Federal University of Bahia

MASTER THESIS

Trajectory optimization applied to motion planning of industrial manipulators

Miguel Felipe Nery Vieira

Postgraduate Program in Electrical Engineering

Salvador
2022

MIGUEL FELIPE NERY VIEIRA

TRAJECTORY OPTIMIZATION APPLIED TO MOTION PLANNING OF INDUSTRIAL MANIPULATORS

This Master thesis was submitted to the Postgraduate Program in Electrical Engineering from the Federal University of Bahia, as partial requirement for the degree of Master in Electrical Engineering.

Adviser: Prof. Dr. André Gustavo Scolari Conceição

Salvador

2022

V658 Vieira, Miguel Felipe Nery.
Trajectory optimization applied to motion planning of industrial manipulators/ Miguel Felipe Nery Vieira. – Salvador, 2022.
48 f.: il. color.

Orientador: Prof. Dr. André Gustavo Scolari Conceição.

Dissertação (mestrado) – Universidade Federal da Bahia. Escola Politécnica, 2022.

1. Robôs industriais. 2. Robótica. 3. Trajetória - otimização. I. Conceição, André Gustavo Scolari. II. Universidade Federal da Bahia. III. Título.

CDD: 629.893 3

TERMO DE APROVAÇÃO

MIGUEL FELIPE NERY VIEIRA

TRAJECTORY OPTIMIZATION APPLIED TO MOTION PLANNING OF INDUSTRIAL MANIPULATORS

Esta Dissertação de Mestrado foi julgada adequada à obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Bahia.

Salvador, 08 de Setembro de 2022

Prof. Dr. André Gustavo Scolari Conceição
Universidade Federal da Bahia

Prof. Dr. Tiago Trindade Ribeiro
Universidade Federal da Bahia

Prof. Dr. Rodrigo Antônio Marques Braga
Universidade Federal de Santa Catarina

ACKNOWLEDGEMENTS

A todas e todos que acreditaram e me apoiaram de alguma forma nessa jornada, meus mais sinceros agradecimentos. Em especial, agradeço:

A meus pais e irmã, Dulcinea Nery, Antenor Nunes e Lara Camila por sempre apoiarem minhas decisões e impulsionarem meus sonhos e objetivos. Sem vocês, nada disso seria possível.

A Ederson Reis, pelas palavras de conforto nos momentos mais difíceis, pelo companheirismo, por me dar forças sempre que necessário.

A Cássio de Castro e Ronny Brayner, por toda a amizade, sempre presentes apesar da distância.

A Aline Alencar, Luan Freitas e Stefanne Santos por terem sido minha segunda família em Salvador.

Aos amigos do programa Novos Talentos em Robótica e Sistemas Autônomos, do Senai Cimatec, em especial Anderson Queiroz, Aziel Freitas, Diogo Martins, Jean Paulo, Jéssica Motta, Leonardo Lima, Lucas Silva, Mateus Cerqueira, Marco Reis, Oberdan Pinheiro, Pedro Tecchio, Rebeca Lima, Rodrigo Formiga, Tiago Souza e Vinícius Felismino, por todo o aprendizado proporcionado.

A André Scolari por aceitar me orientar neste trabalho e por toda ajuda para o bom desenvolvimento do mesmo.

A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original.

—ALBERT EINSTEIN

RESUMO

Manipuladores robóticos estão cada vez mais presentes em diversas atividades, dentro e fora da indústria. A utilização destes robôs permite maior precisão e exatidão na realização nas tarefas, porém, é importante levar em consideração alguns fatores que garantam a segurança do sistema, como a capacidade de evitar obstáculos que possam estar presentes no ambiente operacional e a qualidade da trajetória final gerada. Neste trabalho, um sistema para otimização de trajetória de um robô manipulador em ambientes complexos é implementado, utilizando os algoritmos Covariant Hamiltonian Optimization for Motion Planning (CHOMP) e Stochastic Trajectory Optimization for Motion Planning (STOMP). Um sensor visual do tipo RGB+D é integrado para ao sistema detecção de obstáculos no ambiente de operação. O sistema é baseado no *framework open-source Robot Operating System*(ROS) e é aplicado a uma célula de manufatura aditiva composta pelo robô colaborativo UR5 para a realização de tarefas de *pick and place*. Após uma série de execuções em cenários simulados e reais, os algoritmos que compõem o sistema foram comparados com base em sua taxa de sucesso, tempo de planejamento e duração da trajetória gerada. Os resultados obtidos indicam a capacidade do sistema de gerar trajetórias otimizadas e livres de colisões em ambiente estático.

Palavras-chave: Manipuladores robóticos, Robôs colaborativos, Otimização de trajetória, ROS, Moveit.

ABSTRACT

Robotic manipulators are becoming more present in various activities, inside and outside the industry. The use of these robots allows greater precision and accuracy in carrying out the tasks, however, it is important to take into account some factors to ensure the safety of the system, such as the ability to avoid obstacles that may be present in the operating environment and the quality of the final path. In this work, we implement a system for trajectory optimization of a robotic manipulator in static environments using the algorithms Covariant Hamiltonian Optimization for Motion Planning (CHOMP) and Stochastic Trajectory Optimization for Motion Planning (STOMP). We integrated an RGB+D sensor to the system for obstacle detection on manipulator's workspace. The system is based on open-source framework *Robot Operating System* (ROS) and it is applied to pick and place tasks in an additive manufacturing cell composed by the collaborative robot UR5. After a series of executions on real and simulated scenarios, the algorithms were compared based on their success rate, planning time, and duration of the generated trajectory. Results indicate that the proposed system can generate feasible and collision-free trajectories in static environments.

Keywords: Robotic manipulators, Collaborative robotics, Trajectory optimization, ROS, Moveit.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI Artificial intelligence

CHOMP Covariant Hamiltonian Optimization for Motion Planning

DH Denavit-Hartenberg

DoF Degrees of Freedom

FoV Field of View

IoT Internet of Things

ROS Robot Operating System

STOMP Stochastic Trajectory Optimization for Motion Planning

URDF Unified Robot Description Format

SDRF Semantic Robot Description Format

LIST OF SYMBOLS

$\bar{\nabla}$	Functional gradient
λ	Trade-off parameter
$\mathbb{E}$	Expectation
$\mathcal{M}$	STOMP cost function
$\mathcal{U}$	CHOMP cost function
ψ	Trajectory
θ_i	Angle of rotation
ξ_E	Pose of end effector
d_i	Joint displacement
I	Identity matrix
p	Position vector
q_i	Joint Variable
R	Rotation matrix

Chapter 1

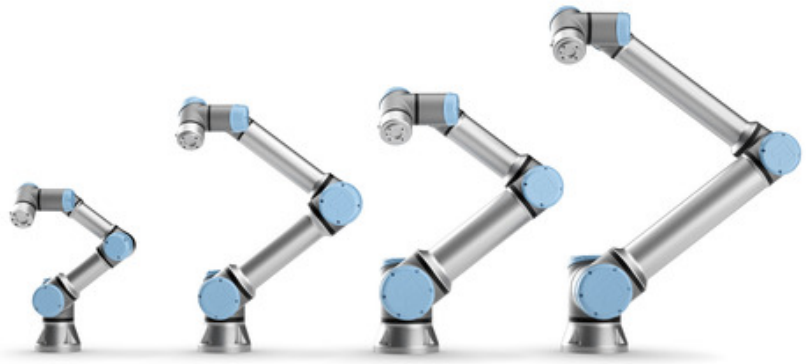
INTRODUCTION

The first known industrial robot was developed by George Dovel, in the early 1960s (GASPARETTO; SCALERA, 2019). The Unimate worked on a vehicle production line and had the function of lifting hot metal parts from a casting machine and packing them, a dangerous task to be performed by human operators. The manipulator was installed for the first time ah the automotive plants of the company General Motors, which became the most automated factory in the world, producing 110 cars per hour, more than double the standard production rate at that time, thus causing a revolution in the automotive industry (IEEE, 2018).

Over the years, several other models of robotic manipulators have emerged, such as the Stanford Arm, an all-electric arm with 6 Degrees of Freedom (DoF), developed by Victor Scheinman at Stanford University in 1969 (INFOLAB, 2019), and the ASEA IRB 6, the first arm to have a microcontroller, developed by ASEA, today ABB, in 1975. The mechanism had 5 DoF, 6kg of load capacity and it was the first of a series of manipulators that keeps evolving until the present time (INFORMATION, 2021).

With the arrival of industry 4.0, a new model of manufacturing it's expected to be created. The fourth industrial revolution features smart and autonomous systems connected and working collaboratively. This revolution is only possible due to the development and adoption of technologies, such as additive manufacturing, blocking chain, Artificial intelligence (AI), Internet of Things (IoT), and robotic systems (OLSEN; TOMLIN, 2019). Despite some challenges, mainly related to the workers' qualification and the digital transformation needed for this new manufacturing concept, industry 4.0 provides a large number of benefits, for example reduction of energy consumption, reduction of overproduction and waste, flexible production, more modular products, increased productivity and efficiency in operations (MOHAMED, 2018).

Figure 1.1: Collaborative UR Series from Universal Robots (from left to right: UR3, UR5, UR10 and UR16).



Source: (ROBOTS, 2021)

Industrial arms usually work isolated from humans, and their ability to share the workspace with operators is a key factor for the industry 4.0 materialization. In smart factories, arms will be equipped with sensors and AI to allow human-robot collaboration. The safety mechanisms can be obtained from a combination of several technologies, such as proximity sensors and force limitations. The use of collaborative robots, so-called "cobots", allows more safety for humans involved in the production process, as well as greater reliability, repeatability, and quality in performed tasks (EVJEMO et al., 2020). A good example of cobots are the UR series robots, Figure 1.1, from Universal Robots.

The manufacturing industry is responsible for most of the applications for collaborative robots, especially in automotive industries. These robots have been mainly used on pick and place tasks, welding, assembling items, handling materials that can be dangerous for human and product inspections. The cobots can perform these tasks with high accuracy, reduced operational costs, and increased security for the operator (SHERWANI; ASAD; IBRAHIM, 2020). To avoid accidents while performing the tasks imposed on the manipulator, it is important to have effective motion planning that guarantees a feasible path, free from obstacles and that respects imposed restrictions.

In this work, we implement a system for the optimization of trajectories using the algorithms Stochastic Trajectory Optimization for Motion Planning (STOMP) (KALAKRISHNAN et al., 2011) and Covariant Hamiltonian Optimization for Motion Planning (CHOMP) (ZUCKER et al., 2013) applied to manipulators inserted in an environment with obstacles, which are detected using an RGB+D sensor. The proposed system is ap-

plied to a robotic additive manufacturing unity, in the context of the FASTEN research project (CONCEIÇÃO et al., 2020).

1.1 LITERATURE REVIEW

The motion planning of manipulators has received a lot of attention in the last years and a large number of works related to this theme have been published Optimization-based planners, such as CHOMP and STOMP, can generate relatively simple solutions to problems with higher DoF in limited spaces, providing more consistent and reliable results, and they have been gaining popularity among robotics community (PENG et al., 2021).

Solutions can be developed based on these algorithms to reach collision-free operations and optimize defined cost functions, such as in (KADEN; THOMAS, 2019). In this paper, the authors maximize the manipulator's manipulability by adding it as a state cost. The system used a combination of STOMP and Gaussian Mixture Models for additional optimization of trajectories generated by RRT-Connect (KUFFNER; LAVALLE, 2000). The algorithms were evaluated in a simulated confined environment and results indicated that the proposed optimization method provides a considerable rase of the robot's manipulability. In (PAVLICHENKO; BEHNKE, 2017), an optimization method based on STOMP is presented, the STOMP-NEW. The inclusion of the speed as a state cost made possible the optimization of trajectory duration. The system also takes into account torque costs, orientation constraints, obstacles, and joint limits. Experiments were carried out in a Momaro 7 DoF manipulator, and results were obtained with high success rates and short execution time, characterizing the proposed method as applicable in situations that require frequent replanning in dynamic environments.

The use of Robot Operating System (ROS) framework (QUIGLEY et al., 2009) allows that the simulation of robots can be evaluated with fidelity, reliability, and low cost. In (YE; SUN, 2020), it's presented the application of motion planning techniques for a 7 DoF robotic manipulator. The authors used ROS and Moveit, a robotic manipulation tool, to establish the necessary configurations for planning tasks and the RRT (LAVALLE; JR, 2001) algorithm was selected as planner. For obstacle avoidance motion planning, a sensor was installed on the manipulator, and the environment was monitored using Moveit perception tools. Simulation tests indicated the efficiency of the RRT algorithm in generating obstacle-free trajectories for redundant manipulators (7DoF). A novel system for trajectory programming based on learning by demonstration techniques is proposed in (ZHANG et al., 2020). The user moves a color marker with the desired path, and the system track and records the information, teaching to the robot and allowing execution

after the demonstration. This way, users unfamiliar with programming techniques can easily send trajectories to the manipulator. Thanks to the development based on ROS, the system is compatible with different types of robots. Experimental results were conducted in a collaborative manipulator UR5, and results indicate that the system can reduce the ergonomic stress of operators, as it allows a simpler and more intuitive programming of the robot's trajectory.

Applications involving ROS for manipulators are often related to Moveit (CHITTA; SUCAN; COUSINS, 2012), as will be discussed later in this work. In (GRUSHKO et al., 2020), authors realized a benchmarking to optimize the parameters related to perception and trajectory planning of Moveit. Different scenarios were performed in a motion planning task with obstacles, using simulated and real models of the UR3 robotic manipulator. The metrics used in the benchmarkings followed two indicators: time taken for the solver to generate the trajectory, where less computational time is considered as high performance, and execution time of the planned trajectory, where trajectories of shorter durations are considered as high performance. Additionally, the percentage of planning and execution successes are also measured. A defined number of variation between the parameters were tested, each one performed 30 consecutive times for the studied trajectory planning problem, and results statistically analyzed in order to verify the influence of each parameter on the system performance.

One of the most related application for robotic manipulators is pick and place of objects. In (JUNG et al., 2020), a simulation with object manipulation using Gazebo and Moveit is presented. The authors applied deep learning techniques for object detection and Gazebo's grasp plugins for gripper control. The system has been developed using the UR5 manipulator integrated with robotiq's 2F-85 gripper. A system based on the collaborative robot UR5 and Moveit is also proposed in (KUMAR et al., 2017). In this work, the authors developed an autonomous system for pick and place tasks in warehouses. The detection of the object to be manipulated, among 29 possibilities, was in charge of convolutional neural networks, allied to the Moveit perception tool and sensor data. Tests were performed with the robot in a real scenario, and results indicate an average task execution time of 24 seconds and accuracy of 90 % for objects detection.

When talking about collaboration between humans and robots, the robot must have the ability to plan its trajectory free from collisions with humans and other obstacles in the work environment. In (BRITO et al., 2018), two path planning algorithms, RRT and PRM (KAVRAKI et al., 1996), were tested and compared in a virtual model of the collaborative manipulator UR5. The proposed system uses a Kinect sensor to perceive the environment. Through interconnected ROS nodes, sensor data is available to Moveit, which is responsible to plan and execute collision-free trajectories. A novel motion plan-

ning algorithm based on RRT and Memory-Goal-Biasing is proposed in (HAN et al., 2018). As a way to escape the local minimum problem, the algorithm memorizes the exploration information in the goal extension and then selects the closest node to the goal expecting the memorized nodes. Additionally, to improve obstacle avoidance and the efficiency of path planning, the obstacle area was extended according to the radius of the manipulator. The algorithm was implemented on a simulation developed on ROS and Moveit, using the redundant manipulator Baxter robot in different scenarios with obstacles. Results indicated that the proposed method has better optimization performance and lower computation complexity than other RRT-based algorithms.

1.2 OBJECTIVES OF THIS WORK

The main objective of this work is to implement a system for the optimization of trajectories of a robotic manipulator inserted in an environment with obstacles using each one of the algorithms STOMP and CHOMP, with visual feedback for obstacle detection. In order to accomplish the main objective, the following specific objectives were achieved:

- study and analyze the model of UR5 robotic manipulator, from Universal Robots, in which the chosen algorithms will be applied;
- implement in C++ or Python on ROS the techniques proposed by (KALAKRISHNAN et al., 2011) and (ZUCKER et al., 2013) to solve manipulator motion planning;
- use visual data from an RBG+D sensor to detect obstacles;
- evaluate the performance of the implemented techniques in a pick-and-place situation with obstacles on real scenario containing an additive manufacturing unity, composed by a 3D printer and a robotic arm UR5;
- analyze and compare the results of the motion planning algorithms based on planning time, path duration, and success rate.

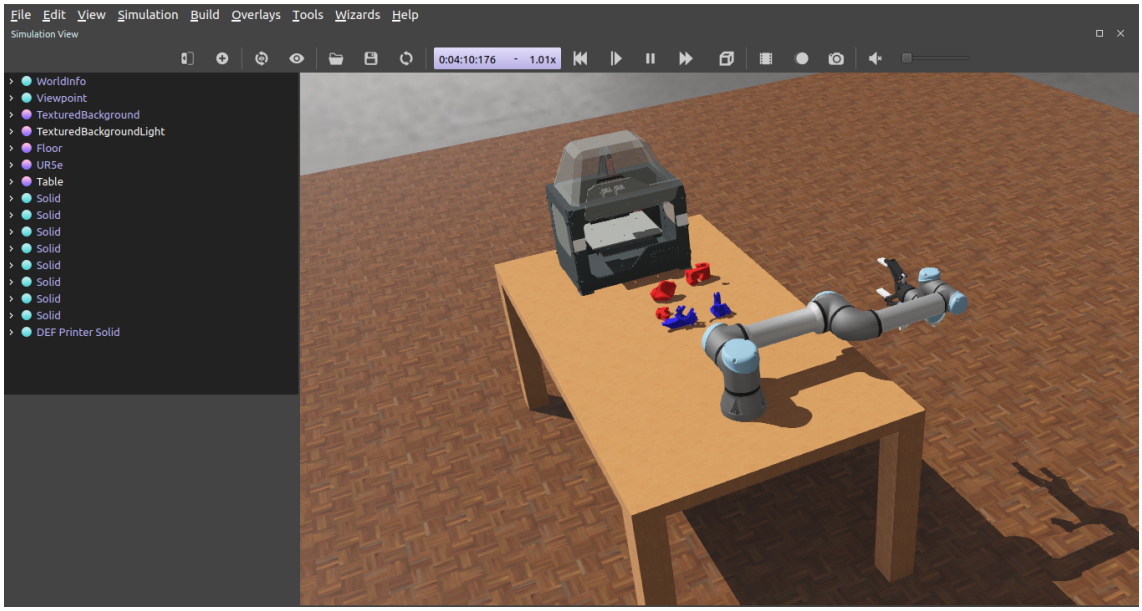
1.3 METHODOLOGY

The first stage of this work consisted of a literature review on the theme in order to understand the state of the art for obstacle-free trajectory optimization algorithms, focusing on techniques based on STOMP and CHOMP.

The implementation took place in the Robot Operating System (ROS) (ROS, 2017) framework, together with Moveit (CHITTA; SUCAN; COUSINS, 2012). The routine

execution was developed in Python language using open source packages available on ROS. The selected collaborative robot was the UR5, a six DoF collaborative robot from Universal Robots (ROBOTS, 2022), with all the models and drivers being freely available in GitHub. The simulation was developed on Webots (CYBERBOTICS, 2021), shown in Figure 1.2, an open source simulator that can provide realistic and complex robotic simulations.

Figure 1.2: Webots simulation containing a collaborative manipulator UR5 in an additive manufacturing cell.



Source: Own authorship

The use case selected for the experiments was a pick and place task on a real scenario with obstacles containing an additive manufacturing unity, composed by a 3D printer and the UR5 manipulator. The robot must take an object located inside of the 3D printer and place it on the desired position. The task should be accomplished avoiding collisions with the environment, which is assumed to be previously known and static. Information about obstacles position will be provided by an RBG+D sensor. The CAD environment was developed to reproduce the real workspace of the robot at the Federal University of Bahia.

After the simulated experiments, the algorithms were validated in the real additive manufacturing cell, shown in Figure 1.3. The obtained results will be discussed in this document.

Figure 1.3: Additive manufacturing cell of robotics laboratory from the Federal University of Bahia.

Source: Own authorship

1.4 PUBLICATIONS

This work provides analysis and comparison of motion planning optimization algorithms for collaborative robots in environments with obstacles. The results obtained during this work allowed the publication of the following scientific paper:

- Miguel Felipe Nery Vieira and André Gustavo Scolari Conceição, 2020, Motion planning using stomp applied to UR5 manipulators. In: III Brazilian Humanoid Robot Workshop (BRAHUR) and the IV Brazilian Workshop on Service Robotics (BRASERO). DOI: doi://10.29327/118637.1-5

- Miguel Felipe Nery Vieira and André Gustavo Scolari Conceição, 2022, Trajectory Optimization for a Collaborative Robot UR5 in a Scenario with Obstacles. In XXIV Congresso Brasileiro de Automática (CBA 2022). No prelo.

1.5 ORGANIZATION OF THIS WORK

The remainder of this work is organized as follows. Chapter 2 presents the basis of the modeling techniques for robotics. In Chapter 3, the components of the proposed system are discussed. The experimental results of this work are presented and discussed in Chapter 4. Conclusions and future works are provided in Chapter 5.

Chapter 2

MODELING

This chapter will present a theoretical overview of forward, inverse and differential kinematics for robot manipulators, focusing on the cobot UR5.

2.1 ROBOT MANIPULATOR

A robot manipulator can be described as a series of rigid elements connected by joints, that can be simple, such as prismatic and revolution, or more complex, such as ball and socket joints. These joints allow that these rigid elements, called links, to move with each other and the number of joints determines the manipulator's degrees of freedom. All this structure forms a kinematic chain. One of the ends of this chain is called base and at the other, an end effector can be integrated, allowing manipulation of objects, with characteristics that can vary depending on the application.

Assuming that a robot manipulator has n joints, it can be said that it will have $n+1$ links. Following the established convention in (SPONG; HUTCHINSON; VIDYASAGAR, 2005), we number the links from 0 to n, starting from the base, and the joints from 1 to n. The joint i connects the link $i-1$ to link i. As link 0 represents the base, it will be fixed. For each i^{th} joint, we can associate a joint variable, q_i. For prismatic joints, q_i is the joint displacement and for revolution joints, q_i is the angle of rotation:

$$q_i = \begin{cases} d_i, \text{ for prismatic joints} \\ \theta_i, \text{ for revolution joints} \end{cases} \tag{2.1}$$

For each link i, a coordinate frame $o_i x_i y_i z_i$ is attached, allowing kinematics analysis to be performed. The frame $o_0 x_0 y_0 z_0$, that is attached to manipulator base, is called of

inertial frame. Additionally, the actuation of joint i results in the motion of link i and its attached frame $o_i x_i y_i z_i$. Supposing A_i the homogeneous transformation matrix that defines the position and orientation, or pose, of $o_i x_i y_i z_i$ with respect to $o_0 x_0 y_0 z_0$, the matrix A_i is not constant, it varies according to joints variables.

$$A_i = Ai(q_i) \tag{2.2}$$

The transformation matrix T_j^i expresses the pose of $o_i x_i y_i z_i$ with respect to $o_j x_j y_j z_j$.

$$T_j^i = \begin{cases} A_{i+1} A_{i+2} ... A_{j-1} A_j & \text{if } i < j \\ I & \text{if } i = j \\ (T_i^j)^{-1} & \text{if } j > i \end{cases} \tag{2.3}$$

The pose of the end-effector with respect to the inertial frame is denoted by a combination of a position vector p_n^0 and the 3x3 rotation matrix R_n^0 and it's given by

$$H = T_n^0 = A_1(q_1) ... A_n(q_n) \tag{2.4}$$

where each homogeneous transformation A_i is of the form

$$A_i = \begin{bmatrix} R_i^{i-1} & p_i^{i-1} \\ 0 & 1 \end{bmatrix} \tag{2.5}$$

which results in

$$T_j^i = A_{i+1} A_{i+2} ... A_{j-1} A_j = \begin{bmatrix} R_j^i & p_j^i \\ 0 & 1 \end{bmatrix} \tag{2.6}$$

2.2 FORWARD KINEMATICS

The objective of forward kinematics is to calculate the pose of the end effector as a function of its joints variables (SICILIANO et al., 2010). Forward kinematics can be expressed in the form

$$\xi_E = \kappa(q) \tag{2.7}$$

with ξ_E being the pose of end effector and $\kappa(q)$ a function of joint coordinates. The pose of the end effector has six degrees of freedom - three in translation and three in rotation - and in order to reach arbitraries end effector poses, robot manipulators usually have six DoF (CORKE, 2011). For UR5 manipulators, the forward kinematics equations calculate

a transformation matrix T_0^6, based on q_i values, defined as:

$$\begin{aligned} T_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) &= \begin{bmatrix} R_0^6 & P_6^0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \hat{X_{6x}^0} & \hat{Y_{6x}^0} & \hat{Z_{6x}^0} & P_{6x}^0 \\ \hat{X_{6y}^0} & \hat{Y_{6y}^0} & \hat{Z_{6y}^0} & P_{6y}^0 \\ \hat{X_{6z}^0} & \hat{z_{6z}^0} & \hat{Z_{6z}^0} & P_{6z}^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.8}$$

where P_6^0 is the origin of frame 6 seem from frame 0, and $\hat{X_6^0}$, $\hat{Y_6^0}$, $\hat{Z_6^0}$ are unit vectors defining the direction of three axes of frame 6 in relation to frame 0.

The transformation matrix shown in equation 2.8 can be split as a chain of transformations, one for each of 6 joints of UR5 manipulator:

$$T_6^0 = T_1^0(\theta_1)T_2^1(\theta_2)T_3^2(\theta_3)T_4^3(\theta_4)T_5^4(\theta_5)T_6^5(\theta_6) \tag{2.9}$$

2.2.1 Denavit-Hartenberg Convention

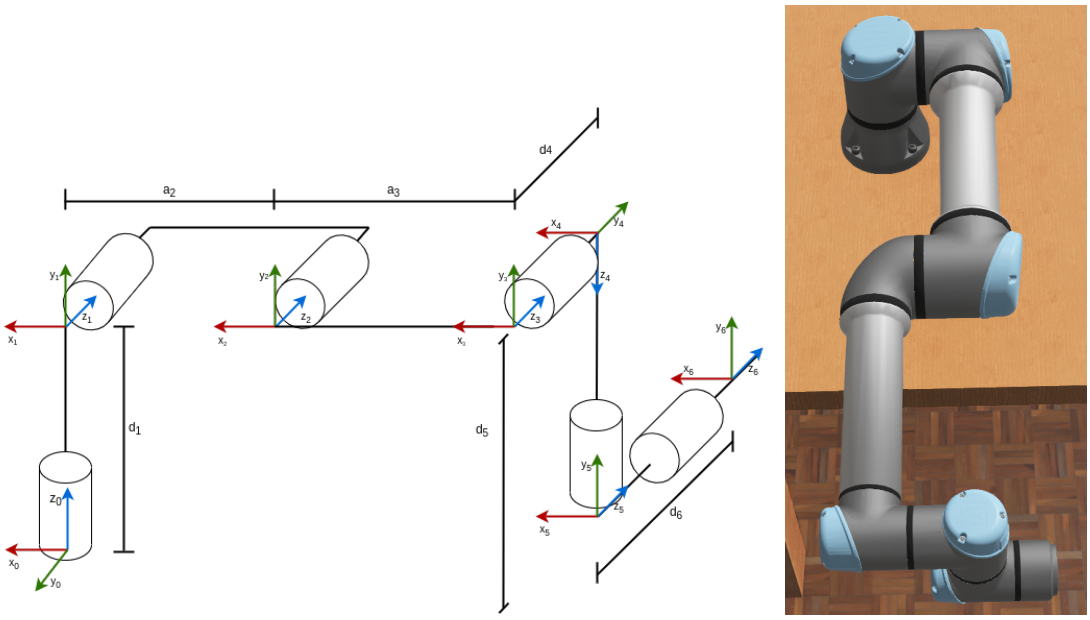
The kinematic analysis of an n-link robot manipulator can become a complex task and the use of conventions simplifies the analysis. A commonly used convention in robotics application is the DH convention, which represents each homogeneous transformation T_i^{i-1} as a product of four basic transformations

$$\begin{aligned} T_i^{i-1} &= Rot_{z,\theta_i}Trans_{z,d_i}Trans_{x,a_i}Rot_{x,\alpha_i} \\ &= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_ic_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_is_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.10}$$

where θ_i, α_i, a_i and d_i are parameters associated with link i and joint i, also known as DH parameters. A common representation of UR5 robot kinematic structure, with all

joint variables (θ_i) at 0, is shown in Figure 2.1.

Figure 2.1: UR5 robot kinematic.



(a) Kinematic chain. (b) Virtual model.

Source: a) (ANDERSEN, 2018) b) Own authorship

According to (SICILIANO et al., 2010; KEATING, 2014; ANDERSEN, 2018) the DH parameters are specified as:

θ_i angle between axes X_{i-1} and X_i, measured about axis Z_{i-1}

α_i angle between axes Z_{i-1} and Z_i, measured about axis X_i

d_i distance from axis X_{i-1} to X_i, measured along axis Z_{i-i}

a_i distance from axis Z_{i-1} to axis Z_i, measured along axis X_i

The DH parameters for UR5 manipulator are shown in Table 2.1. These parameters can be used to write 6 transformations matrixes, one for each link, with their formats following equation 2.10. The complete equation, from axis 6 to base, can be obtained by multiplication of the 6 transformation matrixes, as shown in equation 2.9. After calculations, the matrix T_6^0 that represents the homogeneous transformation from end effector to base is obtained.

Table 2.1: DH parameters for UR5 manipulator.

i	θ	α	d	a
1	θ_1^*	$\pi/2$	0.089159	0
2	θ_2^*	0	0	-0.425
3	θ_3^*	0	0	-0.39225
4	θ_4^*	$\pi/2$	0.10915	0
5	θ_5^*	$-\pi/2$	0.09465	0
6	θ_6^*	0	0.0823	0

* joint variable

2.3 INVERSE KINEMATICS

The Inverse Kinematics propose to solve the inverse problem of forward kinematics: Given a desired pose for end effector, ξ_E, we must calculate the joint variables ($q_i \in [0, 2\pi)$) that the robot needs to reach it. The inverse kinematics can be written in functional form as

$$q = \kappa^{-1}(\xi) \tag{2.11}$$

The function above is not unique, it can have or not have a solution, or it can also have multiple solutions. The inverse kinematics problem is, in general, more complex than the forward kinematics problem, and it can be stated as follows. Given a homogeneous transformation H that represents the desired position and orientation of the end effector

$$H = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \tag{2.12}$$

we must find one or all joint variables values that satisfy the equation

$$T_n^0(q_1, ...q_n) = H \tag{2.13}$$

resulting in 12 nonlinear equations in n unknown variables, with n being the manipulator DoF, which can be written as

$$T_{ij}(q_1, ..., q_n) = h_{ij}, \qquad i = 1, 2, 3, \quad j = 1, 2, 3, 4 \tag{2.14}$$

with T_{ij}, h_{ij} referring, respectively, to the entries of T_n^o and H. These equations are much

difficult to solve in the closed form, so techniques that exploit the kinematic structure of the manipulator are usually used to solve the inverse kinematics problem. The following derivation of UR5 inverse kinematics is adopted from (ANDERSEN, 2018), which used the geometrical approach to determine joint variables based on a desired pose. Another approaches for UR5 inverse kinematics have also been developed by (KEATING, 2014; KEBRIA et al., 2016). The equations below show the six joint variables for UR5 based on a desired end effector pose, with P_j^i expressing the location of frame j in relation to frame i. In order to obtain each P_j^i, methods based on geometrical characteristics of the robot are applied and can be seen with more details in (ANDERSEN, 2018).

$$\theta_1 = atan2(P_{5y}^0, P_{5x}^0) \pm acos(\frac{d_4}{\sqrt{(P_{5x}^0)^2 + (P_{5y}^0)^2}}) + \frac{\pi}{2} \tag{2.15}$$

$$\theta_2 = atan2(-P_{4z}^1, -P_{4x}^1) - asin(\frac{-a_3 sin\theta_3}{\mid P_{4xz}^1 \mid}) \tag{2.16}$$

$$\theta_3 = \pm acos(\frac{\mid P_{4xz}^1 \mid^2 - a_2^2 - a_3^2}{2a_2a_3}) \tag{2.17}$$

$$\theta_4 = atan2(X_{4y}^3, X_{4x}^3) \tag{2.18}$$

$$\theta_5 = \pm acos(\frac{P_{6x}^0 sin\theta_1 - P_{6y}^0 cos\theta_1 - d_4}{d_6}) \tag{2.19}$$

$$\theta_6 = atan2(\frac{-\hat{X}_{0y}^6 sin\theta_1 + \hat{Y}_{0y}^6 cos\theta_1}{sin\theta_5}, \frac{\hat{X}_{0y}^6 sin\theta_1 - \hat{Y}_{0y}^6 cos\theta_1}{sin\theta_5}) \tag{2.20}$$

We can see that the joint variables θ_1, θ_3 and θ_5, equations 2.15, 2.17, 2.17 respectively, have more than one possible solution. For θ_1, the two solutions correspond to the shoulder joint being "left" or "right". For θ_3, the solutions correspond to "elbow up" and "elbow down". Lastly, the two possible solutions for θ_5 correspond to the wrist joint being "up" or "down".

2.4 JACOBIAN MATRIX

The forward and inverse kinematics equations seen in 2.2 and 2.3 establish relationships between the pose of the end effector and joint variables. The differential kinematics, on its turn, relates linear and angular velocities of the end effector with the velocities of the joints. This mapping is obtained through the Jacobian $\boldsymbol{J}$ matrix, one of the

most important tools for manipulator study. The Jacobian matrix is essential for the implementation of kinematic control and planning and execution of smooth trajectories.

In order to obtain a relationship between the velocities of the end effector and the velocities of joints, it is necessary to determine the contribution of the velocity of each joint $\dot{\boldsymbol{q}}$ to the angular velocity $\boldsymbol{w}$ and linear velocity $\boldsymbol{v}$ of the end effector. Being $\boldsymbol{z_i}$ the unit vector that defines the direction of each joint, the contribution of a revolution joint i is given by:

$$\begin{aligned} w_i &= \dot{q}_i z_i, \\ v_i = w_i \times (p_e^i) &= \dot{q}_i z_i \times (p_e^i) \end{aligned} \tag{2.21}$$

This way, the velocity of the end effector $\boldsymbol{V_e}$ can be defined as:

$$V_e = \begin{bmatrix} v_e \\ w_e \end{bmatrix} = \begin{bmatrix} z_1 \times p_e^1 & z_2 \times p_e^2 & ... & z_n \times p_e^n \\ z_1 & z_2 & ... & z_n \end{bmatrix} \dot{q} \tag{2.22}$$

$$V_e = \begin{bmatrix} J_p(q) \\ J_o(q) \end{bmatrix} \dot{q} \tag{2.23}$$

In 2.23 the therm J_p is the matrix that represents the contribution of the joint velocities to the linear velocity v_e. In the other hand, the matrix J_o represents the contribution of the joint velocities to the angular velocity w_e. The $6 \times n$ matrix in 2.24 is the manipulator jacobian:

$$J = \begin{bmatrix} J_p \\ J_o \end{bmatrix} \tag{2.24}$$

which represents the mapping between the joint velocity vector $\dot{\boldsymbol{q}}$ and the end effector velocity vector $\boldsymbol{V_e}$.

Chapter

3

MOTION PLANNING AND OPTIMIZATION SYSTEM

In this chapter, the components of the motion planning and optimization system used in this work are evaluated. Optimization algorithms, CHOMP and STOMP will be discussed, as soon as the concepts of ROS framework and Moveit. The computer vision system used in this work and the obstacle collision detection pipeline will be exhibited. Finally, a flowchart summarizing the execution of the proposed system is presented.

3.1 MOTION PLANNING

The goal of trajectory optimization in motion planning is to find a feasible trajectory that minimizes a cost function and respects a set of defined constraints. This constraints are usually related to obstacles, smoothness and joints torque. In the sequence, the two algorithms for trajectory optimization used in this work will be explained and their characteristics will be discussed, CHOMP and STOMP.

3.1.1 CHOMP

Covariant Hamiltonian Optimization for Motion Planning (CHOMP) (ZUCKER et al., 2013) is a method for trajectory optimization based on covariant gradient technique which produces both *smooth* and *collision-free* trajectories between two specified points x_{init}, x_{goal} for complex robotic systems with many DoF. It uses a similar approach to the elastic bands, where the trajectory is repelled from obstacles by forces, however, unlike previous techniques, CHOMP dispenses the requirement that the initial trajectory be collision-free. The algorithm minimizes the cost function, which consists of obstacle,

velocity and acceleration costs, by means of iteratively updating the trajectory. As it is based on gradient techniques, the cost function must be differentiable.

Given a trajectory $\psi : [0, 1] \rightarrow \mathcal{C}$ as a function mapping time to robot configuration, the algorithm minimizes an objective functional $\mathcal{U} : \Psi \rightarrow \mathbb{R}$ which maps each trajectory ψ in the space of trajectories Ψ to a real number. The objective functional is defined as follows

$$\mathcal{U}(\psi) = \mathcal{F}_{obs}(\psi) + \lambda \mathcal{F}_{smooth}(\psi) \tag{3.1}$$

where the term $\mathcal{F}_{smooth}$ penalizes a trajectory based on joint velocities and accelerations to encourage smoothness. At the same time, the term $\mathcal{F}_{obs}$ penalizes proximity to objects in the environment to encourage collision-free trajectories and λ is a defined trade-off parameter. CHOMP defines the objective functional exclusively in terms of physical aspects of the trajectory, so the calculations depend only on the trajectories physics. To assure computational efficacy, the algorithm discretizes the trajectory ψ into a set of n waypoints equally distributed in time x_1,...,x_n, excluding the end points $x_{init} = x_0$ and $x_{goal} = x_{n+1}$, and computes velocities and accelerations via finite differencing.

The *smooth* term, $\mathcal{F}_{smooth}$, measures dynamical quantities across the trajectory, for example the integral over squared velocity norms:

$$\mathcal{F}_{smooth}(\psi) = \frac{1}{2} \int_0^1 \left\| \frac{d}{dt} \psi(t) \right\|^2 dt \tag{3.2}$$

Using the waypoint parameterization, we can write 3.2 as a sum of finite differences

$$\mathcal{F}_{smooth}(\psi) = \frac{1}{2} \sum_{t=1}^{n+1} \left\| \frac{x_{t+1} - x_t}{\Delta t} \right\|^2 dt \tag{3.3}$$

Being K a finite difference matrix and e a constant vector that condenses the contributions of the fixed end points x_0 and x_{n+1}, we can rewrite 3.3 as

$$\mathcal{F}_{smooth}(\psi) = \frac{1}{2} \|K_d \psi + e\|^2 = \frac{1}{2} A\psi + \psi^T b + c \tag{3.4}$$

with matrix $A = K^T K$, vector $b = K^T e$, and scalar $c = e^T e/2$, the three of them constants. The matrix A is used to measure the acceleration in the trajectory and it will always be symmetric positive definite.

The other term, $\mathcal{F}_{obs}$, measures the proximity to obstacles. Being $c : \mathbb{R}^3 \rightarrow \mathbb{R}$ a scalar potential defined on the workspace that penalizes body elements u of the robot, $\mathcal{B} \subset \mathcal{R}^3$, and $\kappa : \mathcal{Q} \times \mathcal{B} \rightarrow \mathcal{R}^3$ representing the forward kinematics, the obstacle cost function is

defined as:

$$\mathcal{F}_{obs} = \int_0^1 \int_{\mathcal{B}} c(\ \kappa(\psi(t), u)\ ) \left\| \frac{d}{d\tau} \kappa(\psi(t), u) \right\| du\, dt \tag{3.5}$$

By multiplying c by the norm of the velocity of each body point, 3.5 represents the arc-length parametrized integral of the robot's body in the workspace's cost function. For the computation, the set $\mathcal{B}$ is discretized and the integral becomes a summation. The algorithm also maps the central differences of the elements x_t of ψ by the jacobian J to compute the velocity and position of a point.

Then, CHOMP minimizes the cost function $\mathcal{U}$ iteratively updating the initial trajectory ψ_0, which is, in general, a straight-line path. As we said before, the algorithm does not require that the initial trajectory be feasible. It minimizes a first order Taylor expansion of $\mathcal{U}$ around ψ_i:

$$\psi_{i+1} = \arg\min_{\psi} \mathcal{U}\left[\psi_i\right] + \left(\psi - \psi_i\right)^T \bar{\nabla}\mathcal{U}\left[\psi_i\right] + \frac{\eta}{2} \left\|\psi - \psi_i\right\|_M^2 \tag{3.6}$$

where $\bar{\nabla}\mathcal{U}$ represents the computed functional gradient. The CHOMP's update rule is obtained by taking the gradient of the equation 3.6 and equalizing it to zero, which gives:

$$\psi_{i+1} = \psi_i - \frac{1}{\eta} A^{-1} \bar{\nabla}\mathcal{U}\left[\psi_i\right] \tag{3.7}$$

CHOMP uses a signed distance field as an environment representation, which allows obtaining gradients even for non-collision-free points of the trajectory (PAVLICHENKO; BEHNKE, 2017). However, since it uses gradient descent based techniques for optimization, CHOMP's cost function must be differentiable.

3.1.2 STOMP

STOMP (KALAKRISHNAN et al., 2011) is another algorithm that treats the motion planning problem as an optimization problem. It is based on CHOMP and uses the same environment representation that the previous one. But, in contrast, as it uses a stochastic approach for cost minimization, STOMP does not require that the cost function be differentiable, this way reducing the risk of local minima. The objective is to find a smooth and collision-free trajectory which minimizes a predefined cost function that contains costs related to obstacles and constraints of the robot.

The algorithm starts with a trajectory not necessarily feasible, i.e it can be in collision with the environment, of fixed duration and discretized in n points equally spaced in

time. It takes as input the start and the goal pose of the end effector (x_{init}, x_{goal}), both kept fixed during optimization and outputs a path vector $\psi \in \mathbb{R}^N$ for each manipulator joint. In order to simplify its demonstration, the algorithm will be presented considering trajectories of only one dimension, which naturally extends for multiple dimensions.

The trajectory cost function $\mathcal{M}(\psi)$ in STOMP is defined as the sum of state costs $\mathcal{M}_x(\psi)$ and control costs $\mathcal{M}_u(\psi)$

$$\mathcal{M}(\psi) = \mathcal{M}_x(\psi) + \mathcal{M}_u(\psi) \tag{3.8}$$

The term $\mathcal{M}_x$ includes the state-dependent costs and it can contain costs about obstacles, constraint violations, and other objectives related to the task accomplishment. Being $f(\psi_t)$ an arbitrary state-dependent cost function at time t, the state costs can be defined as:

$$\mathcal{M}_x(\psi) = \sum_{t=1}^{N} f(\psi_t) \tag{3.9}$$

The other term, $\mathcal{M}_u(\psi)$, is quadratic in parameters ψ and, being B a positive semi-definite matrix that represents the control costs, it is defined as:

$$\mathcal{M}_u(\psi) = \frac{1}{2}\psi^T B \psi \tag{3.10}$$

The matrix B is chosen so $\mathcal{M}_u(\psi)$ represents the sum of squared accelerations along the trajectory, by using a finite differencing matrix A that produces acceleration ($\ddot{\psi}$) when multiplied by the position vector ψ. The finite differencing matrix A is defined as:

$$A = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

$$\ddot{\psi} = A\psi \tag{3.12}$$

$$\ddot{\psi}^T \ddot{\psi} = \psi^T (A^T A) \psi \tag{3.13}$$

so the definition of $B = A^T A$ ensures that $\mathcal{M}_u(\psi)$ represents the sum of squared accelerations over the trajectory. STOMP optimizes the objective function using a derivative-free

stochastic optimization method, this way enabling arbitrary costs to be optimized, even those that are non-differentiable or non-smooth. Being $\tilde{\psi}$ a noisy parameter vector with mean ψ and covariance Σ, the algorithm attempts to solve the following optimization problem:

$$\min_{\psi} \mathbb{E}\left[\mathcal{M}(\psi)\right] = \min_{\psi} \mathbb{E}\left[\sum_{t=1}^{N} f(\tilde{\psi}_t) + \frac{1}{2}\tilde{\psi}^T B \tilde{\psi}\right] \tag{3.14}$$

Taking the gradient of the expectation in 3.14:

$$\nabla_{\tilde{\psi}} \left(\mathbb{E}\left[\sum_{t=1}^{N} f(\tilde{\psi}_t) + \frac{1}{2}\tilde{\psi}^T B \tilde{\psi}\right]\right) = 0 \tag{3.15}$$

equation 3.15 leads to:

$$\mathbb{E}\left(\psi\right) = -B^{-1}\nabla_{\tilde{\psi}} \left(\mathbb{E}\left[\sum_{t=1}^{N} f(\tilde{\psi}_t)\right]\right) \tag{3.16}$$

which results in:

$$\mathbb{E}\left(\psi\right) = -B^{-1}\mathbb{E} \left(\nabla_{\tilde{\psi}} \left[\sum_{t=1}^{N} f(\tilde{\psi}_t)\right]\right) \tag{3.17}$$

The equation 3.17 can be written as $\mathbb{E}\left(\psi\right) = -B^{-1}\delta\hat{\psi}_G$, where $\delta\hat{\psi}_G$ is now the estimated gradient as it follows below:

$$\delta\hat{\psi}_G = \mathbb{E} \left(\nabla_{\tilde{\psi}} \left[\sum_{t=1}^{N} f(\tilde{\psi}_t)\right]\right) \tag{3.18}$$

Based on probability matching and path integral reinforcement learning, STOMP estimates the gradient as the expectation of the noise ϵ in the vector ψ under the probability metric $P \propto \exp\left(-\frac{1}{\lambda}\mathcal{M}(\tilde{\psi})\right)$. So, the stochastic gradient can be formulated as:

$$\delta\hat{\psi}_G = \int \epsilon \, dP = \int \exp\left(-\frac{1}{\lambda}\mathcal{M}\left(\psi + \epsilon\right)\right) \epsilon \, d\epsilon \tag{3.19}$$

The equation 3.19 is estimated, in practice, by sampling a finite number of trajectories:

$$\delta\hat{\psi}_G = \sum_{k=1}^{K} P(\psi + \epsilon_k)\epsilon \tag{3.20}$$

$$P(\psi + \epsilon_k) = \frac{\exp\left(-\frac{1}{\lambda}\mathcal{M}(\psi + \epsilon_k)\right)}{\sum_{l=1}^{K} \exp\left(-\frac{1}{\lambda}\mathcal{M}(\psi + \epsilon_l)\right)} \tag{3.21}$$

At every iteration of the algorithm, the gradient update shown in equation 3.20 is applied to the original trajectory with sample trajectories being re-generated from the newly updated trajectory. In equation 3.21, the probabilities of each noisy parameter are computed per time-step. The parameter λ regulates the sensitivity of the exponential cost, and this term can be calculated as:

$$e^{\left(-\frac{1}{\lambda}\mathcal{M}(\psi_{k,t})\right)} = e^{-h\frac{\mathcal{M}(\psi_{k,t})-\min\mathcal{M}(\psi_{k,t})}{\max\mathcal{M}(\psi_{k,t})-\min\mathcal{M}(\psi_{k,t})}}, \tag{3.22}$$

with h set to be a constant value. Additionally, by considering that the state cost $f(\psi_i)$ is purely dependent on the parameter ψ_i and not taking future or past costs to the current state, STOMP accelerates the convergence of the algorithm. The STOMP algorithm can be seen in Algorithm 1:

Algorithm 1 STOMP

• **Given:**

- Start pose x_s
- Goal pose x_g
- Initial trajectory ψ discretized in N points
- State-dependent cost function $f(\psi_i)$
- Control cost matrix B
- Standard deviation of exploration noise σ

• **Repeat until convergence of** $\mathcal{M}(\psi)$**:**

1. Create K noise trajectories $\overline{\psi}_1, \overline{\psi}_2 ... \overline{\psi}_k$ with parameters $\psi + \epsilon_k$, where ϵ_k is a zero mean normal distribution ($\epsilon_k = \mathcal{N}(0, \sigma^2 B^{-1})$
2. For $k = 1...K$ and $t = 1...N$, compute:

   a) $\mathcal{M}(\tilde{\psi}_{k,t}) = f(\tilde{\psi}_{k,t}) + \dfrac{1}{2N}\psi^T B\psi$

   b) $P(\tilde{\psi}_{k,t}) = \dfrac{e^{\left(-\frac{1}{\lambda}\mathcal{M}(\tilde{\psi}_{k,t})\right)}}{\sum_{l=1}^{K}\left[e^{\left(-\frac{1}{\lambda}\mathcal{M}(\tilde{\psi}_{k,t})\right)}\right]}$

3. For $t = 1...(N-1)$, compute: $\left[\delta\tilde{\psi}\right]_t = \sum_{k=1}^{K} P(\tilde{\psi}_{k,t})\left[\epsilon_k\right]_t$
4. Update vector $\psi \leftarrow \psi + B^{-1}\delta\tilde{\psi}$
5. Calculate trajectory cost $\mathcal{M}(\psi) = \sum_{t=1}^{N} f(\psi_t) + \frac{1}{2}\psi^T B\psi$

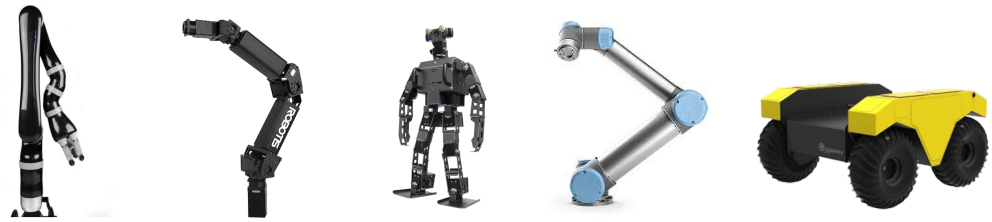
3.2 ROS FRAMEWORK

Robot Operating System (ROS) is an open-source framework that works between multiple platforms and provides a series of tools and databases for robotics development. It has high compatibility, being used with a wide number of robots, and we can say ROS is one of the most popular robot development platform nowadays (XU; DUGULEANA, 2019). The official description of ROS is:

> *ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers (ROS.ORG, 2018).*

The main goal of ROS is to make the components of a robotic system more easy to develop and share, so they can be used on other robots with minimal changes, allowing code reuse and improving code's quality (MAHTANI et al., 2016). ROS provides essential functions for robots programming, such as communication among heterogeneous hardware and error treatment and it has been forming an ecosystem that distributes packages made by users (PYO et al., 2017). A lot of research institutions and companies have been developing projects in ROS by adding hardware drivers and sharing code samples, some examples of compatible ROS robots can be seen in Figure 3.1. In this work, we use ROS packages to calculate UR5 forward and inverse kinematics, get RBG+D sensor data and integrate it into the system, and control the movements of the arm.

Figure 3.1: Some ROS compatible robots. From left to right: Jaco, by Kinova; Open-Manipulator and OP3, by Robotis; UR5, by Universal Robots and Warthog, by Clearpath.

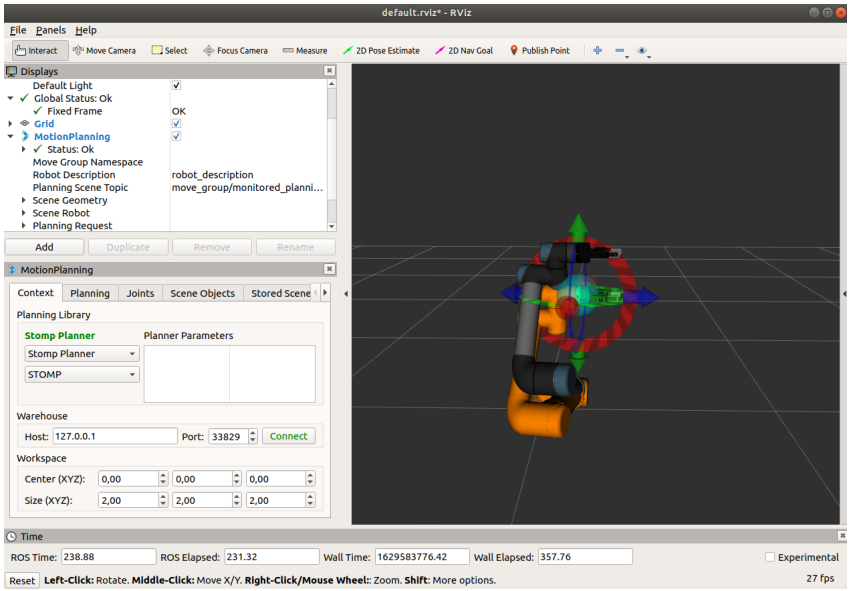


Source: (ROBOTICS, 2022b; ROBOTIS, 2022; ROBOTS, 2022; ROBOTICS, 2022a)

3.3 MOVEIT

Moveit is a library for manipulation that integrates a set of tools for motion planning and control of robot arms. It supports popular solutions for inverse kinematics, such as KDL, IKFast and TRAC-IK (BEESON; AMES, 2015). It also integrates advanced motion planning algorithms, including OMPL (SUCAN; MOLL; KAVRAKI, 2012), CHOMP (ZUCKER et al., 2013) and STOMP (KALAKRISHNAN et al., 2011). Moveit combines state-of-the-art algorithms for motion planning, kinematics, control, perception and navigation, and we can say it is the most advanced tool for robotic manipulation. It also offers a friendly easy-to-use interface for the development of advanced applications, as we can see in Figure 3.2, and it has been extensively used with a wide range of robots for industry, research, commerce, and other sectors.

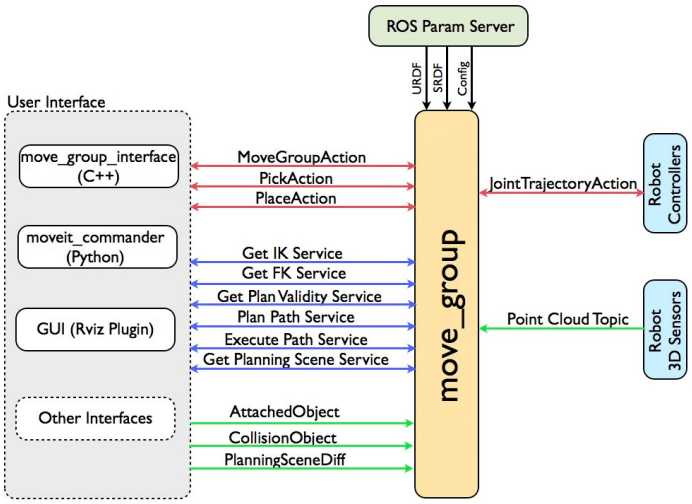
Figure 3.2: UR5 motion planning using Moveit and RViz.



In the center of Moveit architecture, we have the `move_group node`, as shown in Figure 3.3. This node integrates all the individual components to provide ROS actions and services for users (MOVEIT, 2018). From ROS Param Server, it collects the robot kinematics data, such as Unified Robot Description Format (URDF), Semantic Robot Description Format (SDRF) and configuration files. The SDRF and the configuration files are both generated when we create the Moveit package for our robot, and they contain the parameters of the manipulator, such as joint limits, kinematics and end effector. The move_group node also provides the state and control of the robot through ROS topics and actions, for example the `/joint_states` topic and the `JointTrajectoryAction` interface. The `move_group` also provides a interface to motion planners which can use different

libraries, so it can generate trajectories for desired locations of the end effector respecting constraints such as position, orientation or joint constraints.

Figure 3.3: The move_group node.



Source: (MOVEIT, 2022)

In this work, Moveit is used to integrate TRAC-IK, for inverse kinematics, with STOMP and CHOMP, for motion planning, allowing it to control a UR5 collaborative manipulator and the Robotiq gripper attached to the manipulator end effector in a pick and place application at an environment with obstacles. The Point Cloud provided by the Intel Realsense D435 is integrated into the framework to provide obstacle and collision detection. In addition, the framework provides visualization of the noisy trajectories generated by STOMP, as well as the final trajectory optimized by both algorithms.

3.4 COMPUTER VISION

Computer vision is a common topic in robotic researches nowadays, since vision sensors have becoming more accessible than other types of perception sensors, while the computers are getting smaller and more powerful, enabling the use of complex vision algorithms. In this work, we use Intel Realsense D435 to provide data utilized for obstacles detection. Some technical specifications of the used sensor are listed below (INTEL, 2022; MEJIA-TRUJILLO et al., 2019):

- Technology: Active stereoscopy
- Depth Field of View (FoV) (Horizontal x Vertical): 85.2°x 52°(+/- 3°)

- Depth resolution: 1280 x 720
- Depth frame rate: 90 fps
- Minimum depth distance: 0.11 m
- Maximum Range: close to 10 meters
- RGB resolution: 1920 x 1080
- RGB frame rate: 30 fps
- RGB FoV (Horizontal × Vertical): 69.4°x 42.5°(+/- 3°)

The depth camera is mounted on the robot's end effector, in an eye-on-hand configuration, which means that the camera is attached to the wrist frame of the robot. The captured data by the depth sensor was then used to provide information about objects on the scene and avoid collisions. Based on the perception tool of Moveit, a schematic of the obstacle detection pipeline is shown in Figure 3.4 as it contains the following stages:

1. The initial state of the robot is stored.
2. The image of the camera obtained.
3. As we use an RGB+D sensor, the point cloud data is obtained.
4. Based on the robot state and point cloud data, a Planning Scene is generated with geometrical representation of the objects in the environment.
5. If some link of the robot is assumed to be in contact with any object, the mesh in contact is represented in the color red, and the movement is not allowed to be executed. Examples of collision detection with objects (5a) and printer (5b) are shown.

3.5 FLOWCHART OF THE SYSTEM

A flowchart with a high-level overview of the system described in this chapter is presented in Figure 3.5.

Figure 3.4: Obstacle collision detection pipeline.

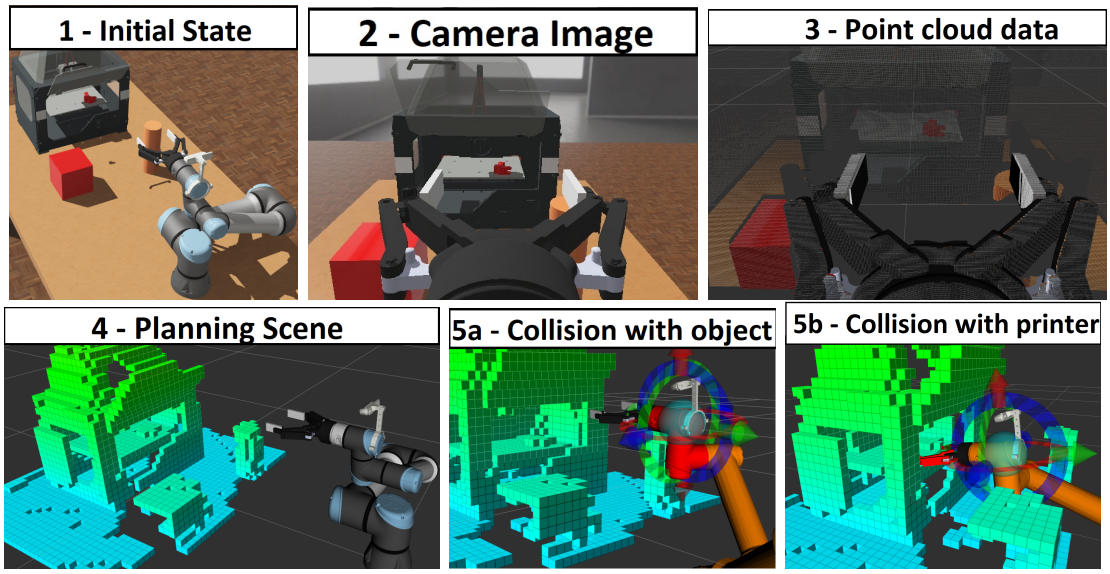
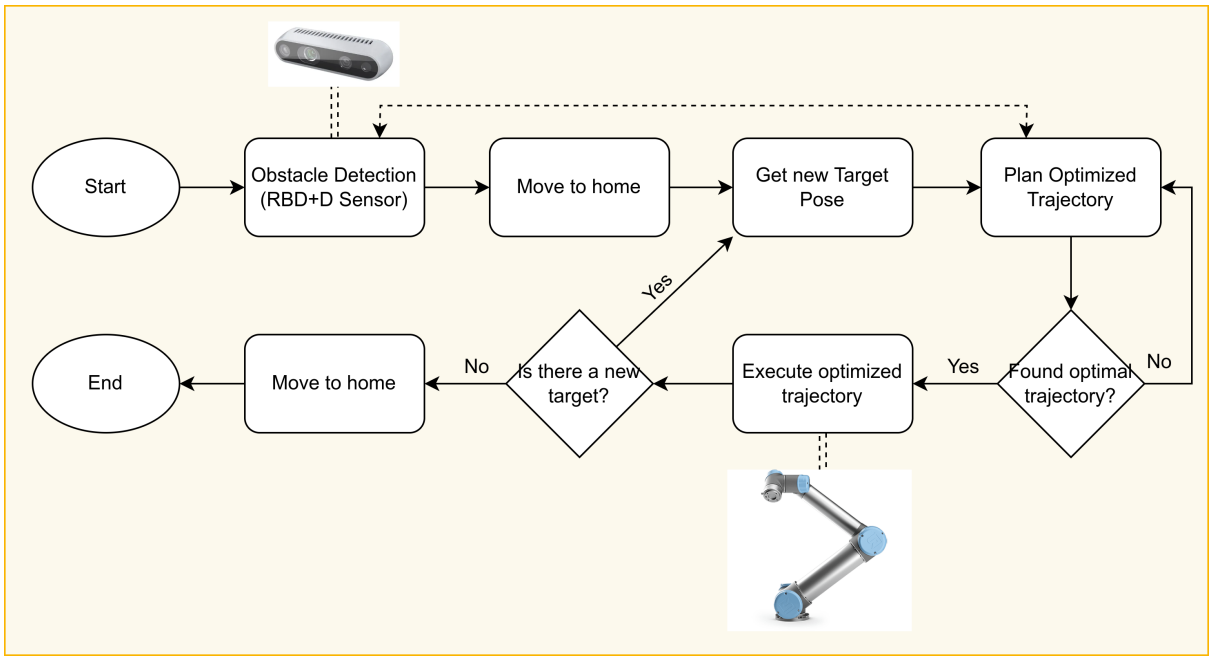


Figure 3.5: Flowchart of the developed system.



Chapter

4

EXPERIMENTAL RESULTS

4.1 THE SETUP

All the experiments were conducted in an additive manufacturing unit composed by an UR5 robotic manipulator, from Universal Robots, controlled by ROS/Moveit and a 3D printer. Units like these can be used in a wide range of applications evolving cobots for the industry 4.0, such as pick and place of manufactured components in dynamic environments. In this work, the experiments were performed with the objective of a pick and place task, avoiding collision with obstacles in the scene.

The simulation was developed on Webots, reproducing the characteristics of Robotics Lab, from Federal University of Bahia. In figure 4.1 its shown the simulation environment, with the cobot UR5 placed on a table, in front of a 3D printer, containing the target piece. Besides other manufactured pieces placed on the table, it's also displayed one of the obstacles that should be avoided, a cylinder of 0.25 m height and 0.04 m radius.

The CPU platform used was an Intel® Core™ i7-7050H CPU @ 2.60GHz, 8GB memory, and the GPU platform was NVIDIA® GeForce® GTX 1650 4GB.

From an simulated Intel Realsense D435 camera, coupled to the manipulator's end effector, and from the Perception tool, integrated into moveit, it is possible to detect obstacles present in the environment, in order to plan trajectories that avoid them. Figure 4.2 shows a representation on Rviz of the detected collision objects present on cobot's workspace.

Figure 4.1: Experiment simulation, containing the UR5 manipulator, a 3D printer, manufactured pieces and obstacles.

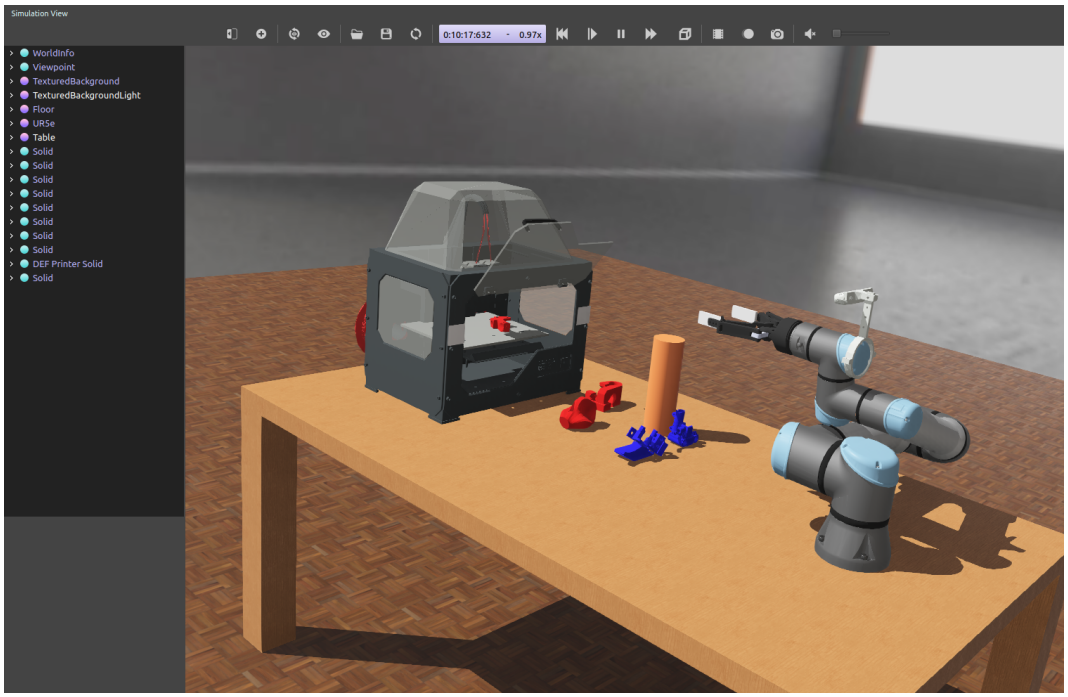
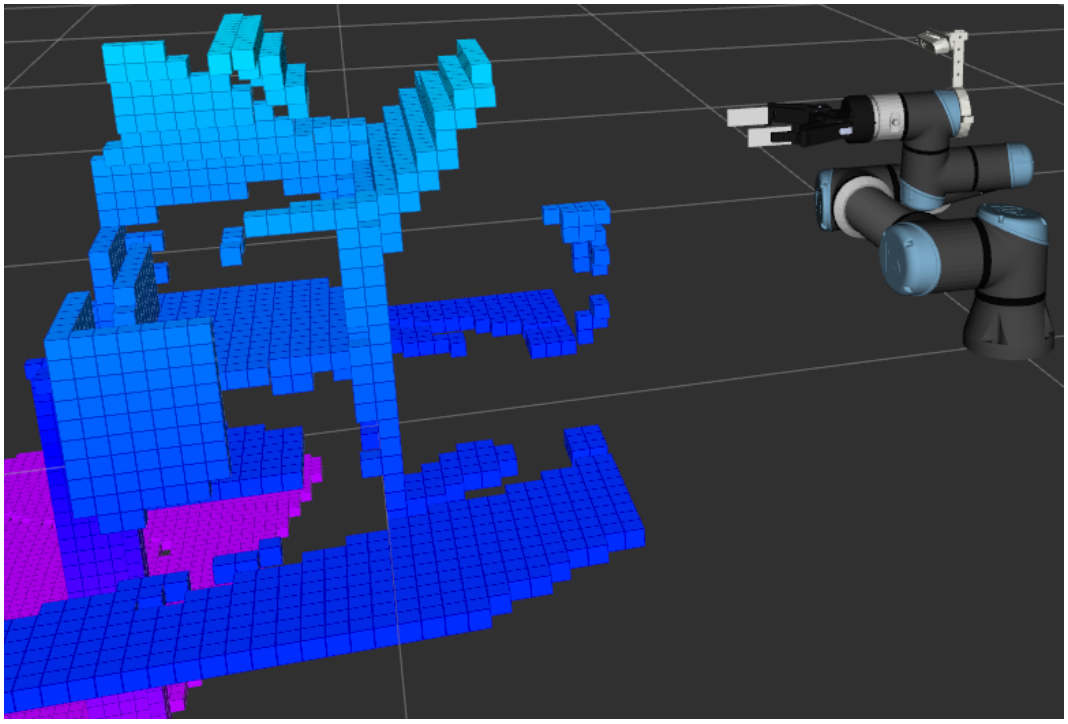


Figure 4.2: Collision objects representation on Rviz.



4.2 THE EXPERIMENTS

As already explained, experiments were performed with the objective of realize a pick and place task, avoiding collision with obstacles of known location in the scene. After the execution, the cobot must return to its initial position. We developed four simulation and three real scenarios for the tests. On these scenarios, the cobot routine was executed using CHOMP and STOMP algorithms, one each at a turn. Additionally, for comparison, we have also executed the routine using the algorithm RRT-Connect from OMPL, the default planner on Moveit.

All the code used in this work in available at ⟨https://github.com/migueelnery/ur5_trajectory_optimization⟩ and the videos from the performed experiments can be seen at: ⟨https://youtu.be/ARbzy7xFaq8⟩.

4.2.1 Simulation Scenario I

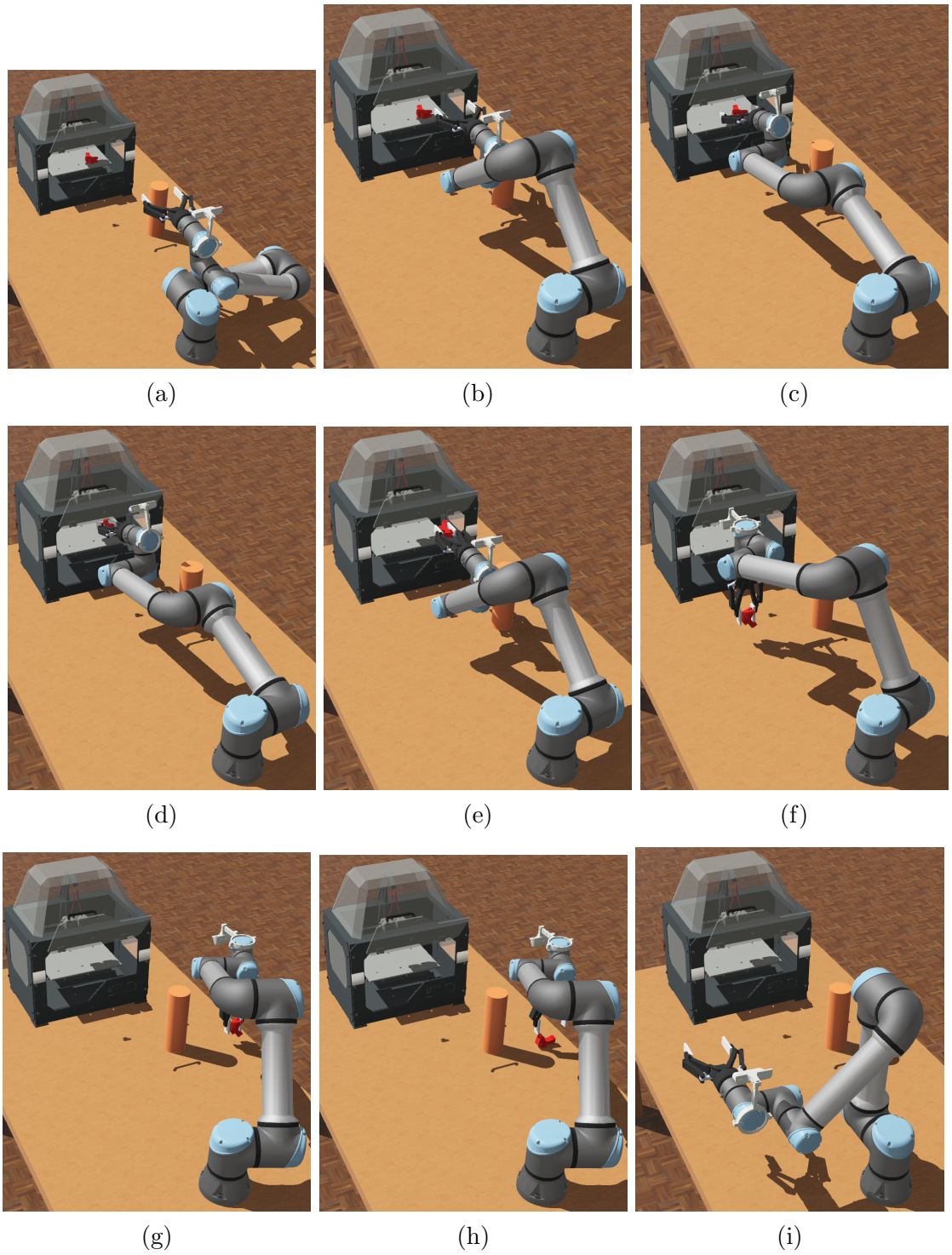
For the first simulation scenario, the cobot must pick a manufactured piece and place it on the table, while avoiding collisions with a cylinder located on the robot's workspace. Figure 4.3 shows the simulated scenario and the desired routine. The routine was executed 15 times for each algorithm, CHOMP, and STOMP and RRT-Connect.

The metrics used to compare the performance of the algorithms were: Success Rate, Planning time(s), and Trajectory Duration(s). From the `plan()` function, implemented on `moveit_commander`, API in python for Moveit, it is possible to obtain as a return a boolean variable indicating success and a float variable containing the planning time. The function also returns a `RobotTrajectory` message, which contains the planned trajectory, indicating the time in which each point of the trajectory must be performed. In this way, from the execution time of the last point of this message, it's possible to infer the trajectory duration.

The success rate for the experiments performed on the first simulated scenario can be seen in Table 4.1. Results indicate that CHOMP, STOMP, and RRT-Connect have good and similar success rates in the routine execution, generating feasible trajectories for the collaborative robot in the presence of obstacles, successfully getting to solve the planning problem in most of the tries.

To analyze the planning time and the duration of the computed path, the trajectory referring to the displacement between frames 4.3f to 4.3g was selected. This trajectory characterizes a displacement from the center to the left of the table, with the obstacle (cylinder) to be avoided between the initial and final points, while the manipulator holds the manufactured part with its end effector. The collected data were analyzed using the

Figure 4.3: Simulation routine I.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

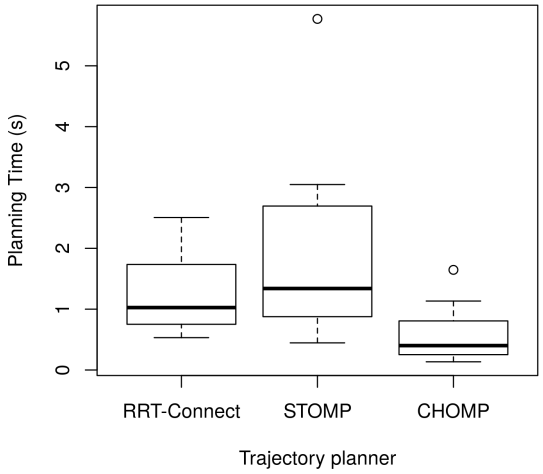
Table 4.1: Success Rate - Simulation Scenario I.

Algorithm	CHOMP	STOMP	RRT-Connect
Success (%)	97.33	98.33	97.33

R statistical programming language, and the boxplot graphs generated for each variable can be seen in the Figures 4.4 and 4.5.

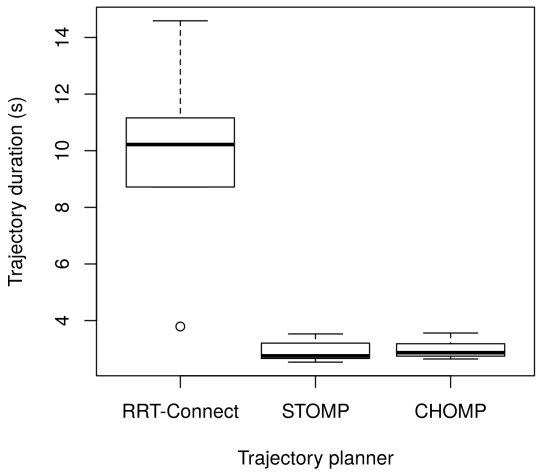
As exhibited in Figure 4.4, the algorithms have similar results for planning time. The algorithm RRT-Connect obtained median value of 1.026 s, upper value of 1.734 s and lower value of 0.751 s; STOMP presented median value of 1.339 s, upper value of 2.695 s and lower value of 0.876 s; lastly, CHOMP presented median value of 0.401 s, upper value of 0.806 s and lower value of 0.252 s. Results indicate that CHOMP obtained slightly better results for planning time, and STOMP presenting more variation in the obtained data, which can be a reflex of its stochastic approach.

Figure 4.4: Planning Time for each trajectory planner on simulated scene I.



Regarding the duration of the trajectory, shown in Figure 4.5, the algorithm RRT-Connect obtained median value of 10.218 s, upper value of 11.159 s and lower value of 8.718 s; STOMP presented median value of 2.755 s, upper value of 3.205 s, and lower value of 2.664 s; CHOMP presented median value of 2.866 s, upper value of 3.183 s and lower value of 2.746 s. We can see that STOMP and CHOMP produces trajectories of shorter duration than RRT-Connect. Figure 4.6 shows an example of a point to point trajectory for each one of the algorithms, with the same start and goal, and it's notable

Figure 4.5: Duration of the generated path for each trajectory planner on simulated scene I.



that the trajectory generated by RRT-Connect has longer duration than the others.

4.2.2 Simulation Scenario II

In the second simulated scenario, we have added a cylinder of 0.25 m height and 0.04 m radius, and a cube of dimensions 0.15 x 0.15 x 0.15 m and on cobot's workspace, as shown in Figure 4.7. In this way, it is expected that the robot executes collision-free trajectories, avoiding the present obstacles.

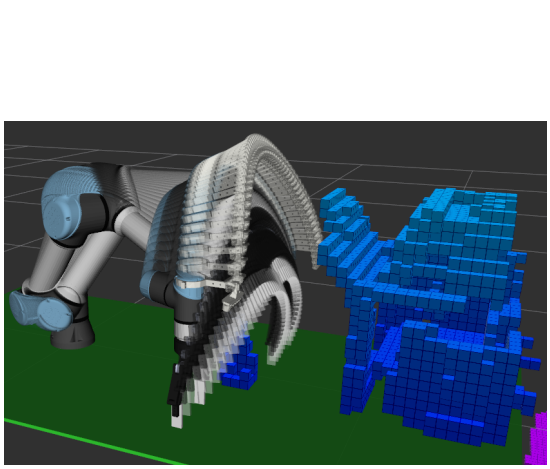
The success rate for the second simulated scenario can be seen in Table 4.2, and the obtained results indicate that CHOMP, STOMP, and RRT-Connect have good and similar values, being able to generate feasible trajectories in mostly cases.

Table 4.2: Success Rate - Simulation Scenario II.

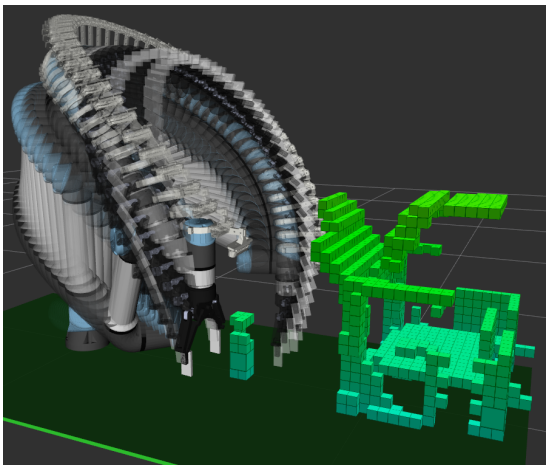
Algorithm	CHOMP	STOMP	RRT-Connect
Success (%)	95.71	97.14	95.71

The trajectory referring to the displacement between frames 4.7e to 4.7f was selected in order to analyze the planning time and the duration of the computed path. This trajectory characterizes a displacement from the 3D printer, where the piece is picked, to the target location while avoiding obstacles in the scene. Results for planning time can be seen in Figure 4.8, the algorithm RRT-Connect obtained median value of 0.309 s,

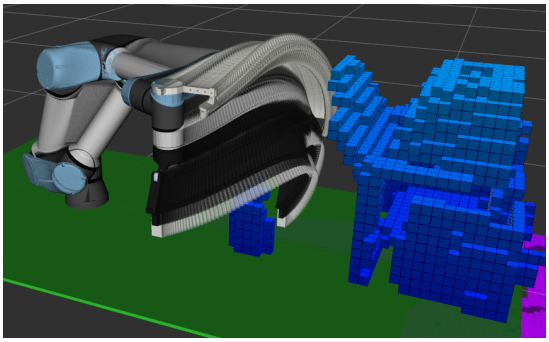
Figure 4.6: Points of the generated path by each algorithm on simulated scenario I.



(a) STOMP



(b) RRT-Connect

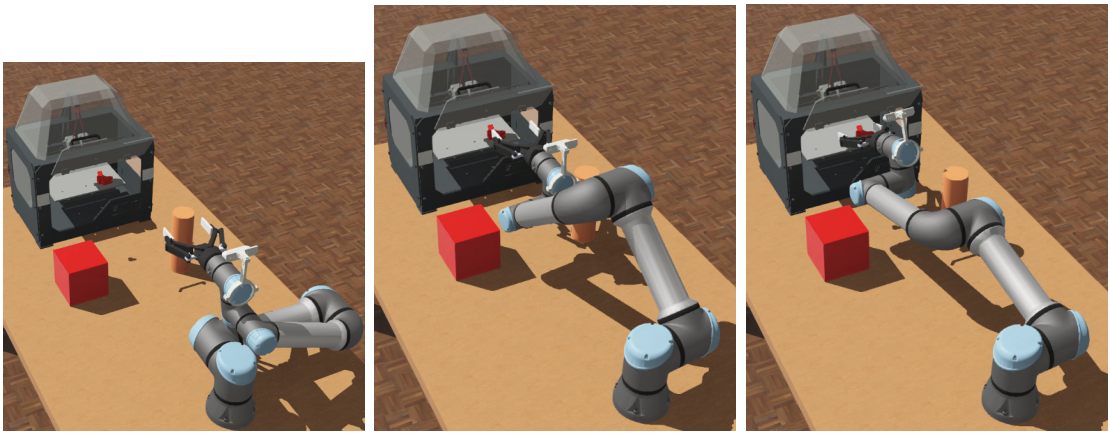


(c) CHOMP

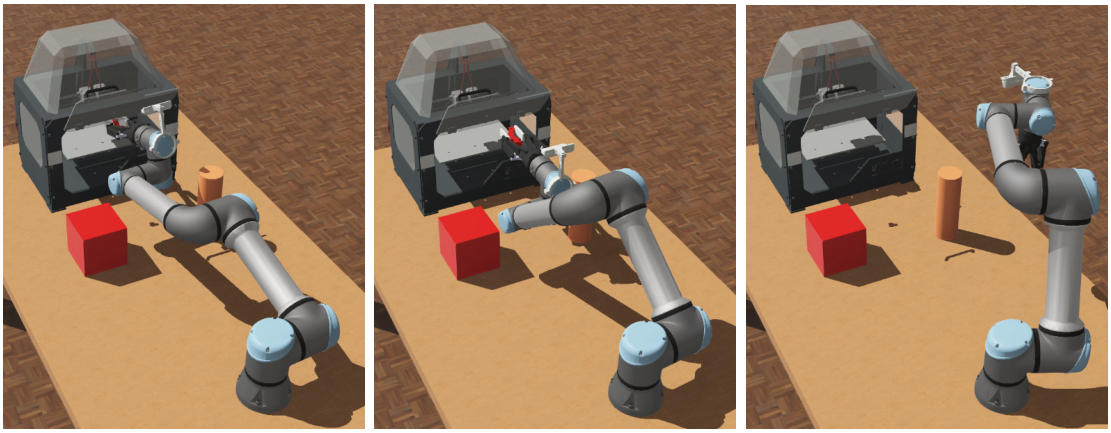
upper value of 0.567 s and lower value of 0.139 s; STOMP presented median value of 1.36 s, upper value of 2.036 s and lower value of 0.983 s; CHOMP presented median value of 0.145 s, upper value of 1.01 s and lower value of 0.087 s.

As we can see in Figure 4.9, for trajectory duration the algorithm RRT-Connect obtained median value of 4.794 s, upper value of 6.038 s and lower value of 3.93 s; STOMP obtained median value of 3.509 s, upper value of 3.997 s and lower value of 3.374 s; CHOMP presented median value of 4.353 s, upper value of 4.847 s, and lower value of 4.119 s. We can notice that, although STOMP needs a longer planning time than CHOMP and RRT-Connect, it generates trajectories of shorter duration than the others, consequently preventing the cobot from performing unnecessary movements during execution.

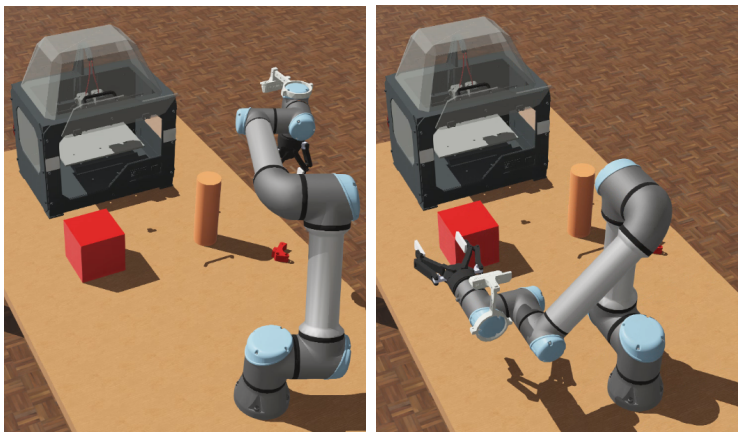
Figure 4.7: Simulation routine II.



(a) (b) (c)



(d) (e) (f)



(g) (h)

Figure 4.8: Planning Time for each trajectory planner on simulated scene II.

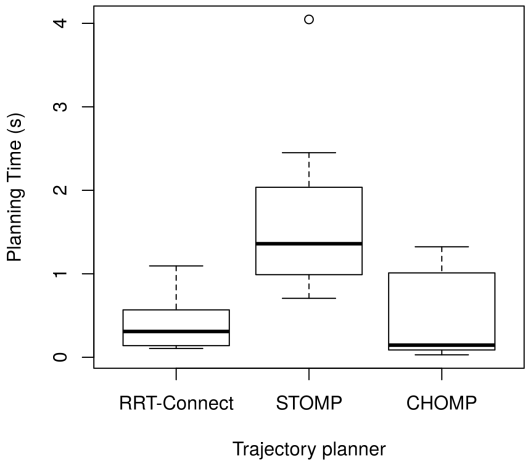
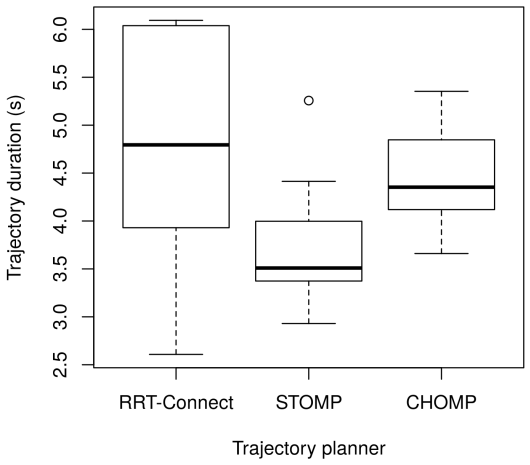


Figure 4.9: Duration of the generated path for each trajectory planner on simulated scene II.

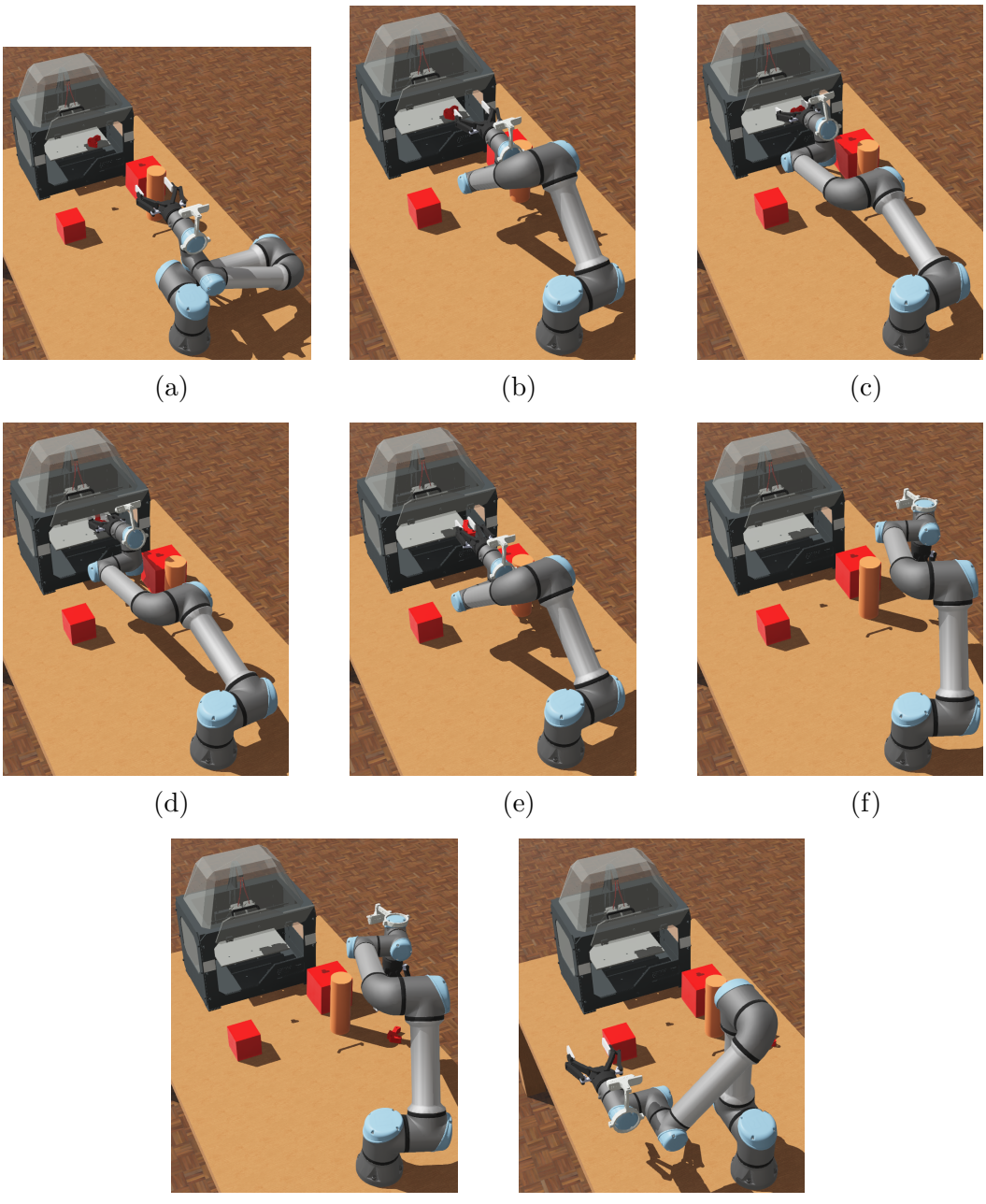


4.2.3 Simulation Scenario III

In the simulation scenario III, we have added a cylinder of 0.25 m height and 0.04 m radius, and two cubes, one of dimensions 0.1 x 0.1 x 0.1 m, and another of 0.15 x 0.15 x 0.15 m, on cobot's workspace, as shown in Figure 4.10.

The success rate for the third simulated scenario can be seen in Table 4.3, and the obtained results are very similar with scenarios I and II, indicating that CHOMP, STOMP, and RRT-Connect are able to generate feasible trajectories in mostly cases.

Figure 4.10: Simulation routine III.



(a) (b) (c)

(d) (e) (f)

(g) (h)

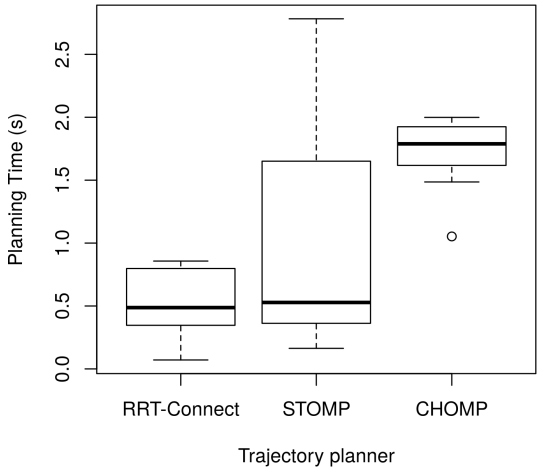
As was done in 4.2.2, the trajectory referring to the displacement between frames 4.10e to 4.10f was selected in order to analyze the planning time and the duration of the computed path, since this trajectory represents a displacement from the 3D printer to the target location while avoiding obstacles in the scene. According to the results

Table 4.3: Success Rate - Simulation Scenario III.

Algorithm	CHOMP	STOMP	RRT-Connect
Success (%)	95.71	97.14	97.14

for planning time exhibited in Figure 4.11, the algorithm RRT-Connect obtained median value of 0.487 s, upper value of 0.798 s and lower value of 0.347 s; STOMP presented median value of 0.528 s, upper value of 1.651 s and lower value of 0.362 s; CHOMP presented median value of 1.789 s, upper value of 1.925 s and lower value of 1.617 s.

Figure 4.11: Planning Time for each trajectory planner on simulated scene III.

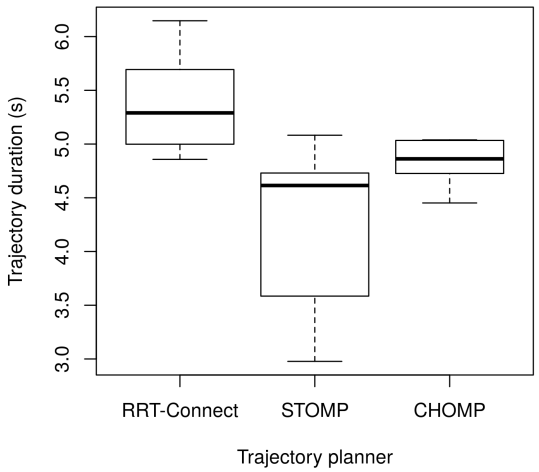


Regarding trajectory duration, as seen in Figure 4.12, the algorithm RRT-Connect obtained median value of 5.290 s, upper value of 5.693 s, and lower value of 4.998 s; STOMP presented median value of 4.614 s, upper value of 4.73 s, and lower value of 3.585 s; CHOMP presented median value of 4.862 s, upper value of 5.033 s, and lower value of 4.726 s. We can notice that the behavior is similar to what has already been shown in the previous situations, with STOMP and CHOMP needing a longer planning time than RRT-Connect, although generating trajectories of shorter duration.

4.2.4 Simulation Scenario IV

The Simulation Scenario IV can be seen in Figure 4.13. This scenario represents a more difficult workspace, since it contains more obstacles: a cylinder of 0.25 m height and 0.04 m radius, a cube of dimensions 0.1 x 0.1 x 0.1 m, and a parallelepiped of dimensions 0.15

Figure 4.12: Duration of the generated path for each trajectory planner on simulated scene III.



x 0.15 x 0.30 m, which the cobot must avoid while executing the routine. The success rate for the fourth simulated scenario can be seen in Table 4.4. The obtained results are very similar with scenarios seen before, indicating that CHOMP, STOMP, and RRT-Connect are able to generate feasible trajectories in the presence of obstacles.

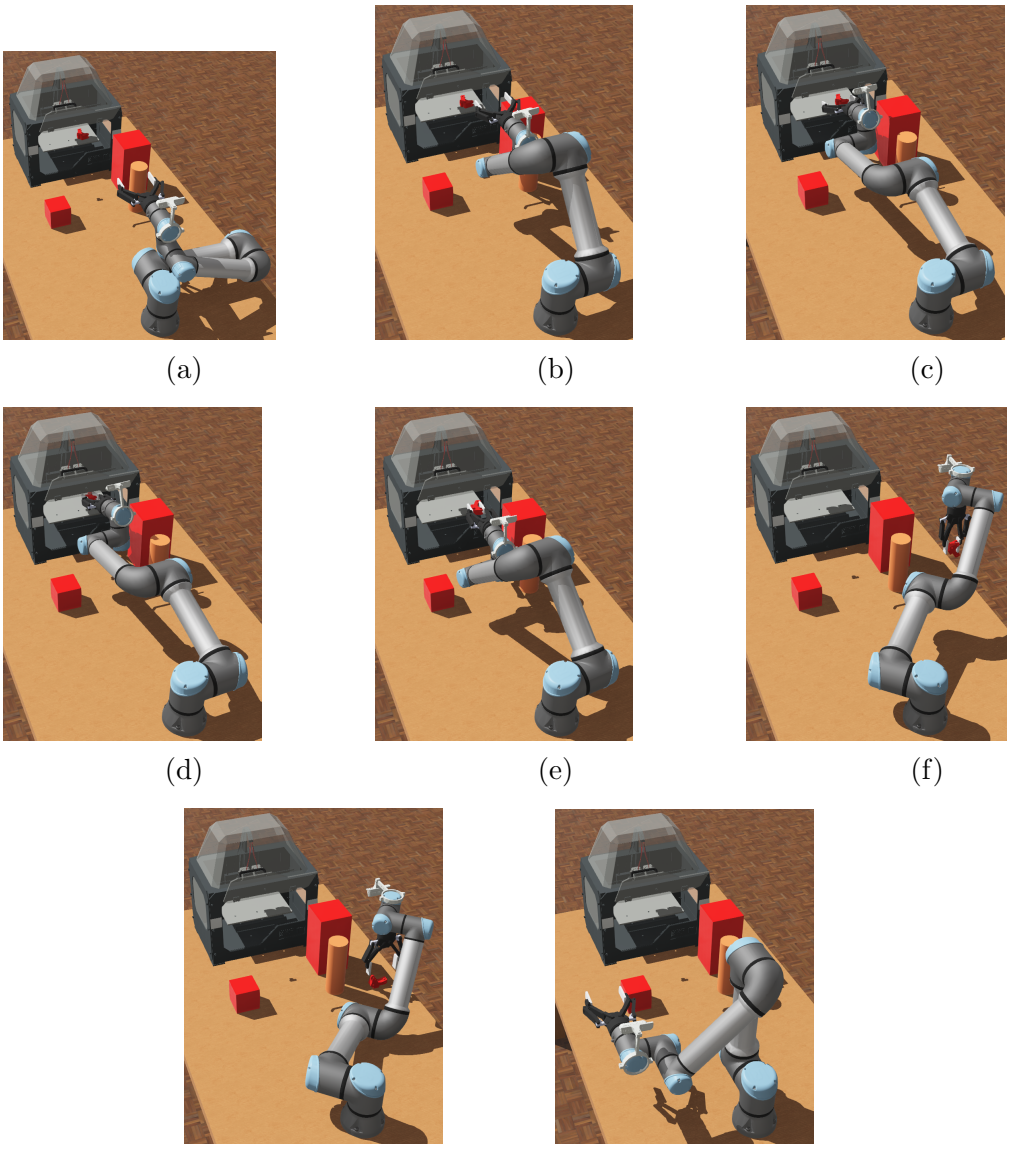
Table 4.4: Success Rate - Simulation Scenario IV.

Algorithm	CHOMP	STOMP	RRT-Connect
Success (%)	95.71	95.71	95.71

As was done in 4.2.2 and 4.2.3, the trajectory referring to the displacement between frames 4.13e to 4.13f was selected in order to analyze the planning time and the duration of the trajectory, since it represents a displacement from the 3D printer to the target location while avoiding obstacles. Results for planning time can be seen in Figure 4.14, the algorithm RRT-Connect obtained median value of 0.312 s, upper value of 0.49 s, and lower value of 0.256 s; STOMP obtained median value of 1.218 s, upper value of 1.717 s, and lower value of 0.373 s; lastly, CHOMP presented median value of 1.02 s, upper value of 1.368 s, and lower value of 0.321 s.

Figure 4.15 shows the results for trajectory duration, RRT-Connect presented median value of 5.776 s, upper value of 6.165 s, and lower value of 5.093 s; STOMP presented median value of 4.717 s, upper value of 4.893 s, and lower value of 4.255 s; CHOMP presented median value of 4.426 s, upper value of 4.783 s, and lower value of 4.361 s. We

Figure 4.13: Simulation routine IV.



(a) (b) (c) (d) (e) (f) (g) (h)

can notice that in all simulated scenarios the algorithm RRT-Connect generated trajectories of longer duration than the others, consequently increasing the risk of executing trajectories that could cause some type of damage to the manipulator.

4.2.5 Simulation Results Summary

The results obtained in 4.2.1, 4.2.2, 4.2.3 and 4.2.4 have been condensed and are shown in the following. The values exhibited in Table 4.5 indicates that the three algorithms have

Figure 4.14: Planning Time for each trajectory planner on simulated scene IV.

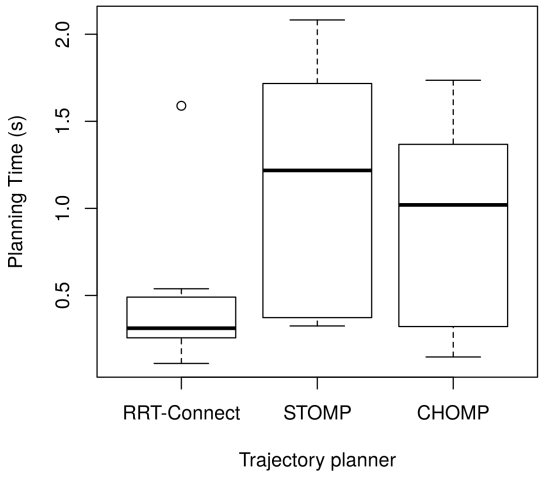
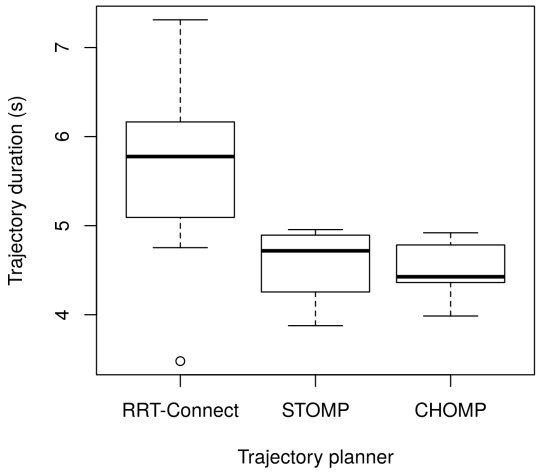


Figure 4.15: Duration of the generated path for each trajectory planner on simulated scene IV.



obtained similar success rates in the routine execution, generating feasible trajectories for the collaborative robot.

As we can see in Figure 4.16, the algorithms also have similar average planning time, with STOMP presenting more variation in the obtained data. When analyzing the duration of the trajectory obtained by each algorithm in Figure 4.17, we can see that STOMP tends to produce trajectories of shorter duration than CHOMP and RRT-Connect. It is also notable that, in all simulated scenarios, RRT-Connect generated trajectories of longer duration than the others, consequently increasing the risk due to the execution of unneeded movements.

Table 4.5: Success rate on simulated scenarios.

	CHOMP	STOMP	RRT-Connect
Success	96.66%	96.66%	93.33%

Figure 4.16: Planning time by scene and algorithm on simulated scenarios.

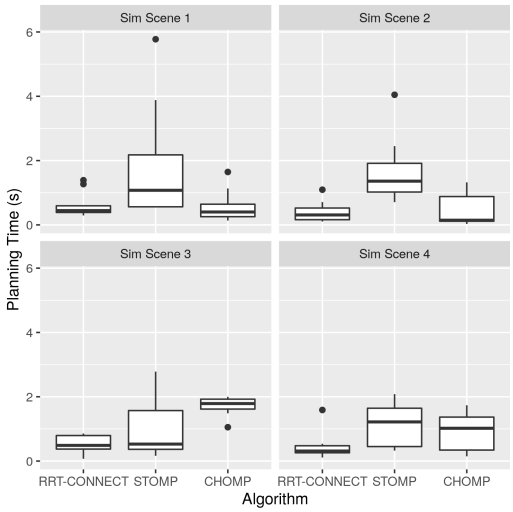
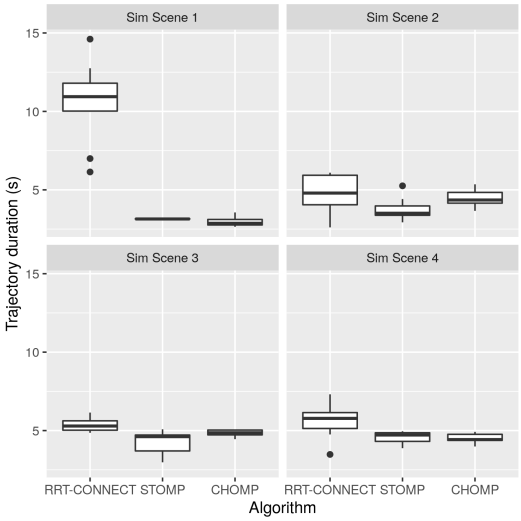


Figure 4.17: Trajectory duration by scene and algorithm on simulated scenarios.

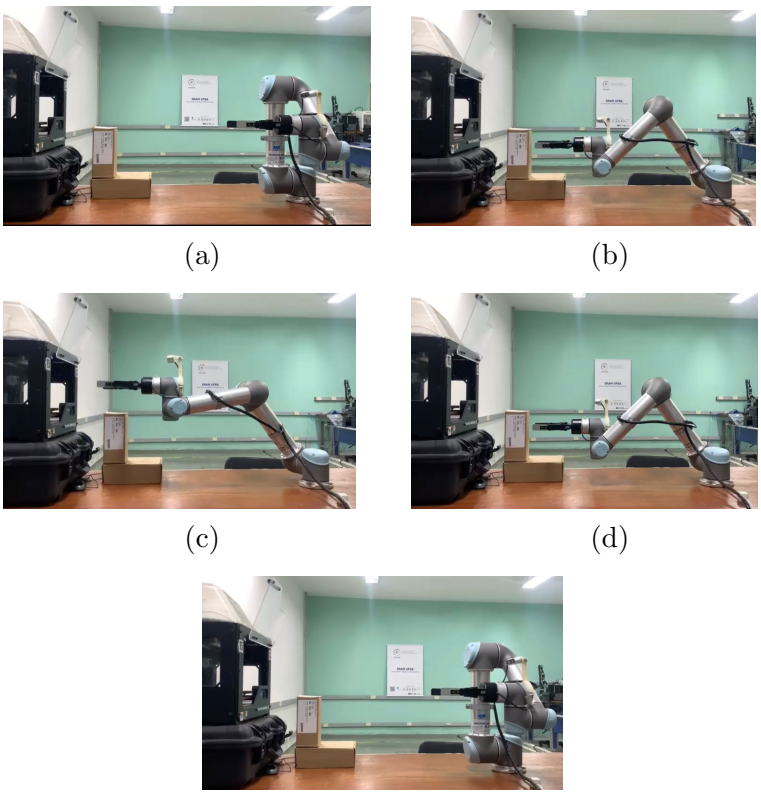


4.2.6 Real Scenario I

After the execution and validation of the algorithms in the already described simulation scenarios, tests were carried out in a real environment, in the robotics laboratory of the Federal University of Bahia. The experiments made for the real scenarios are an approach

task for a 3D printed part, while avoiding collisions with obstacles in the workspace. Figure 4.18 shows the real scenario I and the desired cobot routine.

Figure 4.18: Execution routine I.



(a) (b)

(c) (d)

(e)

Working with high quality image data for obstacle detection implies a high computational cost involved. As a workaround, and as the obstacles have known dimensions and static positions, we have modelled them as known geometric shapes, simplifying the input information. Thus, we have modelled 3D printer as a box of dimensions 0.11 x 0.5 x 0.75 m and also the objects in the workspace, following their respective dimensions. Figure 4.19 shows the simplified model side by side with the real scenario.

As in the simulation tests, we observed that the RRT-Connect algorithm generated longer duration trajectories, which often led the cobot to some kind of self-collision during the execution, this way, for security, we have executed the real tests only for CHOMP and STOMP. The success rate is exhibited in Table 4.6, and we can see that both algorithms have achieved high values, indicating that they are able to generate feasible trajectories for this scenario.

The trajectory referring to the displacement between frames 4.18b to 4.18c was se-

Figure 4.19: Real scenario and objects collision representation for 1st real routine.

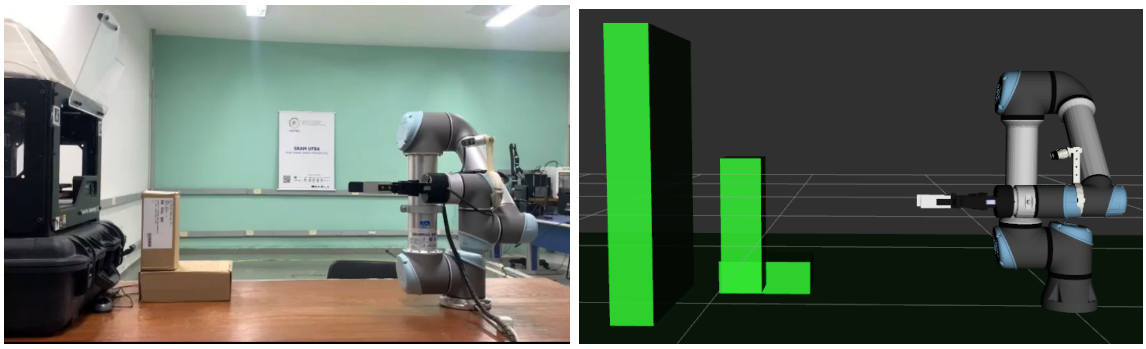


Table 4.6: Success Rate - Real Scene I.

Algorithm	CHOMP	STOMP
Success (%)	93.33	93.33

lected in order to analyze the planning time and the duration of the computed path. This trajectory characterizes an approach to the 3D printer, while avoiding collision with the boxes in the cobot's workspace. The results of obtained data for planning time can be seen in Figure 4.20, the algorithm STOMP obtained median value of 0.359 s, upper value of 0.489 s, and lower value of 0.245 s. CHOMP presented median value of 0.026 s, upper value of 0.062 s, and lower value of 0.02 s.

Results for trajectory duration are exhibited in Figure 4.21, with STOMP presenting median value of 3.0 s, upper value of 3.04 s, and lower value of 2.994 s; and CHOMP presenting median value of 4.684 s, upper value of 4.684 s, and lower value of 3.539 s. We can see that while STOMP needs a bit longer planning time than CHOMP, it delivers shorter trajectories, a similar behavior to what we have observed on simulated scenarios.

4.2.7 Real Scenario II

For the second real test scenario, we added a box of dimensions 0.12 x 0.12 x 0.25 m to the scenario seen in 4.2.6, increasing the difficulty for planning the trajectory. Figure 4.22 shows the real scenario 2 and the desired approach routine for the manipulator. We have also modelled the objects as known geometric shapes, as we have done in first real scenario, in order to reduce computational cost. Figure 4.23 shows the simplified model side by side with the real scenario.

Figure 4.20: Planning Time for each trajectory planner on real scene I.

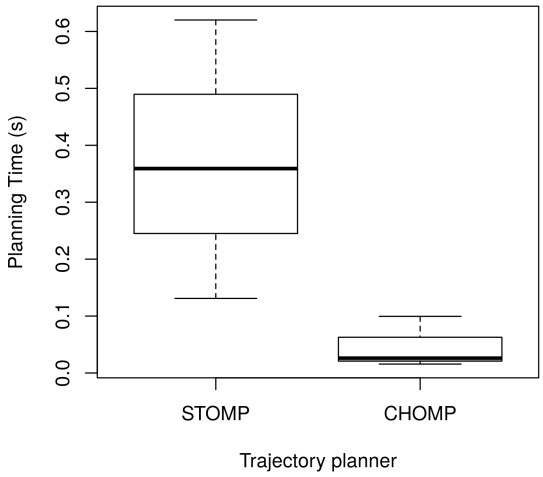
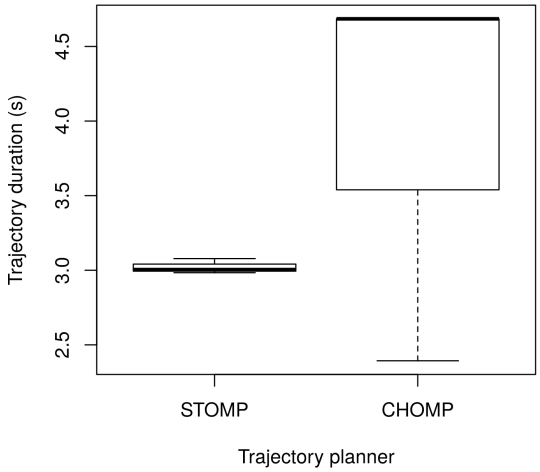


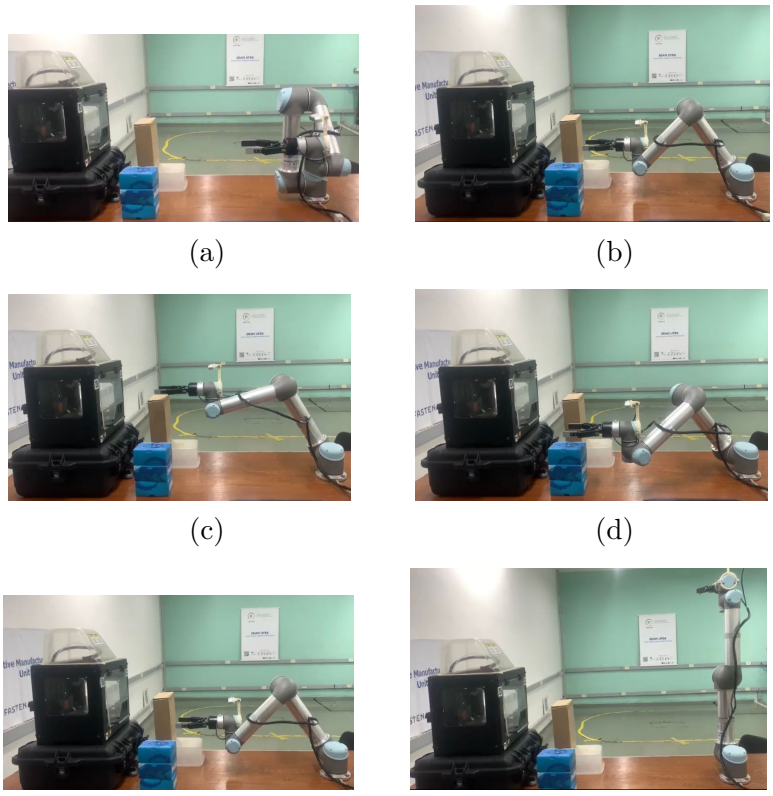
Figure 4.21: Duration of the generated path for each trajectory planner on real scene I.



The success rate for the second real scenario can be seen in Table 4.7. The obtained results were satisfactory for both algorithms, indicating that they are able to generate feasible trajectories for this scenario.

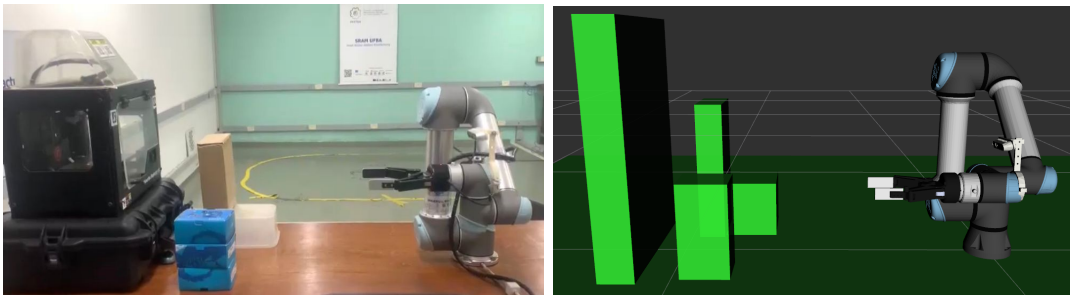
We have selected the trajectory referring to the displacement between frames 4.18c to 4.18d in order to analyze the planning time and the duration of the computed path. The selected trajectory characterizes an approach to the 3D printer, while avoiding collision with the boxes in workspace. The results of obtained data for planning time can be seen in Figure 4.24, STOMP obtained median value of 0.238 s, upper value of 0.326 s, and lower value of 0.221 s. CHOMP presented median value of 0.096 s, upper value of 0.097 s, and lower value of 0.096 s. Regarding trajectory duration, exhibited in Figure 4.25,

Figure 4.22: Execution routine II.



(a) (b) (c) (d) (e) (f)

Figure 4.23: Real scenario and objects collision representation for 2nd real routine.



STOMP presented median value of 2.314 s, upper value of 2.332 s, and lower value of 2.249 s, while CHOMP presented median value of 2.688 s, upper value of 2.733 s, and lower value of 2.653 s.

We can observe on real scenario II the same behavior to the tests performed earlier in this work, with STOMP needing a longer planning time than CHOMP and producing shorter trajectories. We have also observed a small variation in the collected data, this

Table 4.7: Success Rate - Real Scene II.

Algorithm	CHOMP	STOMP
Success (%)	92.33	96.66

Figure 4.24: Planning Time for each trajectory planner on real scene II.

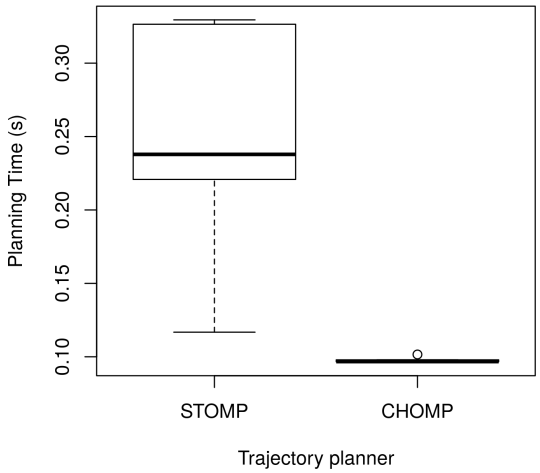
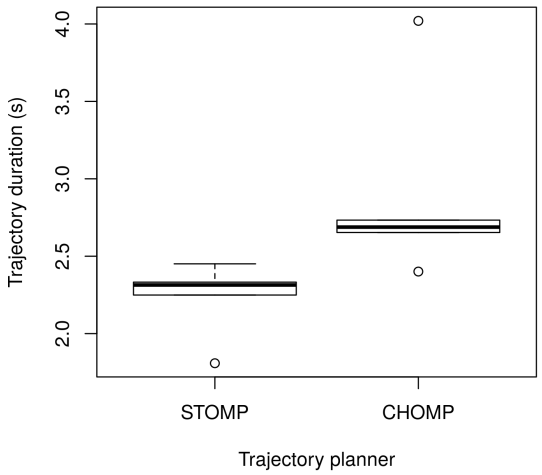


Figure 4.25: Duration of the generated path for each trajectory planner on real scene II.

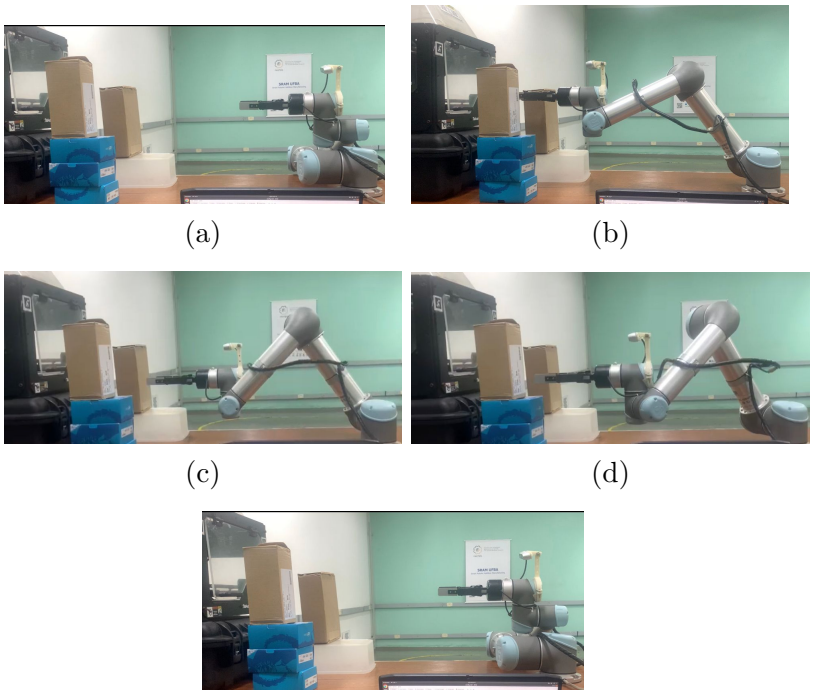


can be caused by the representation of collision objects as geometric shapes, which could lead the algorithms to plan trajectories of similar durations, and with similar planning times, in all performed executions.

4.2.8 Real Scenario III

The real scenario III can be seen in Figure 4.26. It is a more difficult scenario than 4.2.6 and 4.2.7, and in the same way as in the previous ones, the cobot should execute the approach routine while avoiding the obstacles in the scene.

Figure 4.26: Execution routine III.



(a) (b)

(c) (d)

(e)

As a way to provide more realistic results, we have decided to use camera data for object detection in this scenario. To make this possible, some steps were necessary. First, we connected the camera's and manipulator's frames using the static transform ROS tool, considering its relative position and orientation. Second, the topic `/camera/depth/color/points` was connected to the Moveit perception tool, providing a *pointcloud* with information about the environment, then represented as collision objects in the Planning Scene. In order to mitigate the high computational costs due to the use of the camera, we reduced the image size and used 10 fps. Figure 4.27 shows the object detection seen in Rviz side by side with the real scenario.

Table 4.8 summarizes the success rate for the third real scenario. The behavior is quite similar to all the performed experiments, assuring that both algorithms can generate feasible trajectories in environments with obstacles.

Figure 4.27: Real scenario and objects identification for 3rd real routine.

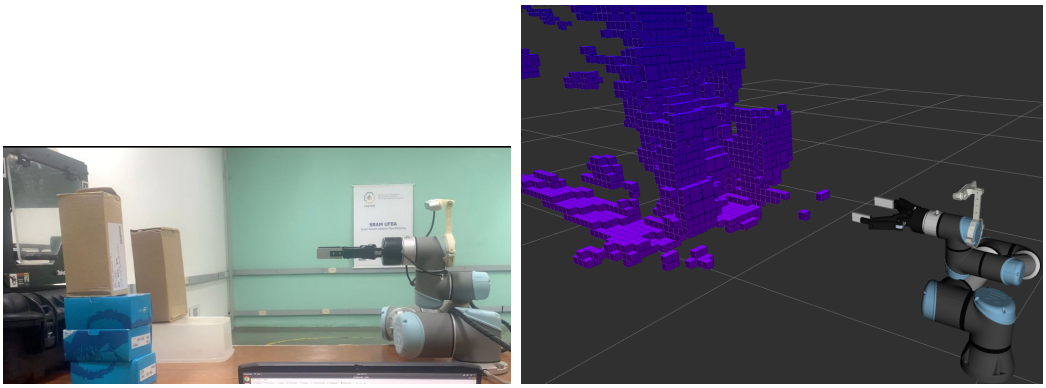


Table 4.8: Success Rate - Real Scene III.

Algorithm	CHOMP	STOMP
Success (%)	94.28	97.14

In a similar way with we have done before, the trajectory referring to the displacement between frames 4.22a to 4.22b was selected to analyze the planning time and the duration of the computed path, since this trajectory characterizes an approach to the 3D printer, while avoiding collision with obstacles in workspace. The results for planning time can be seen in Figure 4.28, STOMP obtained median value of 0.439 s, upper value of 0.682 s, and lower value of 0.369 s. CHOMP presented median value of 0.122 s, upper value of 0.188 s, and lower value of 0.014 s.

Regarding trajectory duration, exhibited in Figure 4.29, STOMP presented median value of 3.165 s, upper value of 3.181 s, and lower value of 3.111 s, while CHOMP presented median value of 3.463 s, upper value of 3.619 s, and lower value of 3.378 s. Results indicate that STOMP can generate trajectories of shorter duration than CHOMP, in a scenario with obstacles, despite needing more planning time.

Figure 4.28: Planning Time for each trajectory planner on real scene III.

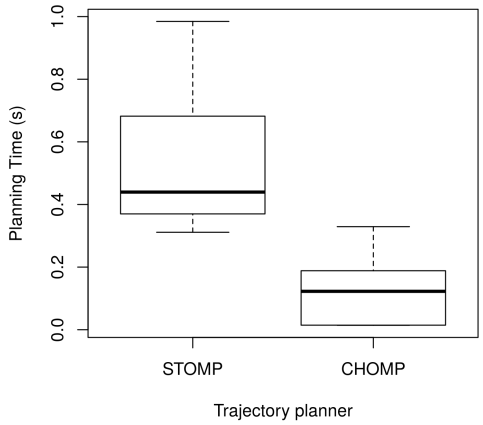
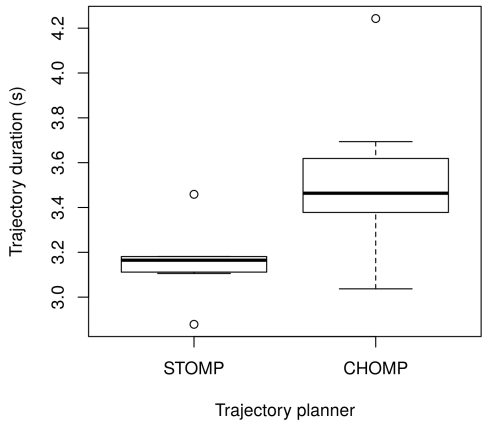


Figure 4.29: Duration of the generated path for each trajectory planner on real scene III.



4.2.9 Real Results Summary

The results obtained in 4.2.6, 4.2.7 and 4.2.8 have been condensed and are shown in the following. The values exhibited in Table 4.9 indicates that the three algorithms have obtained satisfactory results for success rate, generating feasible trajectories for all the scenarios.

Table 4.9: Success rate on real scenarios.

	CHOMP	STOMP
Success	93.33%	93.33%

Results on Figures 4.30 and 4.31 indicate that STOMP can generate trajectories of

shorter duration than CHOMP, in a scenario with obstacles, preventing the cobot from performing unnecessary movements, and reducing risks and efforts on its joints. Therefore, the collected data allow us to conclude that STOMP delivers smoother trajectories than the others algorithms analyzed in this work. By comparing the obtained results, we can also see that the addition of a perception sensor to the system on real scene 3 did not bring major impacts on the planning time.

Figure 4.30: Planning time by scene and algorithm on real scenarios.

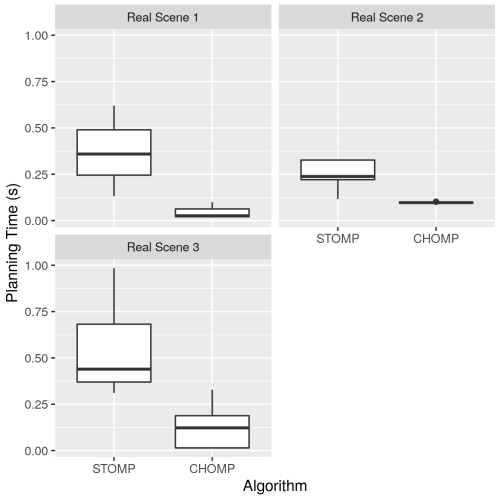
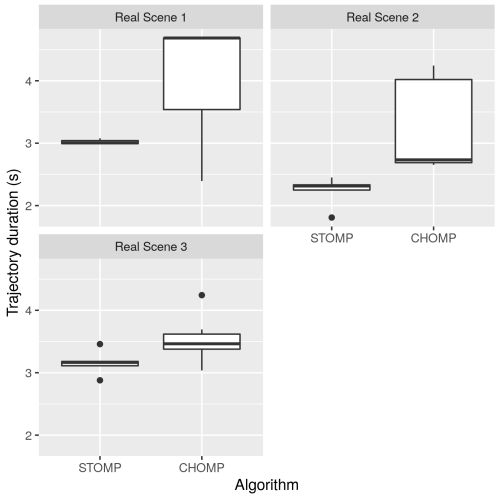


Figure 4.31: Trajectory duration by scene and algorithm on real scenarios.



Chapter 5

CONCLUSIONS

Collaborative robotics is a topic of growing relevance and the presence of these robots in the most diverse scenarios is increasingly frequent. In this context, in order to guarantee the safety of the operator and also of the device, it is important to ensure that the trajectory planning algorithm is capable of generating obstacle-free paths and that it can respect restrictions that may be desired.

In this work, we implemented a system for trajectory optimization of a collaborative robot UR5 ia a scenario with obstacles, using the algorithms CHOMP and STOMP. For obstacle detection and collision avoidance, data provided by a Point Cloud topic from an RGB+D sensor was used. According to the results, both studied algorithms presented similar performances for planning time and success rate. However, analyzing the duration of the trajectory, the STOMP algorithm showed better results, indicating that it has a greater capacity to deliver smoother trajectories to the robot, avoiding possible collisions with the environment, thus guaranteeing safety to the device and the operator.

Furthermore, due to STOMP stochastic optimization method, several other costs can be optimized, whether or not they are differentiable. It was identified that STOMP have usually took more time than the others to generate trajectories, and the use of another (faster) trajectory planner to generate the initial trajectory to be optimized by STOMP is an alternative that could reduce the algorithm planning time, contributing to better performance and keeping the quality of the final trajectory.

The scope of this work covers scenarios that contained static obstacles, and a suggestion of a possible future contribution to this would be the development of strategies for planning and replanning trajectories in environments with dynamic obstacles. This type of environment is even closer to the real operating environment for collaborative robotics,

in which an unexpected object can eventually enter the robot's operating environment and must be detected and avoided.

BIBLIOGRAPHY

ANDERSEN, R. S. Kinematics of a UR5. *Aalborg University: Aalborg, Denmark*, 2018.

BEESON, P.; AMES, B. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In: IEEE. *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. [S.l.], 2015. p. 928–935.

BRITO, T.; LIMA, J.; COSTA, P.; PIARDI, L. Dynamic Collision Avoidance System for a Manipulator Based on RGB-D Data. *Advances in Intelligent Systems and Computing*, v. 694, p. 643–654, 2018. ISSN 21945357.

CHITTA, S.; SUCAN, I.; COUSINS, S. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, IEEE, v. 19, n. 1, p. 18–19, 2012.

CONCEIÇÃO, A. G.; COSTA, F. S.; NASSAR, S. M.; GUSMEROLI, S.; SCHULTZ, R.; XAVIER, M.; HESSEL, F.; DANTAS, M. A. FASTEN IIoT: an open real-time platform for vertical, horizontal and end-to-end integration. *Sensors*, MDPI, v. 20, n. 19, p. 5499, 2020.

CORKE, P. *Robotics, vision and control: fundamental algorithms in MATLAB*. [S.l.]: Springer, 2011.

CYBERBOTICS. *Webots - Open Source Robot Simulator*. 2021. Disponível em: ⟨https://cyberbotics.com/⟩. Acesso em: 2021-09-23.

EVJEMO, L. D.; GJERSTAD, T.; GRØTLI, E. I.; SZIEBIG, G. Trends in smart manufacturing: Role of humans and industrial robots in smart factories. *Current Robotics Reports*, Springer, v. 1, n. 2, p. 35–41, 2020.

GASPARETTO, A.; SCALERA, L. From the unimate to the delta robot: the early decades of industrial robotics. In: *Explorations in the History and Heritage of Machines and Mechanisms*. [S.l.]: Springer, 2019. p. 284–295.

GRUSHKO, S.; VYSOCKY, A.; JHA, V. K.; PASTOR, R.; PRADA, E.; MIKOVA, L.; BOBOVSKY, Z. Tuning perception and motion planning parameters for moveit! framework. *MM Science Journal*, v. 2020, n. November, p. 4154–4163, 2020. ISSN 18050476. Disponível em: ⟨https://dx.doi.org/10.17973/MMSJ.2020_11_2020064⟩.

HAN, D.; NIE, H.; CHEN, J.; CHEN, M. Optimal randomized path planning for redundant manipulators based on Memory-Goal-Biasing. *International Journal of Advanced Robotic Systems*, v. 15, n. 4, p. 1–11, 2018. ISSN 17298814. Disponível em: ⟨https://dx.doi.org/10.1177/1729881418787049⟩.

IEEE. *Unimate*. 2018. Disponível em: ⟨http://robots.ieee.org/robots/unimate/⟩. Acesso em: 2021-08-01.

INFOLAB, S. *Robots and their Arms*. 2019. Disponível em: ⟨http://infolab.stanford.edu/pub/voy/museum/pictures/display/1-Robot.htm⟩. Acesso em: 2021-08-01.

INFORMATION, H. *Asea Produces the IRb 6, the First Microcomputer Controlled Electric Industrial Robot*. 2021. Disponível em: ⟨https://www.historyofinformation.com/detail.php?entryid=4352⟩. Acesso em: 2021-08-01.

INTEL. *Intel RealSense Product Family D400 Series*. [S.l.], 2022. Rev. 13. Disponível em: ⟨https://www.intelrealsense.com/wp-content/uploads/2022/05/Intel-RealSense-D400-Series-Datasheet-April-2022.pdf⟩.

JUNG, H.; KIM, M.; CHEN, Y.; MIN, H. G.; PARK, T. Implementation of a unified simulation for robot arm control with object detection based on ROS and Gazebo. *2020 17th International Conference on Ubiquitous Robots, UR 2020*, p. 368–372, 2020. Disponível em: ⟨https://doi.org/10.1109/UR49135.2020.9144984⟩.

KADEN, S.; THOMAS, U. Maximizing Robot Manipulability along Paths in Collision-free Motion Planning. In: *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, 2019. p. 105–110. ISBN 978-1-7281-2467-4. Disponível em: ⟨https://dx.doi.org/10.1109/ICAR46387.2019.8981591⟩.

KALAKRISHNAN, M.; CHITTA, S.; THEODOROU, E.; PASTOR, P.; SCHAAL, S. STOMP: Stochastic trajectory optimization for motion planning. p. 4569–4574, 2011. Disponível em: ⟨https://doi.org/10.1109/ICRA.2011.5980280⟩.

KAVRAKI, L. E.; SVESTKA, P.; LATOMBE, J.-C.; OVERMARS, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, IEEE, v. 12, n. 4, p. 566–580, 1996.

KEATING, R. *UR5 Inverse Kinematics*. 2014. Disponível em: ⟨https://github.com/yorgoon/ur5_Final_Project/blob/master/UR5_Inverse_Kinematics.pdf⟩. Acesso em: 2021-08-15.

KEBRIA, P. M.; AL-WAIS, S.; ABDI, H.; NAHAVANDI, S. Kinematic and dynamic modelling of ur5 manipulator. In: IEEE. *2016 IEEE international conference on systems, man, and cybernetics (SMC)*. [S.l.], 2016. p. 004229–004234.

KUFFNER, J. J.; LAVALLE, S. M. Rrt-connect: An efficient approach to single-query path planning. In: IEEE. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. [S.l.], 2000. v. 2, p. 995–1001.

KUMAR, S.; MAJUMDER, A.; DUTTA, S.; RAJA, R.; JOTAWAR, S.; KUMAR, A.; SONI, M.; RAJU, V.; KUNDU, O.; BEHERA, E. H. L.; VENKATESH, K. S.; SINHA,

R. Design and Development of an automated Robotic Pick & Stow System for an e-Commerce Warehouse. p. 1–15, 2017. Disponível em: ⟨http://arxiv.org/abs/1703.02340⟩.

LAVALLE, S. M.; JR, J. J. K. Randomized kinodynamic planning. *The international journal of robotics research*, SAGE Publications, v. 20, n. 5, p. 378–400, 2001.

MAHTANI, A.; SANCHEZ, L.; FERNÁNDEZ, E.; MARTINEZ, A. *Effective robotics programming with ROS*. [S.l.]: Packt Publishing Ltd, 2016.

MEJIA-TRUJILLO, J. D.; CASTANO-PINO, Y. J.; NAVARRO, A.; ARANGO-PAREDES, J. D.; RINCÓN, D.; VALDERRAMA, J.; MUNOZ, B.; OROZCO, J. L. Kinect™ and intel realsense™ d435 comparison: A preliminary study for motion analysis. In: IEEE. *2019 IEEE International Conference on E-health Networking, Application & Services (HealthCom)*. [S.l.], 2019. p. 1–4.

MOHAMED, M. Challenges and benefits of industry 4.0: an overview. *International Journal of Supply and Operations Management*, Kharazmi University, v. 5, n. 3, p. 256–265, 2018.

MOVEIT. *Concepts*. 2018. Disponível em: ⟨http://moveit.ros.org/documentation/concepts/⟩. Acesso em: 2021-08-23.

MOVEIT. *Concepts - Moveit*. 2022. ⟨https://github.com/ros-industrial/universal_robot⟩. Accessed: 2022-09-26.

OLSEN, T. L.; TOMLIN, B. Industry 4.0: Opportunities and challenges for operations management. *Available at SSRN: https://ssrn.com/abstract=3365733*, 2019.

PAVLICHENKO, D.; BEHNKE, S. Efficient stochastic multicriteria arm trajectory optimization. *IEEE International Conference on Intelligent Robots and Systems*, v. 2017-Septe, n. September, p. 4018–4025, 2017. ISSN 21530866. Disponível em: ⟨https://dx.doi.org/10.1109/IROS.2017.8206256⟩.

PENG, Y. C.; CHEN, S.; JIVANI, D.; WASON, J.; LAWLER, W.; SAUNDERS, G.; RADKE, R. J.; TRINKLE, J.; NATH, S.; WEN, J. T. Sensor-guided assembly of segmented structures with industrial robots. *Applied Sciences (Switzerland)*, v. 11, n. 6, 2021. ISSN 20763417.

PYO, Y.; CHO, H.; JUNG, R.; LIM, T. *ROS Robot Programming*. [S.l.]: Robotis, 2017.

QUIGLEY, M.; CONLEY, K.; GERKEY, B.; FAUST, J.; FOOTE, T.; LEIBS, J.; WHEELER, R.; NG, A. Y. et al. Ros: an open-source robot operating system. In: KOBE, JAPAN. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5.

ROBOTICS, C. *Clearpath Website*. 2022. Disponível em: ⟨https://clearpathrobotics.com/⟩.

ROBOTICS, K. *Kinova Website*. 2022. Disponível em: ⟨https://www.kinovarobotics.com⟩.

ROBOTIS. *Robotis Website*. 2022. Disponível em: ⟨https://www.robotis.us/⟩.

ROBOTS, U. *Collaborative Robots from Universal Robots*. 2021. Disponível em: ⟨https://www.universal-robots.com/products/⟩. Acesso em: 2021-08-01.

ROBOTS, U. *UR5 collaborative robot arm — flexible and lightweight robot arm.* 2022. ⟨https://www.universal-robots.com/products/ur5-robot/⟩. Accessed: 2022-05-15.

ROS. *ROS Website*. 2017. Disponível em: ⟨http://www.ros.org⟩.

ROS.ORG. *ROS - Introduction*. 2018. Disponível em: ⟨http://wiki.ros.org/ROS/Introduction/⟩. Acesso em: 2021-08-17.

SHERWANI, F.; ASAD, M. M.; IBRAHIM, B. Collaborative robots and industrial revolution 4.0 (ir 4.0). In: IEEE. *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*. [S.l.], 2020. p. 1–5.

SICILIANO, B.; SCIAVICCO, L.; VILLANI, L.; ORIOLO, G. *Robotics: modelling, planning and control*. [S.l.]: Springer Science & Business Media, 2010.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot modeling and control*. [S.l.]: John Wiley & Sons, 2005. 1st edition.

SUCAN, I. A.; MOLL, M.; KAVRAKI, L. E. The open motion planning library. *IEEE Robotics & Automation Magazine*, IEEE, v. 19, n. 4, p. 72–82, 2012.

XU, X.; DUGULEANA, M. Trajectory Planning of 7-Degree-of-Freedom Manipulator Based on ROS. *IOP Conference Series: Materials Science and Engineering*, v. 677, n. 5, 2019. ISSN 1757899X. Disponível em: ⟨https://dx.doi.org/10.1088/1757-899X/677/5/052072⟩.

YE, L.; SUN, C. Trajectory planning of 7-DOF redundant manipulator based on ROS platform. *Proceedings of 2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence, ICIBA 2020*, n. Iciba, p. 733–736, 2020. Disponível em: ⟨https://dx.doi.org/10.1109/ICIBA50161.2020.9277001⟩.

ZHANG, H. D.; LIU, S. B.; LEI, Q. J.; HE, Y.; YANG, Y.; BAI, Y. Robot programming by demonstration: a novel system for robot trajectory programming based on robot operating system. *Advances in Manufacturing*, Shanghai University, v. 8, n. 2, p. 216–229, 2020. ISSN 21953597. Disponível em: ⟨https://doi.org/10.1007/s40436-020-00303-4⟩.

ZUCKER, M.; RATLIFF, N.; DRAGAN, A. D.; PIVTORAIKO, M.; KLINGENSMITH, M.; DELLIN, C. M.; BAGNELL, J. A.; SRINIVASA, S. S. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 32, n. 9-10, p. 1164–1193, 2013.