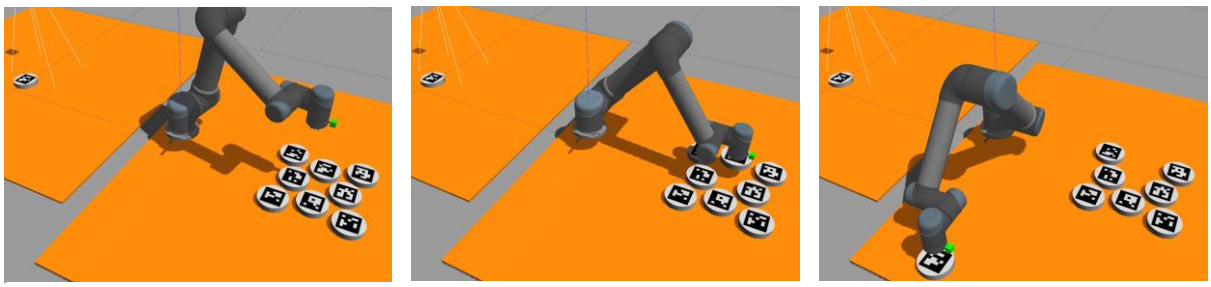

Figure 5.3 : Example 1, Iteration 1. Detection (left) Picking (center) Placing (right)

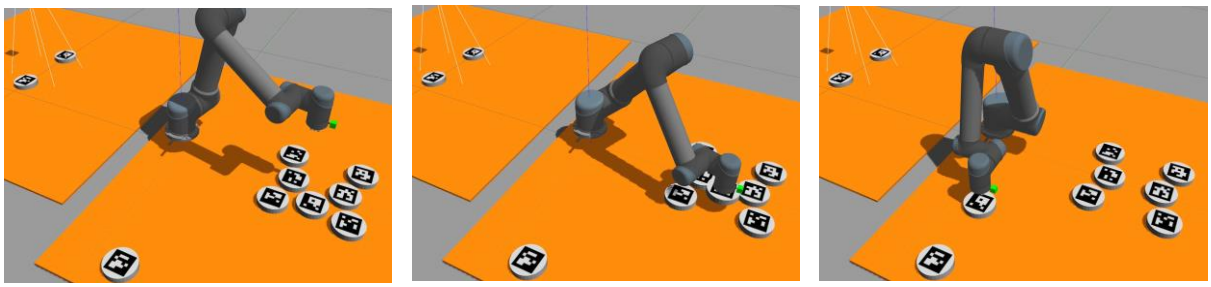

Figure 5.4 : Example 1, Iteration 2

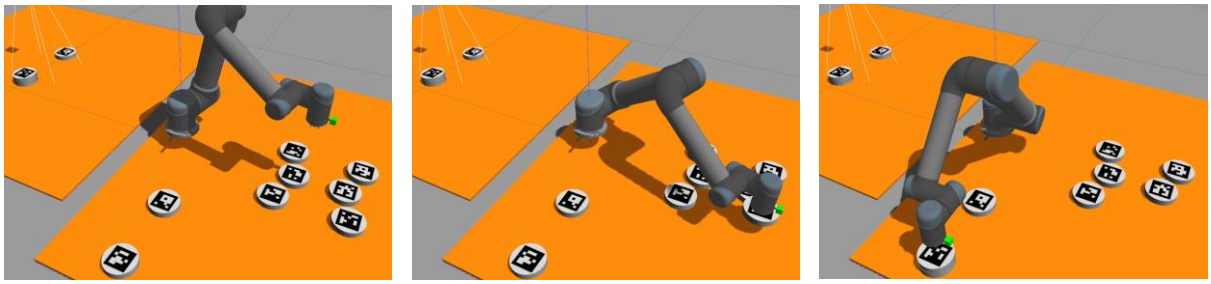

Figure 5.5 : Example 1, Iteration 3

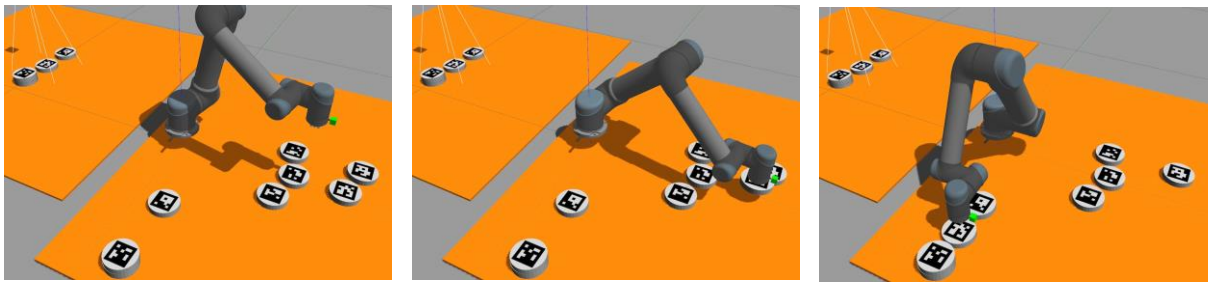

Figure 5.6 : Example 1, Iteration 4

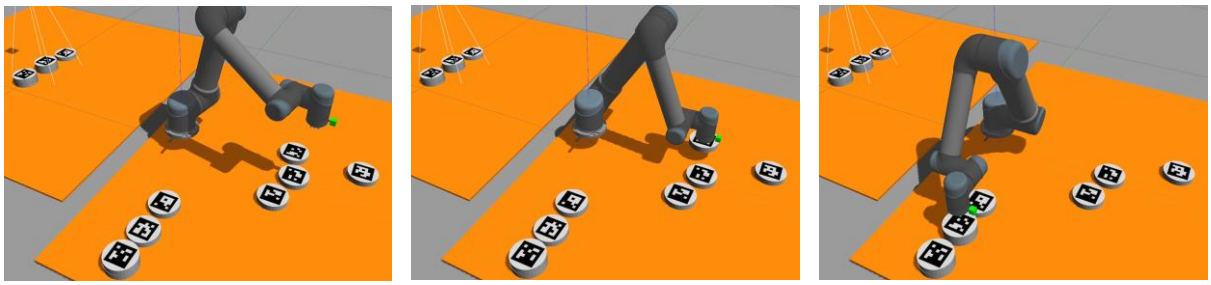

Figure 5.7 : Example 1, Iteration 5

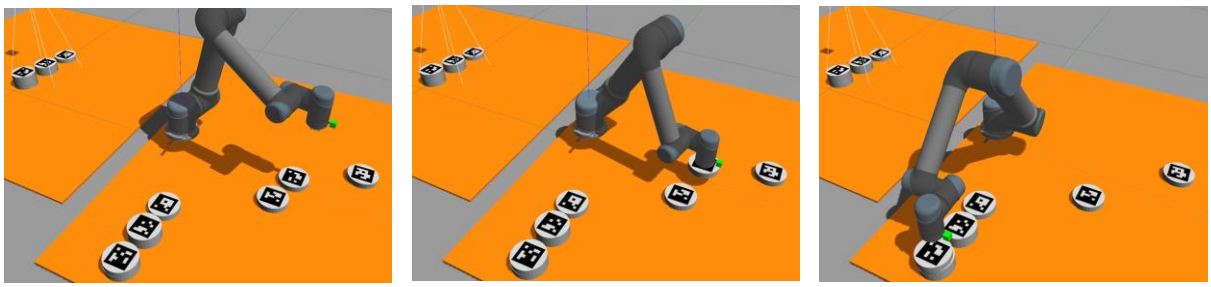

Figure 5.8 : Example 1, Iteration 6

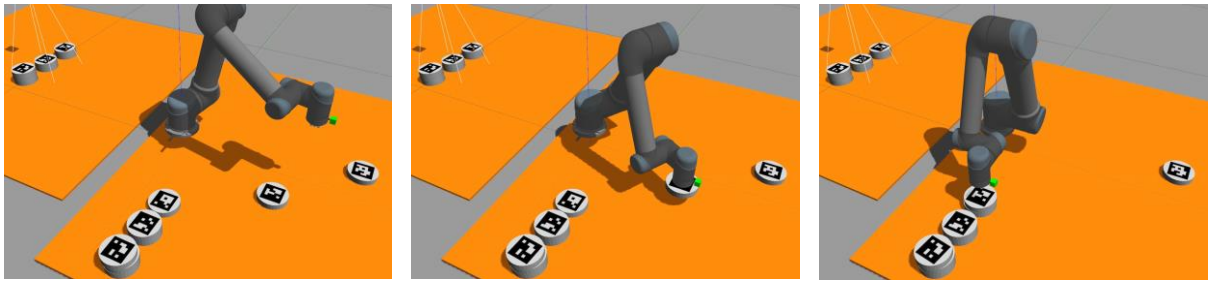

Figure 5.9 : Example 1, Iteration 7

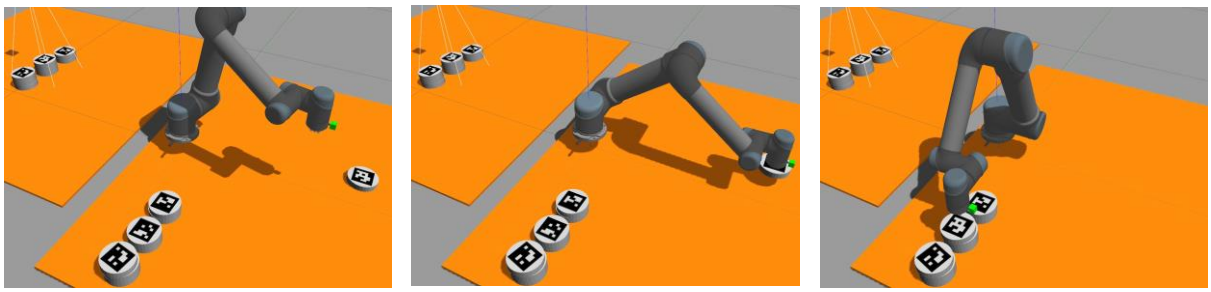

Figure 5.10 : Example 1, Iteration 8

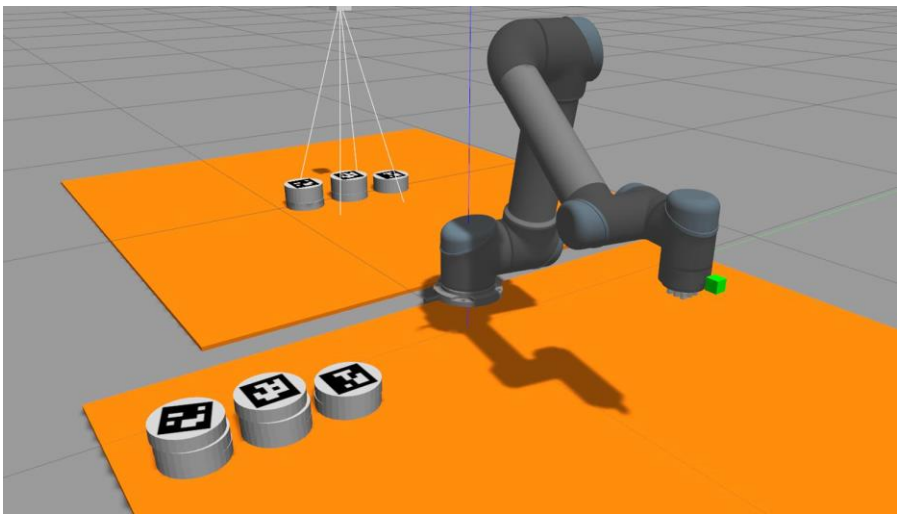

Figure 5.11 : End of simulation (8 cylinders)

Simulation ends when all cylinder blocks are stacked. The final result is two identical sets of towers.

In the second example, a simulation with 10 cylinders is presented:

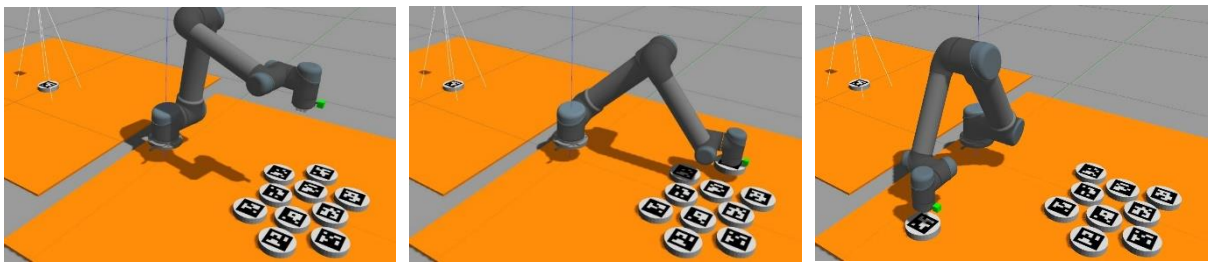

Figure 5.12 : Example 2, Iteration 1

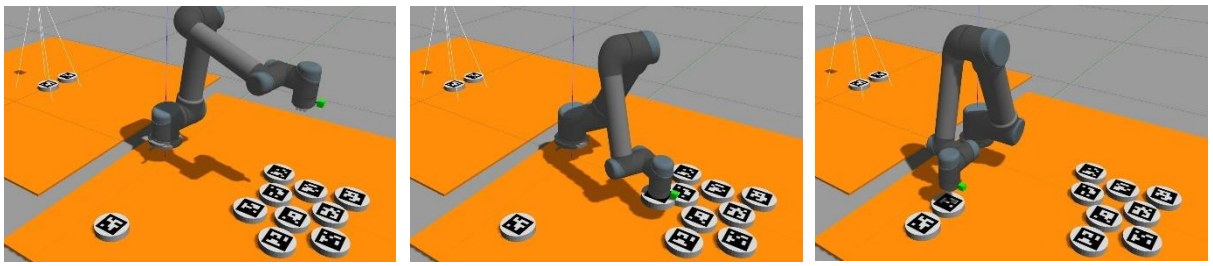

Figure 5.13 : Example 2, Iteration 2

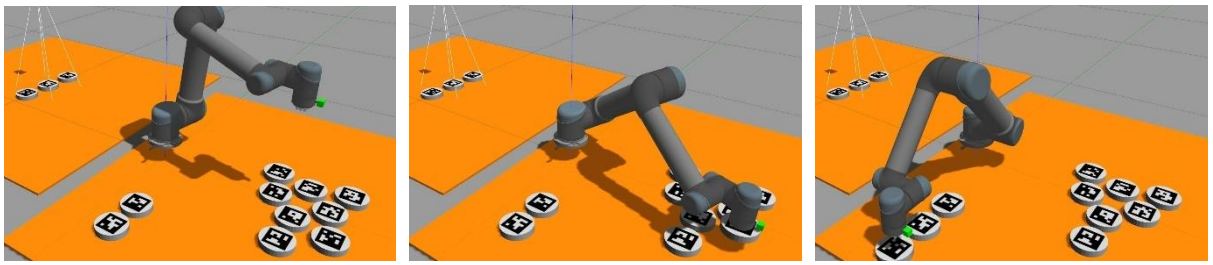

Figure 5.12 : Example 2, Iteration 3

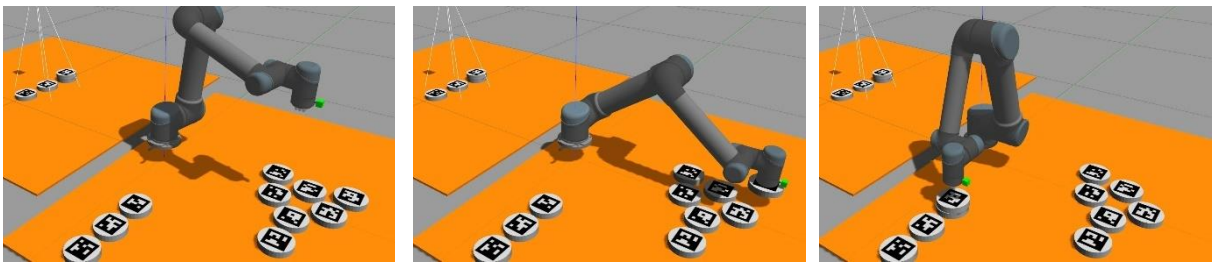

Figure 5.13 : Example 2, Iteration 4

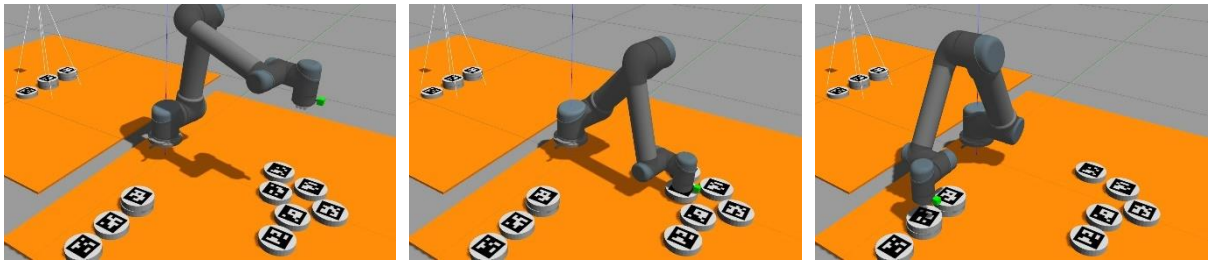

Figure 5.14 : Example 2, Iteration 5

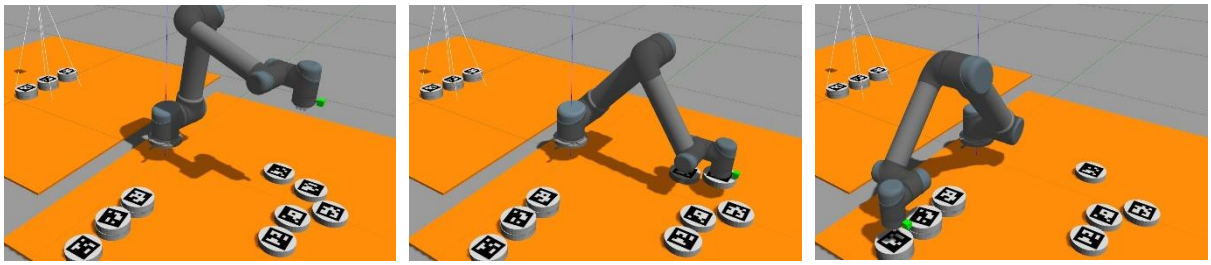

Figure 5.15 : Example 2, Iteration 6

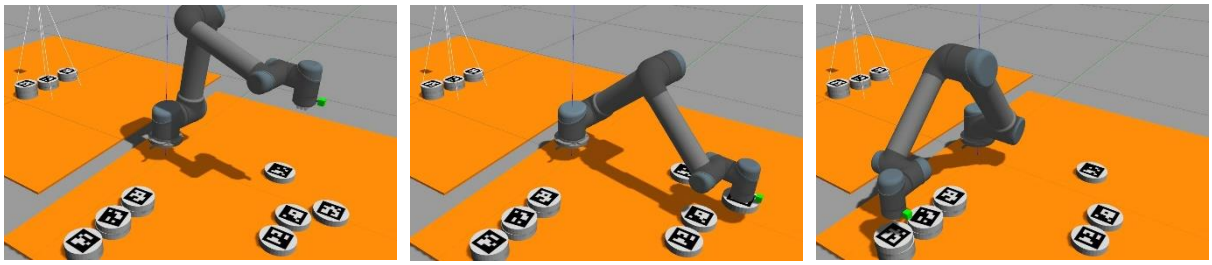

Figure 5.16 : Example 2, Iteration 7

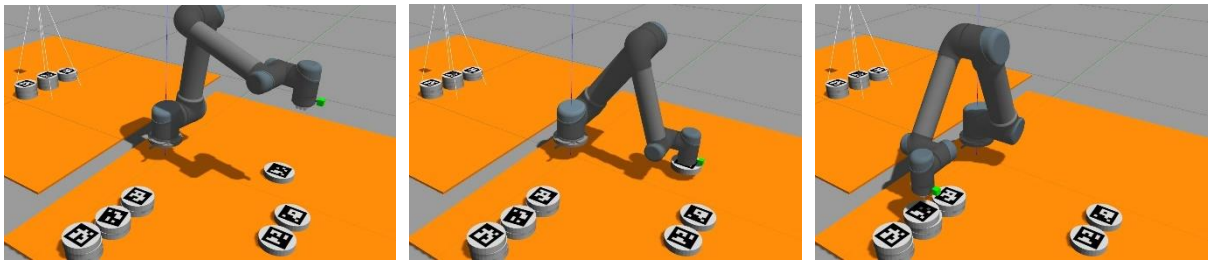

Figure 5.17 : Example 2, Iteration 8

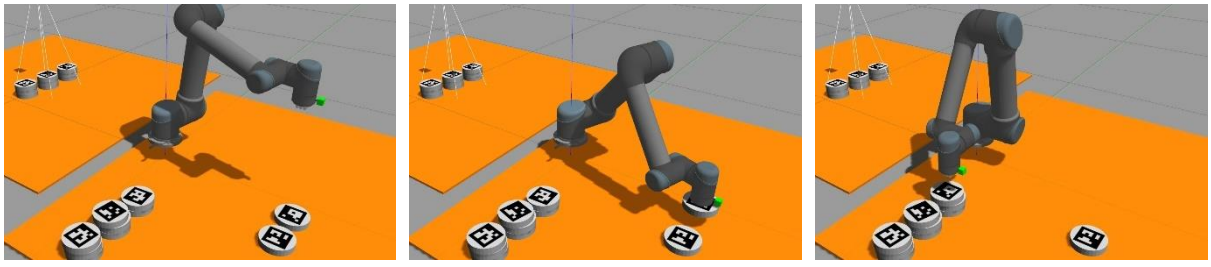

Figure 5.18 : Example 2, Iteration 9

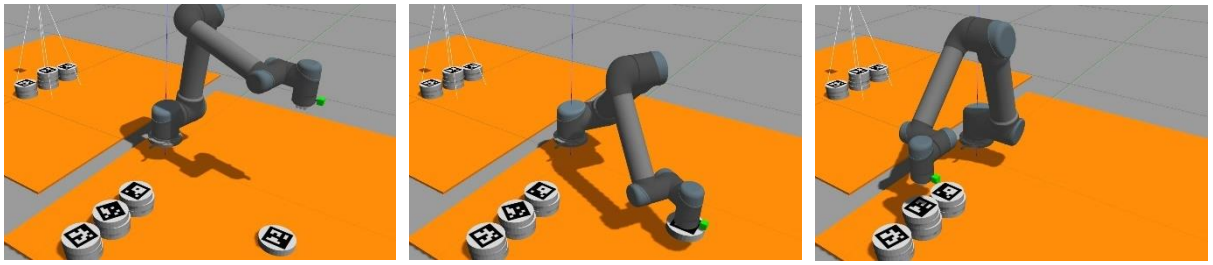

Figure 5.19 : Example 2, Iteration 10

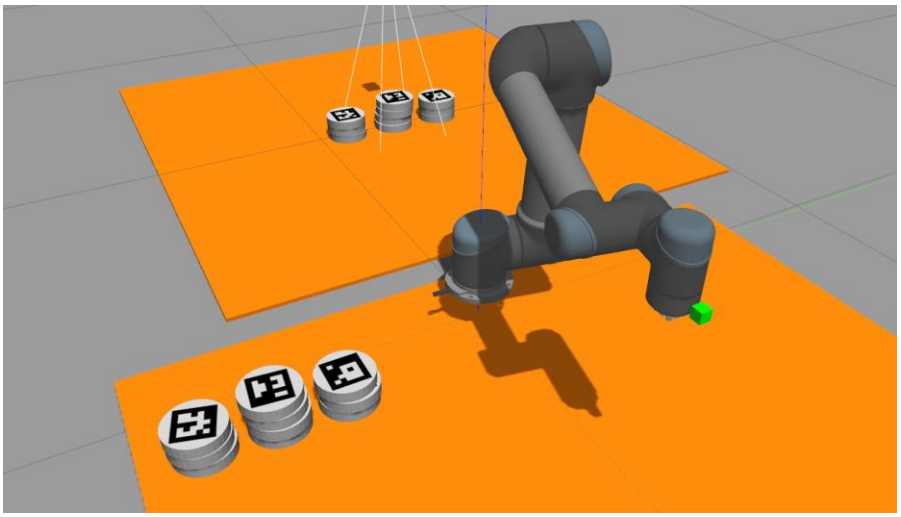

Figure 5.20 : End of simulation (10 cylinders)

In the above examples, we used 8 and 10 cylinders with ArUco markers, even though the project was originally designed to incorporate 16 markers. Unfortunately, the system, this application was developed on, was not able to successfully run the simulation for more than 10 cylinders, since the Gazebo simulator is a resource demanding program. However, the code for all 16 cylinders is implemented and is included in the package.

5.2 Object Pose Estimation

The performance of the robot vision algorithm is shown in the table below. We can conclude that the estimation of the cylinder poses is accurate, since the maximum deviation is 2-3 millimeters. The z axis presents the biggest differences from the true coordinates. Fortunately, the addition of 9 vacuum grippers and the reduced cylinder's mass, enable the robot arm to consistently pick up the blocks. Failure to pick a cylinder is extremely rare and dropping a block mid motion has yet to encountered.

	True world coordinates			Estimated coordinates		
ArUco ID	**x**	**y**	**z**	**x**	**y**	**z**
1	0.5	0	0.025	0.502	0.001	0.028
2	0.7	0	0.025	0.699	0	0.022
3	0.6	-0.1	0.025	0.601	-0.101	0.023
4	0.6	0.1	0.025	0.600	0.102	0.028
5	0.74	-0.14	0.025	0.741	-0.139	0.025
6	0.74	0.14	0.025	0.741	0.140	0.024
7	0.46	-0.14	0.025	0.461	-0.141	0.022
8	0.46	0.14	0.025	0.459	0.141	0.026

Chapter 6

Conclusions

6.1 Conclusion

This thesis describes the implementation of an autonomous 6-DOF robotic manipulator with visual guidance, where the objective is for the robot arm to stack cylinder blocks, recreating randomly generated towers. The blocks have unique ArUco markers displayed on them so they can be identified. The manipulator is equipped with vacuum grippers and a 2D camera attached to its end effector.

The robot receives information about the towers that it is required to copy from a stationary camera, responsible to track their spawning. Since the robotic arm is equipped with a camera, it employs computer vision techniques to locate the desired cylinder. Consequently, it constructs a trajectory to pick and stack each cylinder by making use of motion planning algorithms.

This approach is a proof of concept for a manipulator robot that can perceive its surroundings and operate autonomously, in a closed-loop, according to the changes. In real life scenarios, the prototype towers could be constructed by a human, instead of being randomly generated. Additionally, the information about the sequence that the robot arm has to stack the building blocks, can vary. As an example, in a real-world application if the robot arm was utilized in the warehousing industry, a list of how the packages need to be stacked could be provided to the robot. Subsequently the robot would be able to recognize markers on the packages and place them on wooden pallets according to the requested way, so that they can be conveniently shipped or delivered.

The entire project has been implemented within the Robot Operating System (ROS) and Gazebo 3D robotics simulator and is available as an open source package.

6.2 Future work

6.2.1 Real Robot Application

The results of this project are theoretical and simulation-based. The next step is to test the designed system on an actual robot and observe how the planners and computer vision perform in real-world scenarios. Motion planners probably require adjustment of the planning parameters, to take the environmental disturbances and the actual robots limitation into account, in order to achieve optimal planning process. Also, lighting conditions and environmental noise might hinder the computer vision algorithm's performance.

6.2.2 Motion planners

There is a variety of motion planners available for robotic manipulators. To name a few Rapidly-exploring Random Trees (RRT), Expansive Space Trees (EST) and Path-Directed subdivision Trees (PDST) are all available in the Open Motion Planning Library (OMPL). It is worth investigating if implementing any of those or any other motion planning algorithm, can improve the robots arm precision, solution smoothness or planning time. Additionally, different motion planners could be deployed for more complex environments with introduced obstacles.

References

[1] Ali, M.H., Aizat, K., Yerkhan, K., Zhandos, T., and Anuar, O. (2018). Vision–based robot manipulator for industrial applications. Procedia Computer Science, 133, 205–212.

[2] A. Djajadi, F. Laoda, R. Rusyadi, T. Prajogo and M. Sinaga, "A MODEL VISION OF SORTING SYSTEM APPLICATION USING ROBOTIC MANIPULATOR", Journal.uad.ac.id, 2017.

[3] Robohub. Amazon picking challenge. [Online] Available: https://robohub.org/tag/amazon-picking-challenge/ [Accessed: Sept. 12, 2022]

[4] Juang, Jih-Gau, Yi-Ju Tsai, and Yang-Wu Fan. "Visual recognition and its application to robot arm control." Applied Sciences 5, no. 4 (2015): 851-880.

[5] "Object Sorting System Using Robotic Arm", Journaldatabase.info, 2013. [Online]. Available: http://journaldatabase.info/articles/object_sorting_system_using_robotic.html. [Accessed: Sept. 18, 2022].

[6] Kumar, Rahul, et al. "Object detection and recognition for a pick and place Robot." Computer Science and Engineering (APWC on CSE), 2014 Asia-Pacific World Congress on. IEEE, 2014.

[7] R. Raja and S. Kumar, “A Hybrid Image Based Visual Servoing for a Manipulator using Kinect,” in Proceedings of the Advances in Robotics. ACM, 2017, p. 52.

[8] M. F. Sani, G. Karimian, Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors, in 2017 International Conference on Computer and Drone Applications (IConDA), Nov. 2017, pp. 102–107. DOI: 10.1109/ICONDA.2017.8270408

[9] I. Lebedev, A. Erashov, A. Shabanova, Accurate Autonomous UAV Landing Using Vision-Based Detection of ArUco-Marker, in International Conference on Interactive Collaborative Robotics, Springer, 2020, pp. 179–188. DOI: 10.1007/978-3-030-60337-3_18

[10] P. M. Kebria, S. Al-Wais, H. Abdi, and S. Nahavandi, “Kinematic and Dynamic Modelling of UR5 Manipulator,” in Proceedings of the IEEE International Conference on Systems Man and Cybernetics (SMC), pp. 4229–4234, Budapest, Hungary, 2016.

[11] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. 2014. "Automatic generation and detection of highly reliable fiducial markers under occlusion". Pattern Recogn. 47, 6 (June 2014), 2280-2292. DOI=10.1016/j.patcog.2014.01.005

[12] B. Liang, Y. Cheng, X. Zhu, H. Liu and X. Wang, "Calibration of UR5 manipulator based on kinematic models," *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 3552-3557, doi: 10.1109/CCDC.2018.8407738.

[13] M. N. Hidayati, D. Adzkiya and H. Nurhadi, "Motion Control Design and Analysis of UR5 Collaborative Robots Using Fuzzy Logic Control (FLC) Method," 2021 International Conference on Advanced Mechatronics, Intelligent Manufacture and Industrial Automation (ICAMIMIA), 2021, pp. 162-167, doi: 10.1109/ICAMIMIA54022.2021.9807732.

[14] S. Wahyuningtri, D. Adzkiya and H. Nurhadi, "Motion Control Design and Analysis of UR5 Collaborative Robots Using Proportional Integral Derivative (PID) Method," 2021 International Conference on Advanced Mechatronics, Intelligent Manufacture and Industrial Automation (ICAMIMIA), 2021, pp. 157-161, doi: 10.1109/ICAMIMIA54022.2021.9807805.

[15] Amato, N.M., Wu, Y.: A randomized roadmap method for path and manipulation planning. In: Proceedings of IEEE International Conference on Robotics and Automation. Volume 1. (Apr 1996) 113–120 vol.1

[16] Stilman, M.: Task constrained motion planning in robot joint space. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. (Oct 2007) 3074–3081

[17] Stanford Artificial Intelligence Laboratory et al. Robotic Operating System. Available from: https://www.ros.org

[18] ROS Industrial [Online] Available : https://rosindustrial.org

[19] Ioan A. Sucan and Sachin Chitta. “MoveIt!”. [Online] Available : https://moveit.ros.org.

[20] Ioan A. S¸ucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. IEEE Robotics & Automation Magazine, 19(4):72–82, December 2012. http://ompl.kavrakilab.org.

[21] J. Denavit and R. S. Hartenberg, “A kinematic notation for lower-pair mechanisms based on matrices.” Trans. of the ASME. Journal of Applied Mechanics, vol. 22, pp. 215–221, 1955.

[22] J. Angeles, Rational Kinematics, ser. Springer Tracts in Natural Philosophy. Springer New York, 2013. ISBN 9781461239161. [Online]. Available: https://books.google.se/books?id=WUXTBwAAQBAJ

[23] Kam, H., Lee, S.H., Park, T., Kim, C.H.: Rviz: a toolkit for real domain data visualization. 60 (10 2015) 1–9 12

[24] Foote, T.: tf: The transform library. In: Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on. Open-Source Software workshop (April 2013) 1–6 12, 49

[25] R. L. Williams II, “Notesbook supplement for me 4290/5290 mechanics and control of robotic manipulators,” pp. 71–72, 2014

[26] S. Goto, ed., Robot Arms. Intech, 2011

[27] “OpenCV Online Documentation.” [Online]. Available: https://docs.opencv.org/4.1.1/index.html.

[28] Nate Dimick, Gazebo Fiducial Spawner (2020), GitHub repository. [Online] Available : https://github.com/NateDimick/gazebo_fiducial_spawner

[29] Sezgin, M., Sankur, B.: Survey over image thresholding techniques and quantitative performance evaluation. J. Electron. Imaging 13(1), 146–165 (2004)

[30] Suzuki, S., Abe, K.: Topological structural analysis of digitized binary images by border following. Comput. Vis. Graph. Image Process. 30(1), 32–46 (1985)

[31] Wu, S.-T., da Silva, A.C.G., Márquez, M.R.G.: The Douglas-peucker algorithm: sufficiency conditions for non-self-intersections. J. Braz. Comp. Soc. 9(3), 67–84 (2004)

[32] Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern. 9(1), 62–66 (1979)

Appendix A

Transformation Matrix

The equations in this Appendix form the transformation matrix for the Forward Kinematics of the UR5, as defined in Chapter 2. In these equations $s_{i+j} = \sin(\theta_i+\theta_j)$ and $c_{i+j} = \cos(\theta_i+\theta_j)$. Furthermore the Denavit-Hartenberg (DH) coefficients for di and ai can be found in Figure 2.6. These equations can be used in the cost function in an optimization algorithm to find the Inverse Kinematics.

$$H_6^0 = \begin{bmatrix} R_{11} & R_{12} & R_{13} & x \\ R_{21} & R_{22} & R_{23} & y \\ R_{31} & R_{32} & R_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} R_{11} &= c_6(s_1s_5 + c_{234}c_1c_5) - s_{234}c_1s_6 \\ R_{21} &= -c_6(c_1s_5 - c_{234}c_5s_1) - s_{234}s_1s_6 \\ R_{31} &= c_{234}s_6 + s_{234}c_5c_6 \end{aligned}$$

$$\begin{aligned} R_{12} &= -s_6(s_1s_5 + c_{234}c_1c_5) - s_{234}c_1c_6 \\ R_{22} &= s_6(c_1s_5 - c_{234}c_5s_1) - s_{234}c_6s_1 \\ R_{32} &= c_{234}c_6 + s_{234}c_5s_6 \end{aligned}$$

$$\begin{aligned} R_{13} &= c_5s_1 - c_{234}c_1s_5 \\ R_{23} &= -c_1c_5 - c_{234}s_1s_5 \\ R_{33} &= -s_{234}s_5 \end{aligned}$$

$$\begin{aligned} x &= d_6(c_5s_1 - c_{234}c_1s_5) + d_4s_1 + c_1(a_3c_{23} + a_2c_2) + d_5s_{234}c_1 \\ y &= s_1(a_3c_{23} + a_2c_2) - d_4c_1 - d_6(c_1c_5 + c_{234}s_1s_5) + d_5s_{234}s_1 \\ z &= d_1 + a_3s_{23} + a_2s_2 - d_5c_{234} - d_6s_{234}s_5 \end{aligned}$$