Federal University of Bahia

MASTER THESIS

Anticollision System Applied to Robotic Manipulators Based on Artificial Potential Fields

Ubiratan de Melo Pinto Junior

Postgraduate Program in Electrical Engineering

Salvador
2020

UBIRATAN DE MELO PINTO JUNIOR

ANTICOLLISION SYSTEM APPLIED TO ROBOTIC MANIPULATORS BASED ON ARTIFICIAL POTENTIAL FIELDS

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
2020

P659 Pinto Junior, Ubiratan de Melo.
Anticollision system applied to robotic manipulators based on artificial potential fields/ Ubiratan de Melo Pinto Junior. – Salvador, 2020.
78 p.: il. color.

Orientador: Prof. Dr. André Gustavo Scolari Conceição.

Dissertação (mestrado) – Universidade Federal da Bahia. Escola Politécnica, 2020.

1. Robôs. 2. Campos potenciais artificiais. 3. Controle de orientação. 4. Rota - planejamento. I. Conceição, André Gustavo Scolari. II. Universidade Federal da Bahia. III. Título.

CDD: 629.89

Ubiratan de Melo Pinto Junior

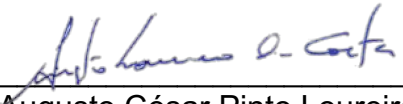
“SISTEMA DE ANTICOLISÃO APLICADO A MANIPULADORES ROBÓTICOS COM BASE EM CAMPOS POTENCIAIS ARTIFICIAIS”.

Dissertação apresentada à Universidade Federal da Bahia, como parte das exigências do Programa de Pós-Graduação em Engenharia Elétrica, para a obtenção do título de Mestre.

APROVADA em: 16 de Dezembro de 2020.

BANCA EXAMINADORA

Prof. Dr. André Gustavo Scolari Conceição
Orientador/UFBA



Prof. Dr. Augusto César Pinto Loureiro da Costa
UFBA

Prof. Dr. Rodrigo Antônio Carlos Braga
UFSC

ACKNOWLEDGEMENTS

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

A Deus por todos os privilégios que tenho a benção de ter.

A meus pais e irmã, Ubiratan de Melo, Marta Almeida e Priscila Almeida por sempre me apoiarem nas minhas decisões e por acreditarem na minha capacidade. Espero corresponder às suas expectativas.

A Raiane Andrade, pelas conversas, conselhos, compreensão, apoio, companheirismo e inspiração durante todas as minhas jornadas. Amo você.

A Cássio Motta, Lucas Cruz, Rafael Trotsiuk e Vitor Passos por toda a amizade, pois quem tem amigos, tem tudo.

A Vera, André, Joisson por serem uma segunda família.

A André Scolari por aceitar me orientar neste trabalho e sempre estar à disposição para me apoiar.

A Caio Viturino pela parceria e por tornar possível este trabalho.

A Hamilton Araújo e Rodrigo Bastos pelos conselhos acadêmicos.

A Rafael Vieira, Fabrício Ferreira e Roberto Costa pelo exemplo, liderança e amizade.

A todo o time da Ford Motor Company, em especial Anderson Doi, Alexandre Costa, Alexandre Moura, Álvaro Peixoto, Caio Nunes, Estácio Lima, Henrique Oliveira, Israel Vasconcelos, Josafá Ramos, Karolie Grassi, Marcelo Pereira, Marx Figueiredo, Matheus Cruz, Lucas Miranda, Rafael Machado, Rafael Santos, Rubens Neto e Ueslei Neves por todo o aprendizado e trabalho em equipe.

A toda equipe e amigos do Colégio Helyos em especial Erivaldo Santana, Filipe Nicolaj, Gerson Júnior, João Amaro, Marcelo Brites e Patrícia Moldes por todo o alicerce e exemplo que recebi e que são grande parte do que construí até hoje.

O homem há de voar.

—SANTOS DUMONT

RESUMO

Esta dissertação de mestrado propõe a integração de um algoritmo de Campos Potenciais Artificiais Adaptativos com uma nova técnica de controle de orientação para planejamento de rotas aplicado a manipuladores robóticos. O desenvolvimento de sistemas autônomos robóticos passou por vários avanços na área de algoritmos de planejamento de rotas. Esses sistemas geram rotas livres de colisão com objetos no espaço de trabalho do robô. Neste contexto, a técnica dos Campo Potenciais Artificiais tem sido foco de melhorias nos últimos anos devido à sua simplicidade de aplicação and eficiência em sistemas de tempo real, desde que não seja requerido o mapeamente do espaço de trabalho do robô. Apesar da sua eficiência, esta técnica é susceptível a problemas de mínimos locais de diferentes naturezas, tais como o *"Goals Non-Reachable with Obstacles Nearby"* (GNRON). Para resolver este problema, é sugerido o uso de uma melhoria chamada Campos Potenciais Artificiais Adaptativos usada em conjutno com a técnica de controle de orientação do efetuador proposta. A força resultante, gerada dos Campos Potenciais Artificiais Adaptativos, guia o efetuador do robô para o objetivo. O framework *Robot Operating System* (ROS) e o robô manipulador colaborativo UR5 são utilizados para validar o método proposto no posicionamento da ferramenta do efetuador final em tarefas de pick and place.

Palavras-chave: Campos Potenciais Artificiais, Controle de Orientação, Manipuladores Robóticos, Planejamento de Rotas

ABSTRACT

This Master's thesis proposes the integration of an Adaptive Artificial Potential Fields algorithm with a new end effector orientation control technique for real-time path planning applied to robotics manipulators. The development of autonomous robotic systems has undergone several advances in path planning algorithms. These systems generate object collision-free paths in the robot's workspace. In this context, the Artificial Potential Fields technique has been the focus of improvements in recent years due to its simplicity of application and efficiency in real-time systems, since it does not require a global mapping of the robot's workspace. In spite of its efficiency, this technique is susceptible to local minimum problems of different natures, such as *Goals Non-Reachable with Obstacles Nearby* (GNRON). To solve this problem, it is suggested the use of an improvement called Adaptive Artificial Potential Fields used in conjunction with the proposed end effector orientation control technique, which allows reaching a desired orientation of the end effector. The resulting force, generated from the Adaptive Artificial Potential Field, guides the robot end effector to the goal. The *Robot Operating System* (ROS) framework and a collaborative robot manipulator UR5 are used to validate the proposed method on an approaching of the end-effector tool in a pick and place task.

Keywords: Artificial Potential Field, Orientation Control, Robotic manipulators, Path planning

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AAPF Adaptative Artificial Potential Fields. 5

ACO Ant Colony Optimization. 4

APF Artificial Potential Fields. 4

CAD Computer Aided Design. 1

CD Cell Decomposition. 4

DH Denavit-Hartenberg. 14

FA Firefly Algorithm. 4

FL Fuzzy Logic. 4

GA Genetic Algorithm. 4

GNRON Goals Non-Reachable with Obstacles Nearby. 5

IOT Internet of Things. 1

MAPF Modifed Artifcial Potential Fields. 5

NN Neural Networks. 4

PRM Probabilistic Roadmap Method. 5

PSO Particle Swarm Optimization. 4

RA Roadmap Approach. 4

RLMP Reacharound Local Minimum Problem. 5

ROS Robot Operating System. 6

RTT Rapidly-exploring Random Tree. 4

SIFORS Simultaneous Forward Search Method. 5

SLPRM Semi-Lazy Probabilistic Roadmap Method. 5

UAVs Unmanned Aerial Vehicles. 5

Chapter 1

INTRODUCTION

The so called industry 4.0 is bringing changes in the relationship between the physical and the virtual worlds. Almost instantaneous connections with devices at long distances, powerful processing capabilities in embedded systems, algorithms capable of teaching itselves and other technological advances are making possible new ways of production tangible goods, management of processes and people and deliver services. Some technologies that are responsible for those enhancements are: additive manufacturing, Internet of Things (IOT), artificial intelligence, advanced robotics, among others (OLSEN; TOMLIN, 2019). In fact, the Industry 4.0 is not only the advances in all those areas, but the potential of interaction and synergies between subsets of these technologies.

Additive manufacturing, also known as 3D printing is the process of producing pieces from a Computer Aided Design (CAD), that is virtually sliced into a set of flat horizontal layers models, and then, through some technology as material jetting, powder bed fusion, vat polymerization or other, is transferred to a printer to produce the object layer by layer. This brings great changes in operational quality, flexibility, speed and cost.

The avarage price of a sensor fell by 90% and microprocessor clockspeeds increased by a factor of 991 between 1992 and 2014 (HOLDOWSKY et al., 2015). Those facts allowed improvements in measurements and communications technologies in embedded systems and connection and monitoring of large sets of devices that can take actions according to real-time changes in the environment. This is the concept of Internet of Things (IOT), where large sets of data are available at any time so decisions can be made much faster and reliably. For example, sensor in a room can monitor its occupancy, temperature, humidity and other variables and this data can be sent over internet to several devices, with different technologies, that can process, store and actuate in the environment autonomously.

Artificial intelligence has not a unique definition. Usually it it refers to algorithms that have a similar behaviour to humans, i.e. it has the capability to adapt and learn and improve its performance. This learning process can be supervised or unsupervised. Several types of algorithms are considered artificial intelligence, such as neural networks,

deep learning algorithms, computer vision, machine learning and others. Artificial intelligence industrial applications can be categorized as descriptive, predictive or prescriptive. Descriptive applications are related to categorization and identification of objects or specific features. Predictive applications have the ability to forecast systems conditions and output based on the understanding on the knowledge of cause-and-effect relationships. Prescriptive applications have a goal and solve it for the optimal decision based on the algorithm used.

Conventional robots in a factory are typically placed within cages so it can safely perform its tasks without threatening the workers around it. This is because was not possible to equip robots with sensors and intelligence to adapt in a constantly changing environment. But the advances in sensors and artificial intelligence allow the collaboration between humans and robots in the same work space and in the same activities. Robots capable of collaborative work are also called "cobots". Examples of cobots are the UR Series robots, figure 1.1, from Universal Robots (ROBOTS, 2019a).

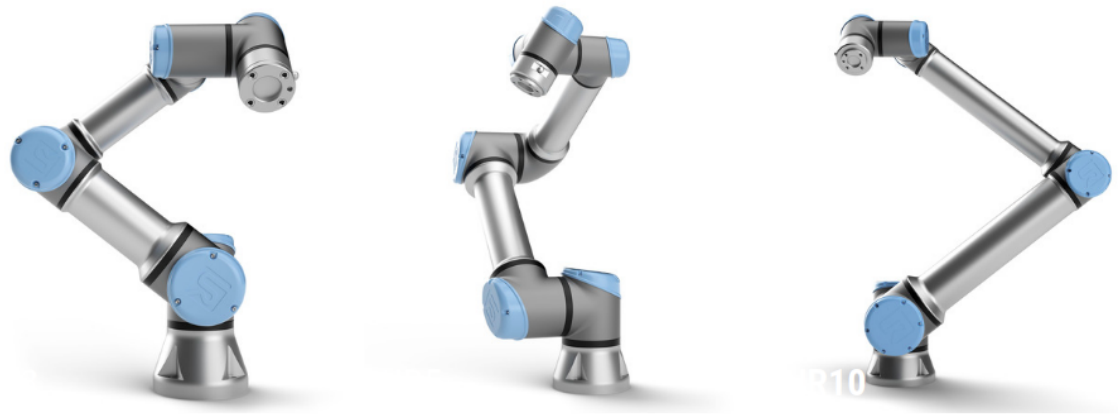


Figure 1.1: Collaborative UR Series from Universal Robots (from left to right: UR3, UR5 and UR10)

Occupational health and safety criteria are of great importance in the implementation of collaborative robotics (GUALTIERI; RAUCH; VIDONI, 2021). From 2015 to 2018, the papers related to this area can be divided in two categories: safety and ergonomics. Safety being divided in two sub-categories: contact avoidance and contact detection and mitigation. And ergonomics divided in: physical ergonomics and cognitive and organizational ergonomics.

Figure 1.2 show that contact avoidance and contact detection and mitigation are the most studied topic within the sub-categories. Also, from figure 1.3 can be noticed that all areas presented a growing in the scientific production in the last years. This points to the increasing relevance of collaborative robotics in science and industrial applications.

In the sub-categories of contact avoidance and contact detection and mitigation, works in motion planning and control represents 12.4% and 6.2%, respectively. Therefore, being a crucial theme in the area (GUALTIERI; RAUCH; VIDONI, 2021). The present work has the main focus in this area.

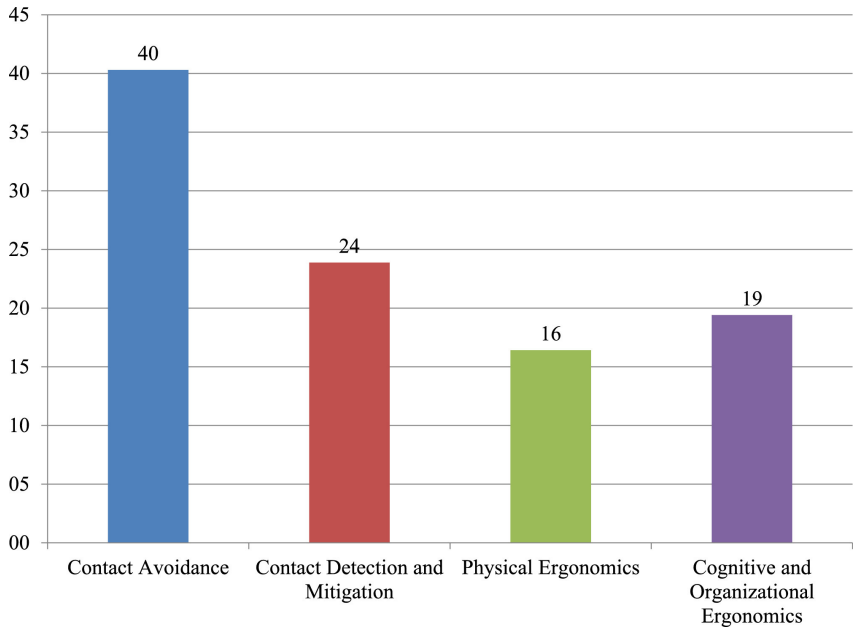


Figure 1.2: Percentage distribution of papers for each sub-category (GUALTIERI; RAUCH; VIDONI, 2021)

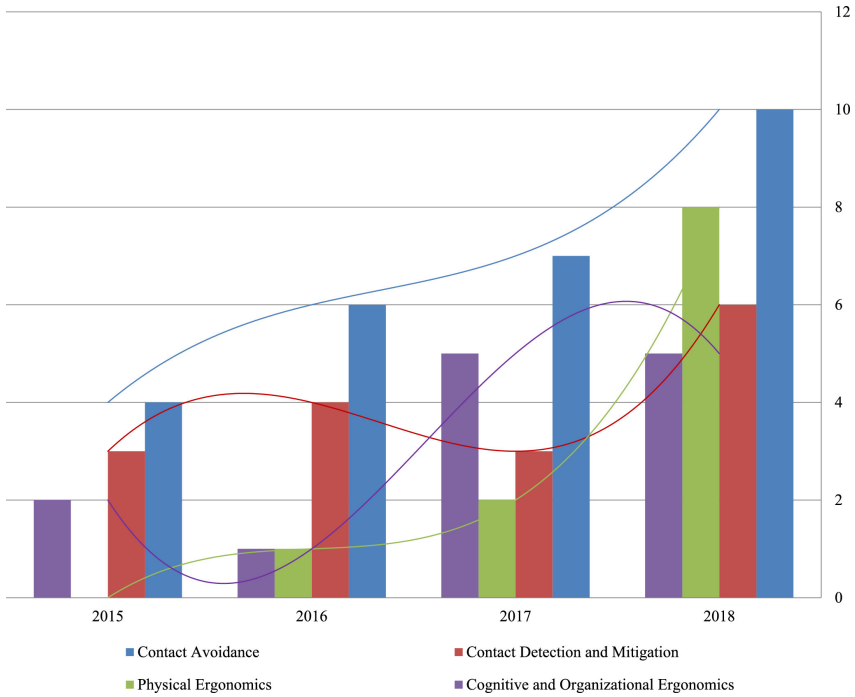


Figure 1.3: Total annual paper production per sub-category (* referring to a two year time period) (GUALTIERI; RAUCH; VIDONI, 2021)

Trajectory planning and control has to main objective to provide solutions to the movement of the robots through algorithms that optimize to several criteria, such as: collision avoidance, shortest path, smooth trajectories, velocity and acceleration control et cetera. Additionally, the optimization can be made in real-time or prior to the motion to start; rely on previous knowledge of the environment, or get the feedback of sensors monitoring variables of interest in the ambient. The present work proposes a path planning algorithm for obstacle avoidance based on the Artificial Potential Fields technique, assuming full knowledge of the static environment and made prior to the movement.

1.1 LITERATURE REVIEW

Several works in path planning algorithms with obstacle avoidance for robots, mobile and manipulators, have been published (PATLE et al., 2019; Isenberg, 2017; Kar et al., 2016; SAFEEA M., 2020; Zhang et al., 2018; Zahroof et al., 2019; XIN et al., 2016). Navigation techniques can be divided in classical approaches - Cell Decomposition (CD), Roadmap Approach (RA), Artificial Potential Fields (APF) - or reactive approaches - Genetic Algorithm (GA), Fuzzy Logic (FL), Neural Networks (NN), Firefly Algorithm (FA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), among other. Reactive approches are more popular and widely used in path planning of mobile robots as they perform better than classical approaches because of their higher capability to handle uncertainty in the environment and are preferably used for real time applications. Also, better performance of classical approaches can be achieved by hybridizing them with reactive approaches (PATLE et al., 2019).

There are solutions for trajectory optimization to reach goals under kinodynamic constraints (SINTOV, 2019). In this work, the algorithm chooses a optimal solution based on a cost function. Simulations were made for a 6R manipulator and the algorithm plans the path off-line prior to the motion. In (SAFEEA M., 2020), the application of Newton method to collision avoidance problems and path planning for manipulators with high degrees of freedom is studied. Compared to the classical gradient descent method, the performance is improved with respect to motion generation, presence of oscillations, gain tuning and convergence velocity. In (Isenberg, 2017), a path planning method is presented for the presence of both obstacles and forbidden axes of rotation. This approach uses geometry of unit-quaternion motion with potential fields.

In (Zahroof et al., 2019), a fast motion planning tool used for collision-free trajectories generation is presented for satellite servicing. It also incorporates perception of constraints in the environment and the continuous track of objects or of the end-effector. In (XIN et al., 2016), the problem of path planning for redundant robots in unknown dynamic environments is studied and a real-time dynamic system is proposed.

A common problem in classical path planning approaches is the presence of local minima in the several cases of environments tested. (Zhang et al., 2018) proposes a improved Rapidly-exploring Random Tree (RTT) algorithm simulated in complex environments that can improve the success rate and efficiency of the planning.

The increasing research in the area of collision avoidance using the APF method has been remarkable due to its typically small computation times (Quiroz-Omaña; Adorno,

2019). The application areas include manipulator robots, mobile robots, autonomous cars, Unmanned Aerial Vehicles (UAVs), autonomous underwater vehicles, among others. Some works in the area, as proposed by (GE; CUI, 2000), developed a technique called Modifed Artifcial Potential Fields (MAPF) to solve Goals Non-Reachable with Obstacles Nearby (GNRON) problems in mobile robots. An expansion to a three-dimensional space of this technique was presented by (LUO; SU; WANG, 2012), but increasing the chances of stagnation in a local minimum. (ZHANG; LIU; CHANG, 2017) developed the so-called Adaptative Artificial Potential Fields (AAPF) to solve this MAPF problem. In this method, concepts of classical APFs are used, when distant from the objective, and MAPFs, when it is close to the objective. Thus, the path travelled does not face problems related to local minimum caused by the repulsive potential fields and, at the same time, is able to solve the GNRON problem.

The integration of classic APFs into Collision Cone Approach was proposed by (KIM et al., 2016), so that a possible collision between mobile robots and obstacles can be predicted. (LI et al., 2015) presented a method of path planning for mobile robots in known, partially known or totally unknown environments. The APFs were integrated into the Simultaneous Forward Search Method (SIFORS) to find a valid and short path to the objective.

APFs have some local minimum problems specific of mobile and manipulators robots. Mobile robots have greater freedom of movement, whereas in a chain of rigid bodies of a manipulator, the movement of each joint changes the position of the previous joint to reach a goal determined by the path planning algorithm. When attempting to reach a final position, a robotic manipulator may stagnate to a local minimum due to the repulsive forces acting on the links, a problem known as Reacharound Local Minimum Problem (RLMP). (BYRNE; NAEEM; FERGUSON, 2013) used the methods known as *Goal Configuration Sampling*, *Subgoal-Selection*, based on the *Sampling-based* method and the *Expanded Convex Hull* algorithm, to avoid RLMPs local minimum caused by APFs. (AKBARIPOUR; MASEHIAN, 2017) developed a method that integrates Probabilistic Roadmap Method (PRM) and *Lazy-PRM* algorithms. This method was named Semi-Lazy Probabilistic Roadmap Method (SLPRM) and it is based on Sampling-based algorithms. The method was developed for application in robot manipulators. The results showed that the computational efficiency of the *SLPRM* algorithm is higher when compared to the PRM and the *Lazy-PRM* algorithms.

1.2 OBJECTIVES OF THIS WORK

The main objective of this work is to propose improvements to the Classic Artificial Potential Fields (APF) method described in (KHATIB, 1986), more specifically, solutions to known problems of local minima such as Goals Non Reachable with Obstacles Nearby (GNRON) and Reacharound Local Minimum Problem (RLMP), so it can be used in collaborative scenarios between human and robotic manipulators and provides the robots to autonomously avoid collisions during the execution of tasks. In order to accomplish the main objective, the following specific objectives were achieved:

- study of the main algorithms based on Artificial Potential Fields for collision avoid-

ance;

- implement in C++/Python on Robot Operating System (ROS) the techniques proposed by (GE; CUI, 2000), (BYRNE; NAEEM; FERGUSON, 2013), (LUO; SU; WANG, 2012) and (ZHANG; LIU; CHANG, 2017) to solve the APF local minima;

- evaluate the performance of the implemented techniques in simulated and real scenarios;

- implement orientation control techniques to the end-effector so it can achieve the desired position to do the proposed task.

1.3 METHODOLOGY

The first stages of this work consisted in a literature review of the area with the main objective of find research gaps and determine the state of the art of path planning algorithms with obstacles avoidance. A special focus on the Artificial Potential Fields (APF) technique.

A common problem present in path planning algorithms is the local minimum problem. This problem consists in the stagnation of the robot before it really reaches the desired goal, that is the global minimum and can occur for several reasons, depending in the planning method and the specific configuration and complexity of the environment as well as the type o robot used. For the APF, for example, a type of local minimum problem that only occurs in manipulators is the Reacharound Local Minimum Problem, or RLMP, that consists in the inability of the manipulator to reach a goal due to the presence of a obstacle the goal and some area of the robot.

Because of that, some local minimum problems were selected based on the literature review and methods to solve or mitigate those problems were searched. And then the main goal were to implement proposed methods or propose new ones either by modification of the existing methods or creation of new ones.

This implementation took place in the *Robot Operating System* (ROS) (ROS.ORG, 2017) framework and with the Gazebo simulation tool. All the code was made with Python programming language and the open source packages available within ROS. The robot used for the experiments was the UR5, from Universal Robots (ROBOTS, 2019b) and all the CAD models and ROS drivers are made available by the Universal Robots in the GitHub. An example of the programming and simulation setup is depicted in figure 1.4.

The use case selected for the experiments was a approaching task of the manipulator to a part inside a 3D printer. This approach should be made avoiding collisions with the environment and the environment is assumed to be previously known and static. The CAD environment was developed to reproduce the real laboratory from the Federal University of Bahia.

After the simulated experiments were completed and validated, the experiments were repeated in the real hardware. The results obtained will be discussed latter in this document.

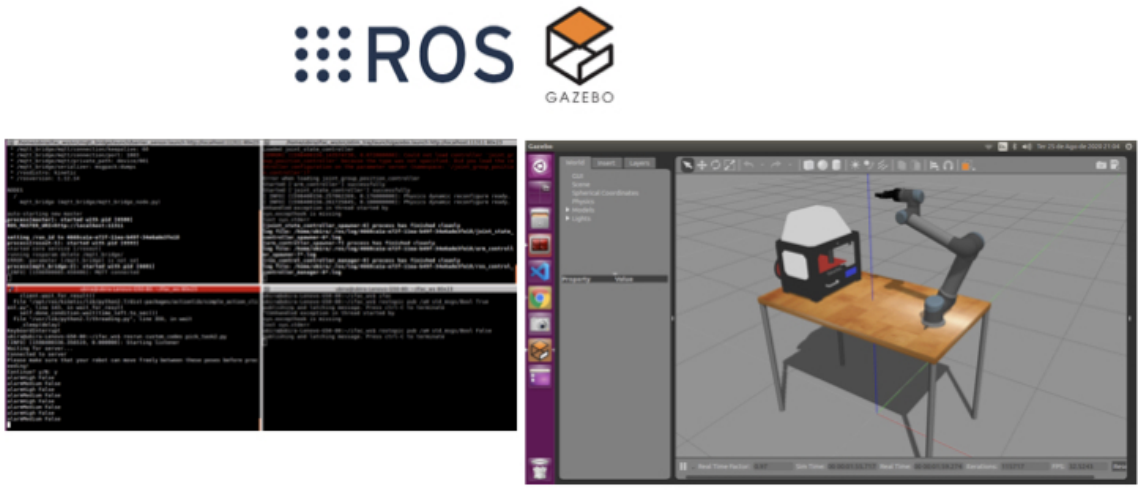


Figure 1.4: ROS and Gazebo interface example

1.4 MAIN CONTRIBUTIONS OF THIS WORK

Besides the present document, two main contributions of this work must be highlighted. The implementation of the APF path planning technique with solutions to the RLMP and GNRON local minima cases while avoiding obstacles in a 2D environtment with a planar manipulator using ROS and Gazebo.

The generalization of the previous work with APF path planning for a 6R manipulator (UR5) in the 3D space with collision avoidance, solution fo the RLMP and GNRON local minima cases and a orientation control technique with potential fields approach.

The results obtained during this work allowed the publication of the following scientific papers:

- Ubiratan De Melo Pinto Junior and Maria Paula Carvalho and André Gustavo Scolari Conceição, 2018, Campos Potenciais Artificiais Aplicados ao Planejamento de Trajetórias do Braço Robótico JACO. In: Congresso Brasileiro de Automática. DOI: doi://10.20906/CPS/CBA2018-0336;

- Ubiratan De Melo Pinto Junior and Caio Cristiano Barros Viturino and Andre Gustavo Scolari Conceição and Leizer Schnitman, 2020, Sistema Anticolisão Aplicado a Manipuladores Robóticos Baseado em Campos Potenciais Artificiais. In: Anais do 14º Simpósio Brasileiro de Automação Inteligente. Campinas : Galoá. 2019. DOI: 10.17648/sbai-2019-111278;

- Caio Cristiano Barros Viturino and Ubiratan De Melo Pinto Junior and Andre Gustavo Scolari Conceição and Leizer Schnitman, 2020, Adaptive Artificial Potential Fields with Orientation Control Applied to Robotic Manipulators. Accepted for

presentation in 21st International Federation of Automatic Control World Congress (IFAC 2020).

1.5 ORGANIZATION OF THIS WORK

The remainder of this work is organizes as follows. Chapter 2 presents the basis of the modeling techniques for robotics, including general movements description relative to reference frames. In Chapter 3, an overview on some path planning algorithms based on the Artificial Potential Fields concepts is carried out. 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

In this chapter the concepts of rigid motions, direct and inverse kinematics and path planning techniques for robotic manipulators.

2.1 RIGID MOTIONS

Let $o_0x_0y_0z_0$ coordinate system be fixed in space. A point p can be represented in this space in relation to the system $o_0x_0y_0z_0$ through its coordinates in the directions x, y and z of this axis. This representation is denoted by p^0

$$p^0 = ux_0 + vy_0 + wz_0 = \begin{bmatrix} u \\ v \\ z \end{bmatrix} \tag{2.1}$$

Likewise, the point p can be represented in space in relation to another system $o_1x_1y_1z_1$ by the dot product of each of its coordinates by the system versors x_1, y_1 and z_1. As follows:

$$p^1 = ux_0.x_1 + vx_0.x_1 + wx_0.x_1 = \begin{bmatrix} (ux_0 + vy_0 + wz_0).x_1 \\ (ux_0 + vy_0 + wz_0).y_1 \\ (ux_0 + vy_0 + wz_0).z_1 \end{bmatrix} \tag{2.2}$$

Similarly, the position of a $o_jx_jy_jz_j$ system can be represented in relation to another $o_ix_iy_iz_i$ system. For this, it is enough to know the coordinates of the source of $o_jx_jy_jz_j$, o_j, in relation to $o_ix_iy_iz_i$, that is, o_j^i. And the ratio of angles between the versors of the j system in relation to those of the i system. Therefore, to represent the position of any system, j, in relation to another reference system, i, it is enough to determine the relation of displacement, or translation, and rotation between them.

2.1.1 Three-dimensional Rotation Matrices

The rotation of a system $o_1x_1y_1z_1$ relative to a system $o_1x_1y_1z_1$ can be represented by a matrix, R_1^0. It represents the projection of the system 1 versors in the system 0 versor system versions.

$$R_1^0 = \begin{bmatrix} x_1.x_0 & y_1.x_0 & z_1.x_0 \\ x_1.y_0 & y_1.y_0 & z_1.y_0 \\ x_1.z_0 & y_1.z_0 & z_1.z_0 \end{bmatrix} \tag{2.3}$$

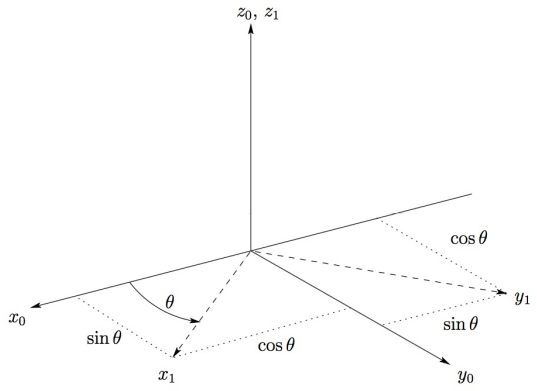


Figure 2.1: System $o_1x_1y_1z_1$ rotation of θ degrees with respect to the z_0 axis (SPONG; HUTCHINSON; VIDYASAGAR, 2006)

If, for example, the $o_1x_1y_1z_1$ system is rotated by θ degrees with respect to the z_0 axis, as in the figure 2.1, the matrix R_1^0, or $Rot_{z,\theta}$, will be given by:

$$R_1^0 = Rot_{z,\theta} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

For convenience, hereinafter the following shorthand notation will be adopted: $cos\theta = c_\theta$ and $sin\theta = s_\theta$.

Similarly, it is possible to determine the rotation matrices in relation to the axes x_0, $Rot_{x,\alpha}$, and y_0, $Rot_{y,\gamma}$.

$$Rot_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\alpha & -s_\alpha \\ 0 & s_\alpha & c_\alpha \end{bmatrix} \tag{2.5}$$

$$Rot_{y,\gamma} = \begin{bmatrix} c_\gamma & 0 & s_\gamma \\ 0 & 1 & 0 \\ -s_\gamma & 0 & c_\gamma \end{bmatrix} \tag{2.6}$$

It is also possible to compose rotation matrices. If, for example, a rotation of θ degrees with respect to the z_0 axis and then a rotation of α degrees with respect to the x_1 axis of the 1 coordinate system are made, the resulting rotation matrix, R, will be given by:

$$R = Rot_{z_0,\theta} Rot_{x_1,\alpha} \tag{2.7}$$

That is, by multiplying the next rotation matrix to the right of the first rotation matrix. However, if successive rotations are made in relation to the 0 system, or fixed, for example, a rotation of θ degrees with respect to the z_0 axis followed by another rotation of γ degrees with respect to the axis y_0, then you must multiply the second matrix on the left of the first. The R matrix is:

$$R = Rot_{y_0,\gamma} Rot_{z_0,\theta} \tag{2.8}$$

2.1.2 Three-dimensional Translation Matrices

A translation matrix, T or o_j^i, represents the coordinates of the origin of a j system in relation to a i system.

$$T = o_j^i = Trans_{x,d_x} + Trans_{y,d_y} + Trans_{z,d_z} \tag{2.9}$$

The translation composition is made by adding the translation matrices, like this:

$$T = o_j^i = \begin{bmatrix} d_x \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ d_y \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ d_z \end{bmatrix} \tag{2.10}$$

$$o_j^i = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \tag{2.11}$$

2.2 HOMOGENEOUS TRANSFORMATIONS

Homogeneous transformations are nothing more than compact forms of representation of rigid movements in just one matrix. A transformation matrix, H, homogeneous has the following general form.

$$H = \begin{bmatrix} R_{3x3} & T_{3x1} \\ f_{1x3} & s_{1x1} \end{bmatrix} \tag{2.12}$$

R being a three-dimensional rotation matrix, T a three-dimensional translation matrix, f the perspective and s the scale factor. For rigid motion applications $f_{1x3} = [0\ 0\ 0]$ and $s_{1x1} = [1]$. Variations in these matrices are used mainly in computer vision applications or computer simulations.

Thus, the H matrix is as follows:

$$H = \begin{bmatrix} n_x & s_x & a_x & d_x \\ n_y & s_y & a_y & d_y \\ n_z & s_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

The composition of homogeneous transformation matrices is given by the composition of the rotation and translation parts individually so that the H matrix is then assembled. That is, by properly multiplying the rotation matrices and adding the translation matrices in relation to the same system.

2.3 DIRECT KINEMATICS

A robotic manipulator can be represented by a set of links, or links, connected together through joints. These joints allow the relative movement between the links corresponding to their nature. Joints can be simple, like the revolutionary or prismatic ones, or they can be more complex like the ball and socket type. Revolution joints work as hinges and allow a rotation movement in relation to a certain axis. Prismatic joints allow displacement in relation to an axis, that is, extensions or retractions. Ball and socket joints are like the human shoulder and allow rotation in relation to two axes. The main difference between this type of joints is that the first two are joined with only one degree of freedom, while the last has two degrees of freedom.

While modeling robotic systems it is possible to consider only joints with a degree of freedom without loss of generality, since a ball and socket type joint can be modeled as two prismatic joints joined by a zero-length link. Thus, the performance of each joint can be represented by a real number, angle of rotation for a revolution joint and linear displacement value for a prismatic joint. This convention will be adopted from now on.

Thus, for each joint i a joint variable q_i will be associated, such that:

$$q_i = \begin{cases} \theta_i & , \quad if\ the\ joint\ i\ is\ of\ revolution \\ d_i & , \quad if\ the\ joint\ i\ is\ prismatic \end{cases} \tag{2.14}$$

For the kinematic analysis of a manipulator to be made, each i link is rigidly fixed with a $o_i x_i y_i z_i$ coordinate system so that if the i joint is actuated, the $o_i x_i y_i z_i$ system undergoes an equivalent movement, i.e. if the i joint is prismatic and causes a d_i offset, the system will undergo a d_i value translation on the joint action axis.

It is also interesting to fix a $o_0 x_0 y_0 z_0$ coordinate system to the base of the manipulator. This will be the inertial system and it is in relation to it that all movements resulting from the other systems will be expressed. The figure 2.2 shows an example of selecting systems rigidly attached to a robotic manipulator in a convenient way for kinematic analysis.

The last system, $o_3 x_3 y_3 z_3$, corresponds to the position of the manipulator's tool (a claw, or a welder for example) and is the position of greatest interest in the manipulator, since it must be controlled for any activity to be carried out.

Each i system is related to the previous system, $i-1$, through a homogeneous transformation, A_i, which represents the orientation of the i system in relation to the $i-1$ system. Therefore, the matrix A_i is a function of the joint variable i, q_i, and has the following form:

$$A_i = \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ 0 & 1 \end{bmatrix} \tag{2.15}$$

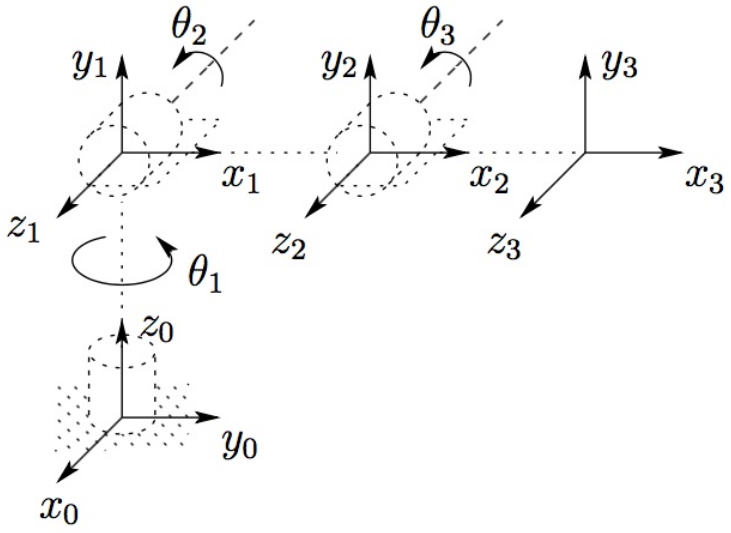


Figure 2.2: Example of coordinate systems rigidly attached to a robotic manipulator (SPONG; HUTCHINSON; VIDYASAGAR, 2006)

Where R_i^{i-1} and o_i^{i-1} represent the orientation and position of the $o_i x_i y_i z_i$ system origin in relation to the $o_{i-1} x_{i-1 system} y_{i-1} z_{i-1}$, respectively.

The homogeneous transformation that relates the orientation and position of a $o_j x_j y_j z_j$ system in relation to a $o_i x_i y_i z_i$ system is called *transformation matrix* and is represented as follows:

$$T_j^i = \begin{cases} A_{i+1} A_{i+2} \dots A_{j-1} A_j & , \quad if\ i < j \\ I & , \quad if\ i = j \\ (T_i^j)^{-1} & , \quad if\ i > j \end{cases} \tag{2.16}$$

Thus,

$$T_j^i = A_{i+1} A_{i+2} \dots A_{j-1} A_j = \begin{bmatrix} R_j^i & o_j^i \\ 0 & 1 \end{bmatrix} \tag{2.17}$$

Being the R_j^i matrix is given by:

$$R_j^i = R_{i+1}^i \dots R_j^{j-1} \tag{2.18}$$

And the coordinates vector o_j^i is given by:

$$o_j^i = o_{j-1}^i + R_{j-1}^i o_j^{j-1} \tag{2.19}$$

This way, the orientation and position of the tool of a robotic manipulator of n joints in relation to the inertial system $o_0 x_0 y_0 z_0$ is found by the following relation:

$$H = \begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix} \tag{2.20}$$

$$H = T_n^0 = A_1(q_1)A_2(q_2)\dots A_n(q_n) \tag{2.21}$$

Thus, to determine the position and orientation of a robotic manipulator tool, it is sufficient to know the values of the variables of each joint and its homogeneous transformation matrices, $A_i(q_i)$.

2.3.1 Denavit-Hartenberg Convention

In order to simplify and standardize the analysis of direct kinematics, it is common to use the Denavit-Hartenberg (DH) convention as a guide for choosing reference systems for each link of a manipulator. The kinematic analysis of a link manipulator can be extremely complex and this convention simplifies the analysis considerably, in addition to providing a universal language with which engineers in the field can communicate (SPONG; HUTCHINSON; VIDYASAGAR, 2006).

The DH convention evaluates four parameters for each link in order to determine its A_i matrix. These parameters are the length of the link, a_i, the twist of the link, α_i, the offset of the link, d_i, and the angle of the joint, θ_i. Thus, the matrix A_i is expressed by a rotation of θ_i on the z axis, followed by a translation of d_i on the current z axis, followed by a translation of a_i on the current axis x, followed by a rotation of α_i on the current x axis. That is:

$$A_i = Rot_{z,\theta_i}Trans_{z,d_i}Trans_{x,a_i}Rot_{x,\theta_i} \tag{2.22}$$

$$A_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

$$A_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_ic_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_is_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

Thus, the DH convention represents a homogeneous transformation with only four parameters while an arbitrary homogeneous transformation is completely represented by six parameters. This gives the DH parameter representation two degrees of freedom. So, to guarantee the existence and uniqueness of a DH representation once the reference systems are chosen, two considerations must be taken into account.

For two systems $o_0x_0y_0z_0$ and $o_1x_1y_1z_1$, the considerations are as follows:

1. The x_1 axis is perpendicular to the z_0 axis
2. The x_1 axis intercepts the z_0 axis

Examples of systems that obey these properties are shown in the figure 2.3. The DH parameters for the UR5 are presented in table 2.1 as described in the figure 2.4 from (ANDERSEN, 2018).

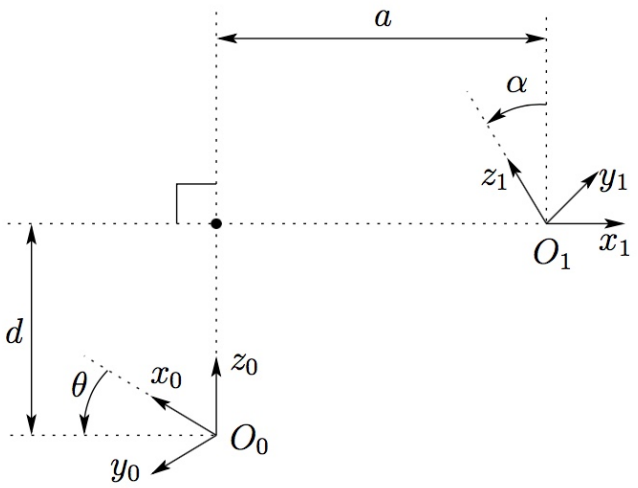


Figure 2.3: Coordinate systems that satisfy considerations 1 and 2 (SPONG; HUTCHINSON; VIDYASAGAR, 2006)

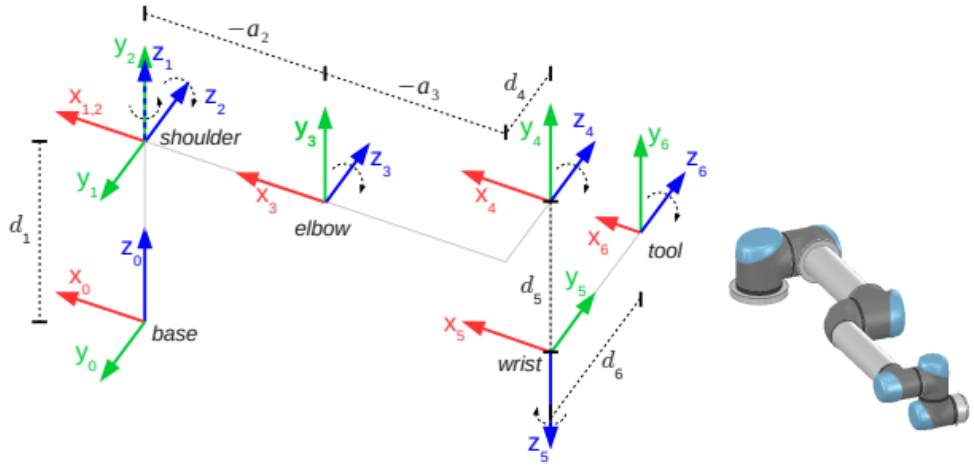


Figure 2.4: UR5 in the zero position (ANDERSEN, 2018)

For the UR5, the general transformation matrix between link $i-1$ and i is given by the following (ANDERSEN, 2018).

$$T_i^{i-1} = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & a_{i-1} \\ sin\theta_i cos(\alpha_{i-1}) & cos\theta_i cos(\alpha_{i-1}) & -sin(\alpha_{i-1}) & -sin(\alpha_{i-1})d_i \\ sin\theta_i sin(\alpha_{i-1}) & cos\theta_i sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.25}$$

i	α_{i-1}	a_{i-1}	d_i	θ_{i-1}
1	0	0	d_1	θ_1
2	90	0	0	θ_2
3	0	a_2	0	θ_3
4	0	a_3	d_4	θ_4
5	90	0	d_5	θ_5
6	-90	0	d_6	θ_6

Table 2.1: DH parameters for UR5

2.4 INVERSE KINEMATICS

The inverse kinematics problem is the opposite to that of direct kinematics. Its objective is, from the cartesian position and orientation of the manipulator tool, or final element, to find the values of the joint variables that satisfy these conditions.

This, however, is in general a more complex problem than that of direct kinematics as described (SPONG; HUTCHINSON; VIDYASAGAR, 2006), as it does not always present a single solution, that is, for a given position, more than a set of values of joint variables satisfy this condition, and the complexity of the problem increases according to the number of joints of each manipulator. In addition, such solutions cannot always be found analytically, requiring numerical and iterative methods for their resolution. This precludes a single method of description and solution to the problem, which requires that each analysis be considered on a case-by-case basis.

In (ANDERSEN, 2018), the method used to find the inverse kinematics for the UR5 was the geometric approach. It calculates the joint angles based on the desired pose of the final frame. The equations that determine the angle values are shown below.

$$\theta_1 = atan2(P^0_{5y}, P^0_{5x}) \pm acos(\frac{d_4}{\sqrt{(P^0_{5x})^2 + (P^0_{5y})^2}}) \tag{2.26}$$

$$\theta_5 = \pm acos(\frac{P^0_{6x} sin\theta_1 - P^0_{6y} cos\theta_1 - d_4}{d_6}) \tag{2.27}$$

$$\theta_6 = atan2(\frac{-X^6_{0y} sin\theta_1 + Y^6_{0y} cos\theta_1}{sin\theta_5}, \frac{X^6_{0y} sin\theta_1 - Y^6_{0y} cos\theta_1}{sin\theta_5}) \tag{2.28}$$

$$\theta_3 = \pm acos(\frac{\mid P^1_{4xz} \mid^2 - a_2^2 - a_3^2}{2a_2a_3}) \tag{2.29}$$

$$\theta_2 = atan2(-P^1_{4z}, -P^1_{4x}) - asin(\frac{-a_3 sin\theta_3}{\mid P^1_{4xz} \mid}) \tag{2.30}$$

$$\theta_4 = atan2(X^3_{4y}, X^3_{4x}) \tag{2.31}$$

Being the positive solution in equation 2.26 shoulder left, the negative solution shoulder right; the positive solution in equation 2.27 wrist up and the negative solution wrist

down; and the positive solution in equation 2.29 elbow up and the negative solution elbow down.

In the ROS environment there is a set of packages *universal-robots* (ROBOTS, 2019a), which already has implemented the inverse and direct kinematics of the UR5 manipulator. In this study this package will be used for the control of UR5.

Chapter 3

PATH PLANNING

In order to robotic manipulators to perform specific tasks while avoiding collisions with obstacles in the workspace, there are research aimed at the development of collision-free path planning algorithms.

In this context, the path planning method by potential artificial fields (KHATIB, 1986), enables high-level control taking advantage of the environment's feedback for real-time and interactive applications. For these characteristics and for having an intuitive concept of operation, this method is widely used in the path planning with diversion of obstacles. Therefore, it will be the method applied in this study.

The trajectory planning by potential artificial fields takes into account the arrangement in the cartesian space of the obstacles, the objective point and the effector, so that through these positions a potential field is generated that will cause forces of attraction and repulsion on the robot links and end effector whose result will define the path to be followed until the final position.

An example of a path planning algorithm for real-time application of an anti-collision system in manipulating and mobile robots, known as Artificial Potential Fields (APF), was firstly proposed by (KHATIB, 1986). With the use of APFs in manipulator robots, the links are seen as charged particles that undergo intervention of repulsive potential fields generated by obstacles and an attractive field generated by the final, or objective, position. Despite their efficiency, classical APFs have some restrictions or local minimum, such as:

- Inability to reach a desired end effector orientation in case of robot manipulators.
- Failure to achieve the goal when it is within the obstacle's influence area, a problem known as *Goals Non-Reachable with Obstacles Nearby* (GNRON) (GE; CUI, 2000). This problem occurs in mobile and manipulator robots.
- Non-convergence of the path in the positioning configurations where there are obstacles near the links. The obstacle's repulsive forces prevent the end effector

from reaching the goal, problem known as Reacharound Local Minimum Problem (RLMP) (BYRNE; NAEEM; FERGUSON, 2013). This problem occurs exclusively in robotic manipulators.

- The generated path need post-processing in order to make it smoother.

3.1 CLASSIC ARTIFICIAL POTENTIAL FIELDS

The philosophy of this method can be described as follows.

"The manipulator moves in a field of forces. The position to be reached is an attractive pole for the end effector and the obstacles are repulsive surfaces for the manipulator parts." (KHATIB, 1986)

The force field consists of the overlapping of an attractive field, $U_{att}(q)$, and a repulsive field, $U_{rep}(q)$.

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{3.1}$$

From this, the path planning can be solved in several ways. A simple algorithm for this is planning from the minimum gradient that defines the force acting on the tool as being.

$$F(q) = - \bigtriangledown U(q) = - \bigtriangledown U_{att}(q) - \bigtriangledown U_{rep}(q) \tag{3.2}$$

And is defined in the following way.

1. $q_{init} = q[0]$, $i = 0$
2. **IF** $|q[i] - q_{final}| < \delta$
   $q[i+1] = q[i] + \alpha \frac{F(q[i])}{|F(q[i])|}$
   $i = i + 1$
3. **ELSE** retorn $< q[0]$, $q[1]$, ... , $q[i] >$
4. **GO TO** 2

The value $q[i]$ is the position of the end effector in the iteration i. The value *alpha* is the size of the step that will be taken for each iteration. As in practice it is very unlikely that $q[i] = q_{final}$ it is necessary to admit a tolerance *delta* for the distance from the end effector to the final point for which it is considered that the objective has been achieved. The return values $< q[0]$ $q[1]$ ... $q[i] >$ are the path to be followed by the manipulator.

The attractive field can be defined as follows.

$$U_{att}(q) = \frac{1}{2} \zeta \rho_f^2(q) \tag{3.3}$$

$\rho_f^2(q)$ being the Euclidean distance between the current position of the tool, q, and the objective position, q_{final}, denoted by:

$$\rho_f(q) = ||q - q_{final}|| \tag{3.4}$$

The field has a quadratic characteristic, because for very small distances the attractive force tends to zero with a continuous derivative. However, for very long distances, this force can take on excessively high values, so it is interesting that after a certain distance d the field assumes a conical behavior and increases linearly with the distance. So, a more convenient way is.

$$U_{att}(q) = \begin{cases} \frac{1}{2}\zeta\rho_f^2(q) & , \quad se\ \rho_f(q) \leq d \\ \\ d\zeta\rho_f(q) - \frac{1}{2}\zeta d^2 & , \quad se\ \rho(q) > d \end{cases} \tag{3.5}$$

The attractive force at each point will have the absolute value, orientation and direction defined by the minimum gradient, as follows.

$$F_{att}(q) = - \bigtriangledown U_{att}(q) = \begin{cases} -\zeta(q - q_{final}) & , \quad se\ \rho_f(q) \leq d \\ \\ -\frac{d\zeta(q - q_{final})}{\rho_f(q)} & , \quad se\ \rho(q) > d \end{cases} \tag{3.6}$$

Interesting features for the repulsive field are: growing to infinity when as the robot approaches the obstacle and decreasing with increasing distance, $\rho(q)$, between them. In addition, in case there are several obstacles, it is interesting that each repulsive field has an influence distance, ρ_0, from which the field becomes null and does not influence the path of the robot. In this way the repulsive field can be defined as follows.

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho(q)} - \frac{1}{\rho_0})^2 & , \quad se\ \rho(q) \leq \rho_0 \\ \\ 0 & , \quad se\ \rho(q) > \rho_0 \end{cases} \tag{3.7}$$

The repulsive force at each point will have the absolute value, orientation and direction defined by the minimum gradient of the field as follows.

$$F_{rep}(q) = - \bigtriangledown U_{rep}(q) = \begin{cases} \eta(\frac{1}{\rho(q)} - \frac{1}{\rho_0})\frac{1}{\rho^2(q)} \bigtriangledown \rho(q) & , \quad se\ \rho_f(q) \leq \rho_0 \\ \\ 0 & , \quad se\ \rho(q) > \rho_0 \end{cases} \tag{3.8}$$

Where $\nabla\rho$ indicates the gradient $\nabla\rho(x)$ evaluated at $x = C_j(q)$. If a b point in the obstacle boundary in the workspace is close to the repulsive field of a control point in the robot, then $\nabla\rho = ||c_j(q) - b||$ and its gradient is represented by:

$$\nabla\rho(x)\big|_x = \frac{a_j(q) - b}{||a_j(q) - b||} \tag{3.9}$$

The forces acting on the robot are summed and applied to each joint i through the transposed Jacobian to obtain the necessary torque to move the joints. The total artificial joint torque acting on the arm is defined as:

$$\tau(q) = \sum_i J_i^T(q) F_{att,i}(q) + \sum_i J_i^T(q) F_{rep,i}(q) \tag{3.10}$$

3.2 ADAPTATIVE ARTIFICIAL POTENTIAL FIELDS

To eliminate GNRON problems present in classical APFs, (ZHANG; LIU; CHANG, 2017) developed the AAPF. The repulsive field of AAPFs is represented by

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta_j \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 \frac{\rho_g^n}{1+\rho_g^n} & ; \quad \rho \leq \rho_0 \\ 0 & ; \quad \rho > \rho_0 \end{cases} \tag{3.11}$$

where ρ_g is the distance from the joint to the goal and $n > 0$.

In (3.11), when $n = 1$ and the robot is far from the target, that is, $\rho_g^n \gg 1$, then $\rho_g^n/(1\rho_g^n) \approx 1$ and the repulsive fields of AAPFs are equivalent to the repulsive fields of APFs, avoiding the path from increasing as the distance to the target increases. When the robot is near the goal, i.e. $\rho_g^n \ll 1$, the expression $\rho_g^n/(1\rho_g^n)$ is equivalent to approximately ρ_g.

In AAPF, the repulsive force is fragmented into two other components, which draw the robot to the target and repel it from the obstacle even though it is positioned within the area of influence ρ_0, that is,

$$F_{rep}(q) = \begin{cases} F_{rep1}\ \gamma_{OR} + F_{rep2}\ \gamma_{RG} & ; \quad \rho \leq \rho_0 \\ 0 & ; \quad \rho > \rho_0 \end{cases} \tag{3.12}$$

where the unit vector $\gamma_{OR} = \nabla\rho(q, q_{obs})$ indicates the direction from the obstacle to the robot control point and $\gamma_{RG} = -\nabla\rho(q, q_{goal})$ indicates the robot's direction to the goal.

The F_{rep1} component, represented by:

$$F_{rep1} = \eta_j \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) \frac{\rho_g^n}{\rho^2(1+\rho_g^n)} \tag{3.13}$$

repels robot from the obstacle and the F_{rep2} component, represented by:

$$F_{rep2} = \frac{n}{2}\eta_j \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 \frac{\rho_g^{n-1}}{(1+\rho_g^n)^2} \tag{3.14}$$

draws the robot to the goal.

Figure 3.1 shows the attractive and repulsive AAPF force components. The total force generated from the AAPF guides the robot end effector to the goal, represented by the green circumference. For this method to work in real situations, the shape of the object must be filled with spheres representing repulsive fields, as shown in figure 4.2.

So that only the end effector, and not the other links, can reach a position within the obstacle's field of influence, AAPFs were applied only to the last link, while the other links remain using the APFs. This ensures that the other links do not enter the obstacle's influence area.

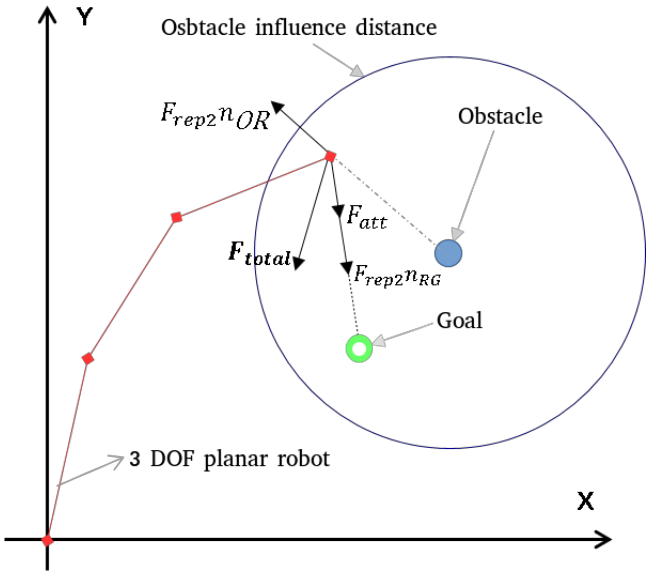


Figure 3.1: AAPF attractive and repulsive force components.

3.3 ORIENTATION CONTROL

The APF was also implemented in configuration space in order to control the orientation of the end effector through all the trajectory. The initial end effector orientation q, corresponding to Roll, Pitch and Yaw angles, is set equal to the grasping orientation q_{final} for the robot to keep current set orientation while it is moving. The attractive force in configuration space is given by

$$\begin{aligned} F_{att_\omega}(q) &= -\nabla U_{att}(q) \\ &= \begin{cases} -\zeta(q - q_{final}) & ; \quad \rho_f(q) \leq d \\ \dfrac{-d\zeta(q - q_{final})}{\rho_f(q)} & ; \quad \rho_f(q) > d \end{cases} \end{aligned} \tag{3.15}$$

where $\rho_f(q)$ is the distance from q to q_{final} and d is defined as the influence distance to the final orientation in radians. Fig. 3.2 shows the UR5 base link frame and the end effector frame. The axis x_i, y_i and z_i of the end effector frame f_{r_2} gets attracted by the axis x_{i-1}, y_{i-1} and z_{i-1} of the UR5 base link frame f_{r_1} in configuration space.

To implement the end effector orientation control, the Jacobian matrix was divided into linear Jacobian J_v (the submatrix formed by the three first rows of the Jacobian matrix) and angular Jacobian J_ω (the submatrix formed by the three last rows of the Jacobian matrix), each one having $3 \times c$ dimension (for this work, $c = 6$ and the control points are the UR5 joints), as described in (3.16).

$$J_{c\times c} = \begin{bmatrix} J_{v3\times c} \\ J_{\omega 3\times c} \end{bmatrix} \tag{3.16}$$

The attractive forces of each control point i in configuration space are transformed into joint torque through the transposed angular Jacobian $J^T_{\omega,i}$, where J_i is the Jacobian

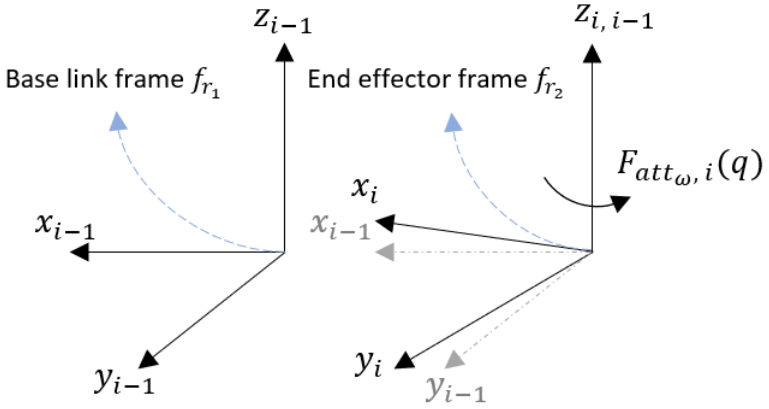


Figure 3.2: Reference frame of the UR5 base link and end effector.

matrix from the base frame to the i^{th} joint. The total artificial joint torque acting on the arm is defined as:

$$\begin{aligned}\tau(q) = F_{\omega,i}(q) + F_{v,i}(q) = \sum_i J^T_{\omega,i}(q) F_{att_\omega}(q) + \\ \sum_i J^T_{v,i}(q) F_{att,i}(q) + \sum_i J^T_{v,i}(q) F_{rep,i}(q)\end{aligned} \tag{3.17}$$

where, i represents the i^{th} joint. The force $F_{\omega,i}(q)$ is equivalent to the attractive force in configuration space imposed to the joints, so the end effector's orientation keeps constant, and $F_{v,i}(q)$ is the resulting force in the workspace.

Chapter 4

EXPERIMENTAL RESULTS

4.1 THE SETUP

All the experiments were made in a smart robotic additive manufacturing (SRAM) cell being made of an UR5 robot manipulator, from Universal Robots, and a 3D printer controlled by ROS. This cell can be used for several applications in the context of collaborative robotics and industry 4.0 such as, pick and place tasks in dynamic environments, interface with cloud services for information exchange and data logging, integration with discrete IOT devices for environment statuses (temperature, humidity, presence sensing etc), among other examples, as can be seen in (COSTA et al., 2020). The experiments made for this work are an approach task the a 3D printed part (goal) while avoiding collision with obstacles (3D printer) planned autonomously with previous knowledge of the environment and considering it static.

The devices of the cell are disposed as shown in figure 4.1. The UR5 is right in front of the 3D printer opening so parts can be more easily picked from the printer tray.

A virtual model of the cell is made and simulated on Gazebo software. This model is composed of CAD models of the UR5 and the 3D printer. But, as the proposed algorithm is a path planner with obstacle avoidance, the important information about the environment are: the position the robot or its joint states, the position of the goal and the position of the obstacles. As working with complex CAD models all the time have a high computational cost, all the input information for the algorithm can be simplified from the previous knowledge the the 3D printer and robot shapes and position. So, it is possible to model the goal, the 3D printed part, as a simple sphere. The obstacle, the 3D printer, can be simplified by the most probable parts that can be collided in this task: the front face and the tray. So the obstacle can be modelled as a set of 12 spheres of the same size placed equally spaced in the 3D printer front face and a bigger sphere representing the printer tray. The sphere representing the tray in involving the goal, so the attractive force from the goal won't drag the end-effector in the tray direction. The simplified model used as input for the presented algorithm can be seen in figure 4.2 side by side with the real cell image.

Figure 4.1: SRAM Cell

The AAPF method was applied only to the tray obstacle to avoid collisions with the outer parts of the printer. Thus the other obstacles repulsive forces stay the same as the classic APF. Similarly, to avoid collisions with the manipulator's links, the AAPF method was applied only to the grasping control point. Therefore, the other control points will never try to enter an obstacle influence area. These considerations make the technique application less likely to cause accidental collisions in the printer inner area.

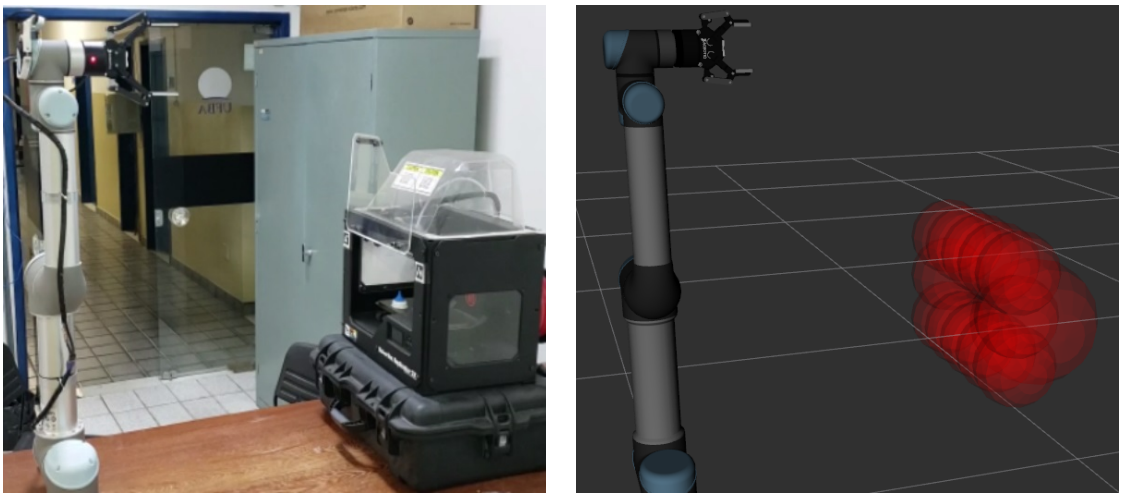
All the code used in this work in available at ⟨https://github.com/caiobarrosv/Project_IFAC_2020⟩ and the videos from the experiment can be seen at ⟨https://www.youtube.com/watch?v=9d7lBqTH_JA⟩.

4.2 THE OBJECTIVE OF THE PROPOSED ALGORITHM

As already mentioned, the experiments made for this work are approach activities i.e., they consist in the movement of the manipulator from a far position from the goal, where a typical pick task can't be performed, to a closer position to the goal, so it becomes possible. The pick task, not proposed in this work, can be computer vision assisted, using cameras, cloud point sensor or others, and it handles a much more shorter movements than the presented algorithm.

A pick task has the main objective to provide a reliable, stable and force precision grip with an specific end-effector tool so an object can be translated between two points, or oriented accordingly with an certain need. This implies in small and slow moves, in the order of centimeter, millimeters or even smaller magnitudes. Due to this required precision, to be executed, a pick task needs a set of sensors and a good initial position.

An approach task, on the other hand, is responsible to move the manipulator between two greater regions in its workspace. This movement generally don't need to be extremely precise, instead it can be more quick and wide, in the order of centimeters or meters, so



(a) Real cell (b) Virtual model

Figure 4.2: Experiment arrange showing (a) the UR5 manipulator, the 3D printer and (b) the simulated environment.

the robot can reach an area where another finer tuned control, but with a short range of action, can take place and be used by the application.

The proposed algorithm in this work lies in the second category. It is an approach algorithm that aims to autonomously move the manipulator from a Home position, far from the 3D printer, to a much closer position where another control can be used in a pick task. Besides simply move the robot, the algorithm also has the capability to avoid collision with obstacles in the environment with previously known position and set a goal orientation for the end-effector tool, so the tool can safely enter the printer and a computer vision assisted control can be easily applied in the grasping of the 3D printed part.

The algorithm works offline, that is, it calculates the path to be followed previously to the start of the movement with the knowledge of the environment and it considers no change will be made in the objects besides the robot. Then, the robot follows the path calculated until it reaches the last point. The stop condition for the path planning is the distance to the goal, it must be $3mm$ or less, or, as this method not always converge to a solution, a determined number of iterations, in this case, 1500 iterations.

4.3 THE EXPERIMENTS

For the three experiments that will be presented next (AAPF and APF with orientation control, and AAPF without orientation control), in all the cases the UR5 starts from the same position (the one shown in figure 4.2 a), it must reach the same goal (the part inside the 3D printer) and the environment has the same configuration all the time (as shown in figure 4.2 a and modelled as shown in figure 4.2 b). The initial end-effector orientation is also different from the final goal orientation. Additionally, as the APF algorithm has the objective to provide a solution for a path between two points, the picking task is not

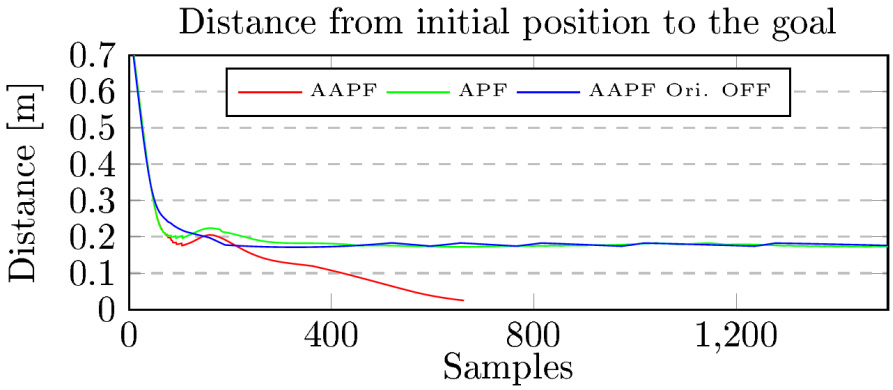


Figure 4.3: Comparison of the distance from UR5 end effector to the goal using APF and AAPF with orientation control and AAPF without orientation control

in the scope of this work, but only the approach to the goal task. Such tasks as pick and place are suitable to be solved by visual assisted algorithms for this purpose and not path planning algorithms. Following every experiment will be discussed in detail.

4.3.1 Distance to the Goal Comparison

In figure 4.3 is presented a plot of the distance of the end-effector tool to the goal for all the path planned by the three algorithms. The main stop condition for this path planning algorithm is the distance to the goal. If the minimal distance to the goal can be achieved and how fast it can be achieved is a measurement of the success of this method. It is possible to see that the AAPF method with orientation control is the only method that converges to the goal. The two other methods were not able to converge to a solution and stopped due to the number of iterations.

Even so the AAPF can enter the repulsive fields, the orientation at the moment of the approach plays an important role in the convergence of the algorithm. This is because the oppening in the 3D printer is so narrow that any orientation much different from the goal orientation will not allow the tool to safely enter the printer. And as the final orientation cannot be reliably predicted without the orientation control, the success of this method is then uncertain as it is susceptible to other local minimum problems than GNRON. So by not entering the 3D printer, the AAPF without orientation control doesn't differ from the APF in overall results. Added to that, the APF with orientation control proves to be inefficient too, once the inability to enter obstacles repulsive fields lets the end-effector outside the printer and it cannot reach the goal.

Another interesting characteristic that can be seen is that the beginning of all the paths is identical. More specifically, the path where the distance to the goal is $0.3m$ or greater. This is the obstacle-free portion of the path, when the manipulator is relatively far from the obstacles, so the resulting force of the AAPF method becomes equal to the one from the APF method, as expected. When the arm gets closer to the obstacles, the repulsive forces are different and so the resultant force. This way, the paths start to differ from each other.

4.3.2 Adaptative Artificial Potential Fields with Orientation Control

The figure 4.4 shows some sequential photos from the experiment in the real UR5 for the case of application of the AAPF with orientation control technique. From the images it is possible to see that the end effector manages to enter the interior of the 3D printer and reach the goal avoiding collisions with the printer front and interior sides.

By seeing the end-effector orientation, it can be noticed that the goal orientation was also achieved and it was gradually corrected during all the trajectory executed by the manipulator.

The analysis of figures 4.7 and 4.8 shows that the attractive and repulsive forces for all joints tends to zero at the end of the trajectory. This means that the net force making the joints move is zero and this indicates a minimum of the potential field, and the forces absolute value being null indicate that this minimum is a global minimum, thus the goal was reached.

The joints movement in figure 4.9 show the smoothness of the manipulator movement, so the robot had a good behaviour during all the trajectory and this is reflected in the forces graph too.

4.3.3 Artificial Potential Fields with Orientation Control

In similar way, figure 4.5 shows sequential images for the APF with orientation control experiment. Unlinke the previous case, it is possible to see that the end-effector cannot approach the goal. This is clearly seen in figures 4.7 and 4.8 as the repulsive force presents a persistent oscillatory behaviour. This happens because of the inability of the end-effector the penetrate obstacles influence area, exclusive feature of the AAPF method.

This continues to occur indefinitely, proving that the algorithm don't converge in this case. The number of iterations, roughly the double than the previous experiment, is another indicator of the fail to reach the goal and meet the first performance.

Although the end-effector orientation is gradually corrected during the trajectory, the goal pose is never reached as the joints goal is never approached.

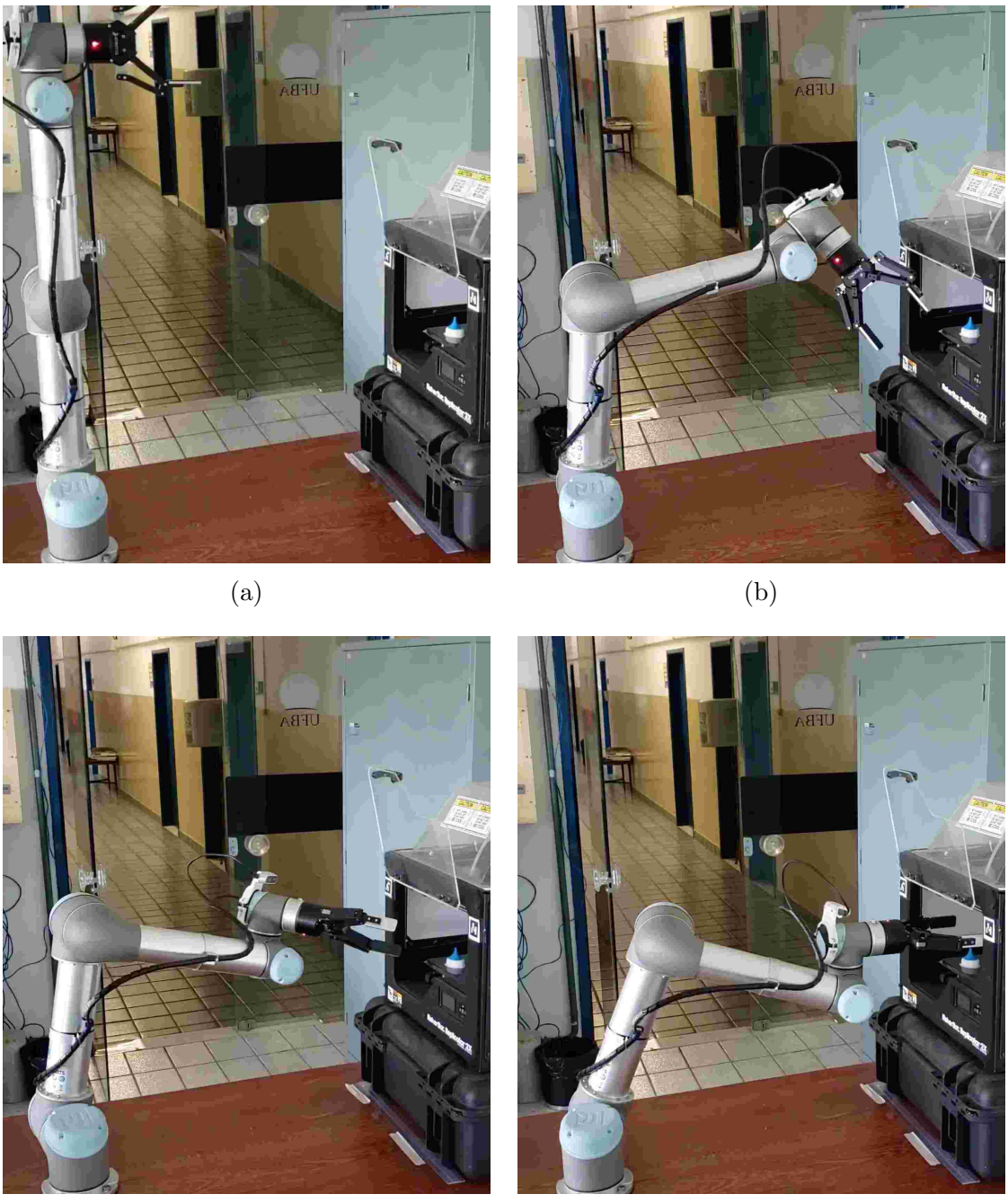
This experiments proves the importance of the AAPF ability to penetrate safely in obstacles influence areas in narrow environments.

4.3.4 Adaptative Artificial Potential Fields without Orientation Control

Finally, the figure 4.6 shows sequential images for the AAPF without orientation control experiment. Is easy to see that this was the worst case from all experiments.

Is possible to see that the lack of orientation control makes impossible to the AAPF to find and adequate approach pose for the task. In a case like this, a good approach orientation would only occur randomly. Without this orientation it can be seen form figures 4.7 and 4.8 that the force reach a stable state at non-zero absolute values, indicating that this is a local minimum and the arm will not progress anymore and can't reach the goal.

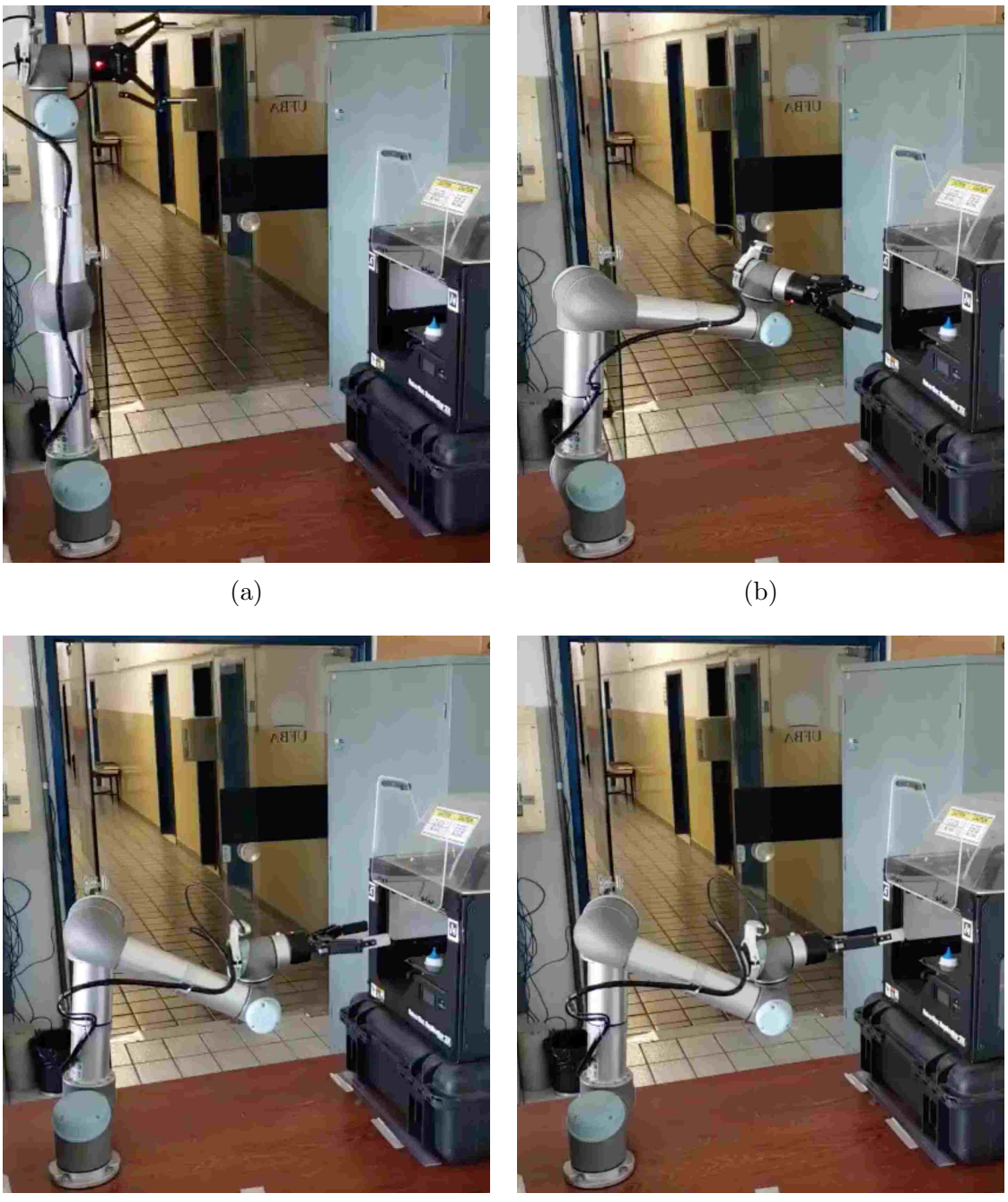
It's also clearly seen that the end-effector orientation stays unchanged during all the trajectory indicating the lack of orientation control. This proves the importance of the orientation control method for cases where the approach is constrained.



(a) (b)

(c) (d)

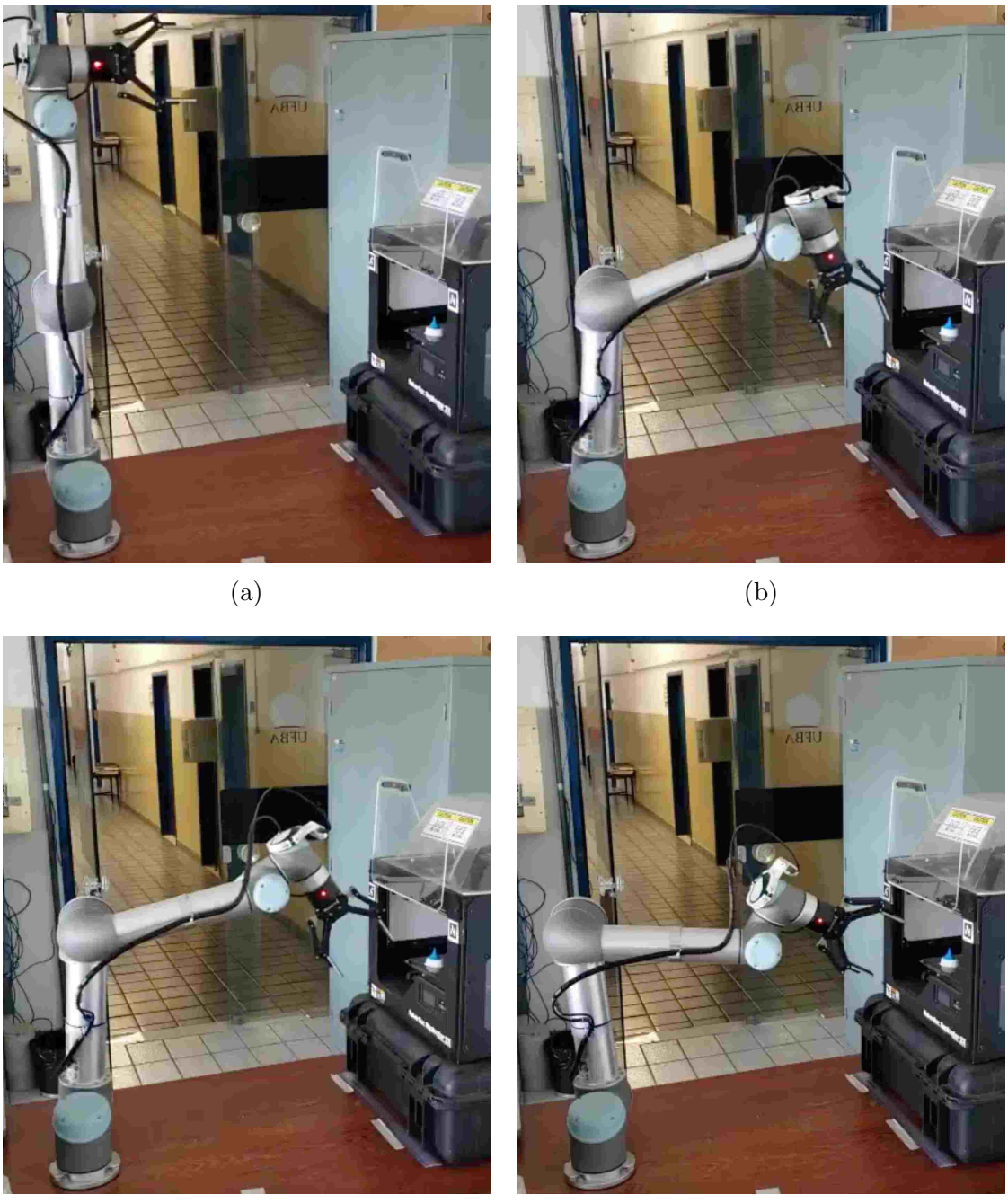
Figure 4.4: AAPF with orientation control method applied on real scenario



(a) (b)

(c) (d)

Figure 4.5: APF with orientation control method applied on a real scenario



(a) (b)

(c) (d)

Figure 4.6: AAPF without orientation control method applied on a real scenario

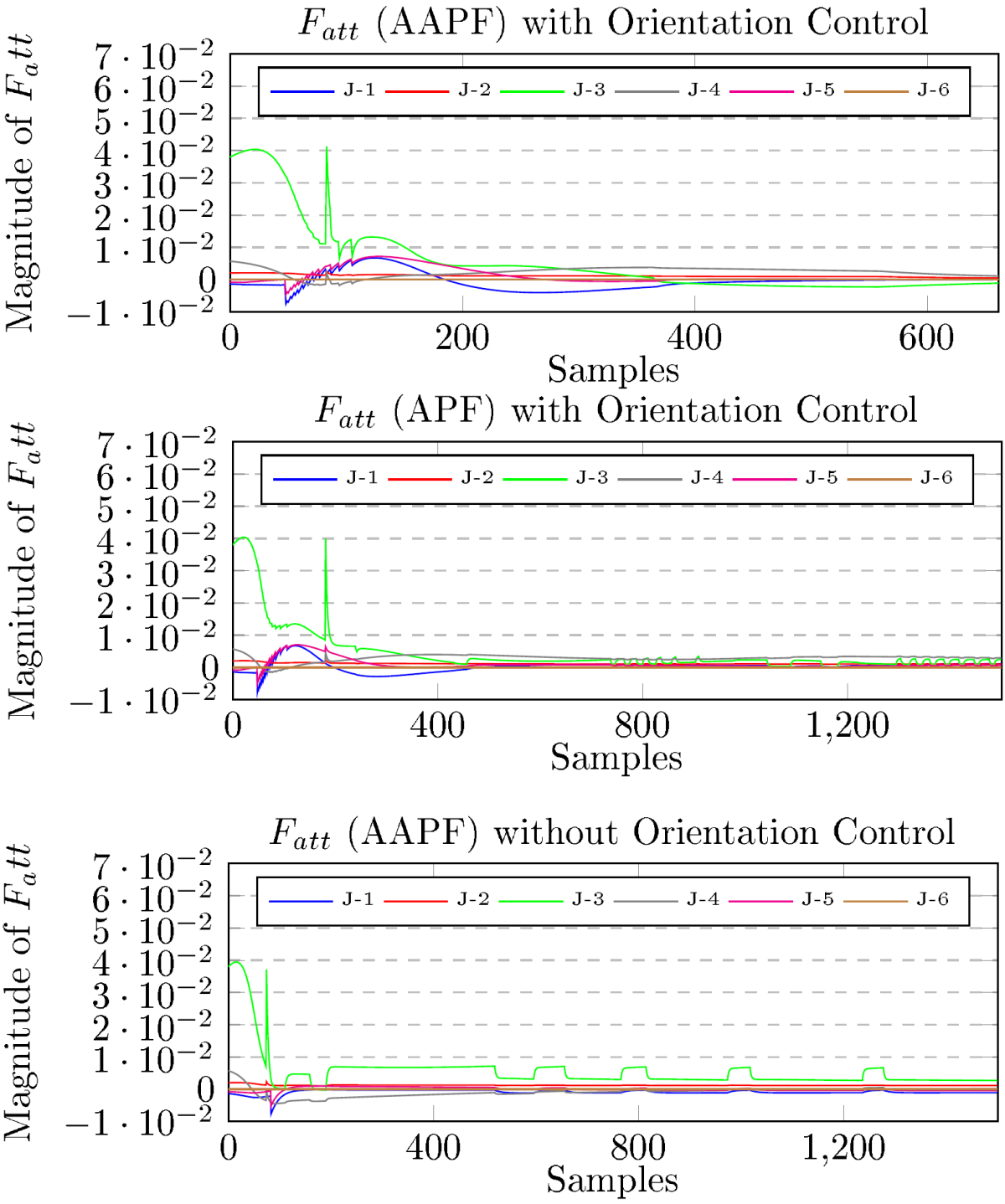


Figure 4.7: Comparison between attractive forces using AAPF and APF with orientation control and AAPF without orientation control

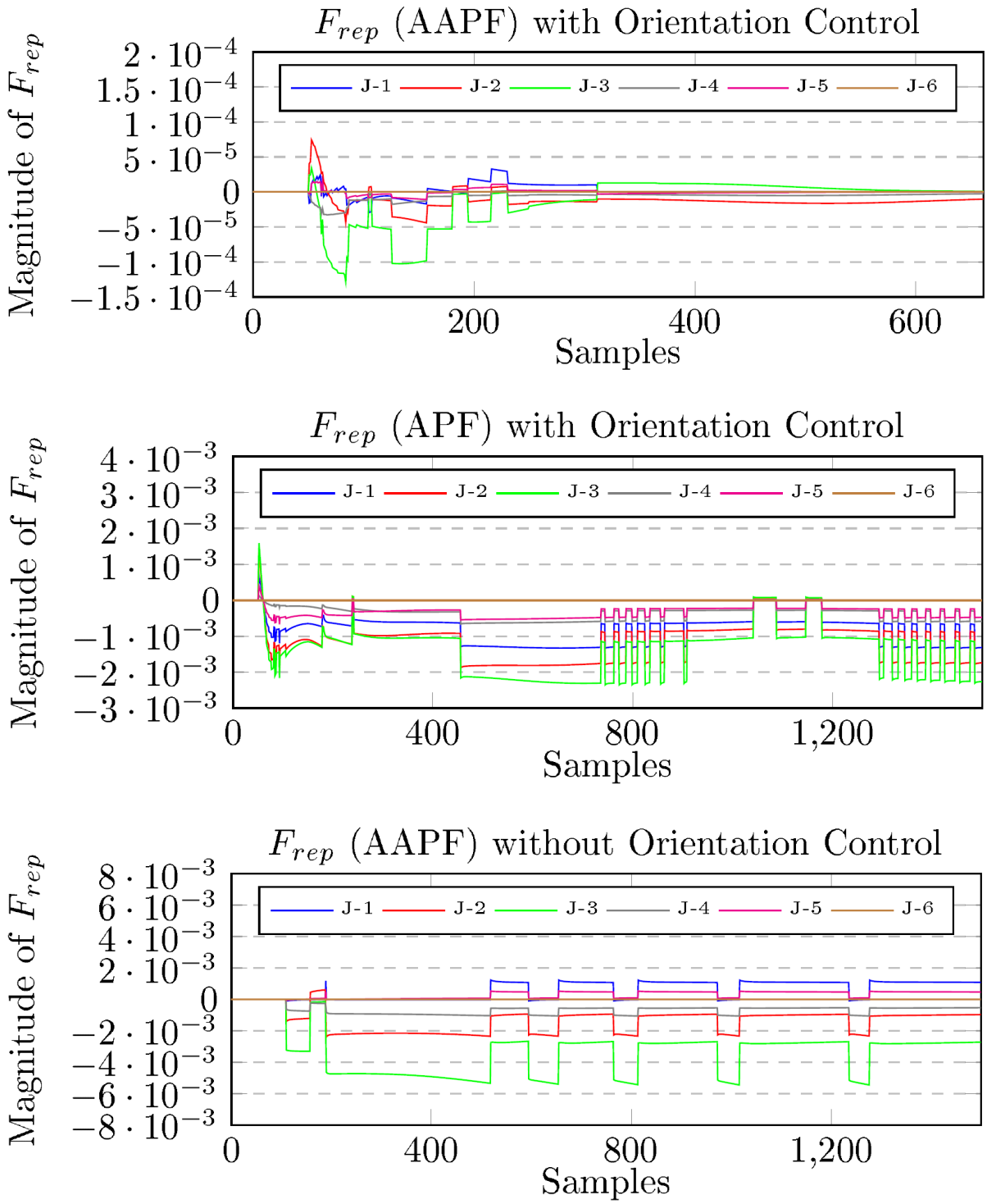


Figure 4.8: Comparison between repulsive forces using AAPF and APF with orientation control and AAPF without orientation control

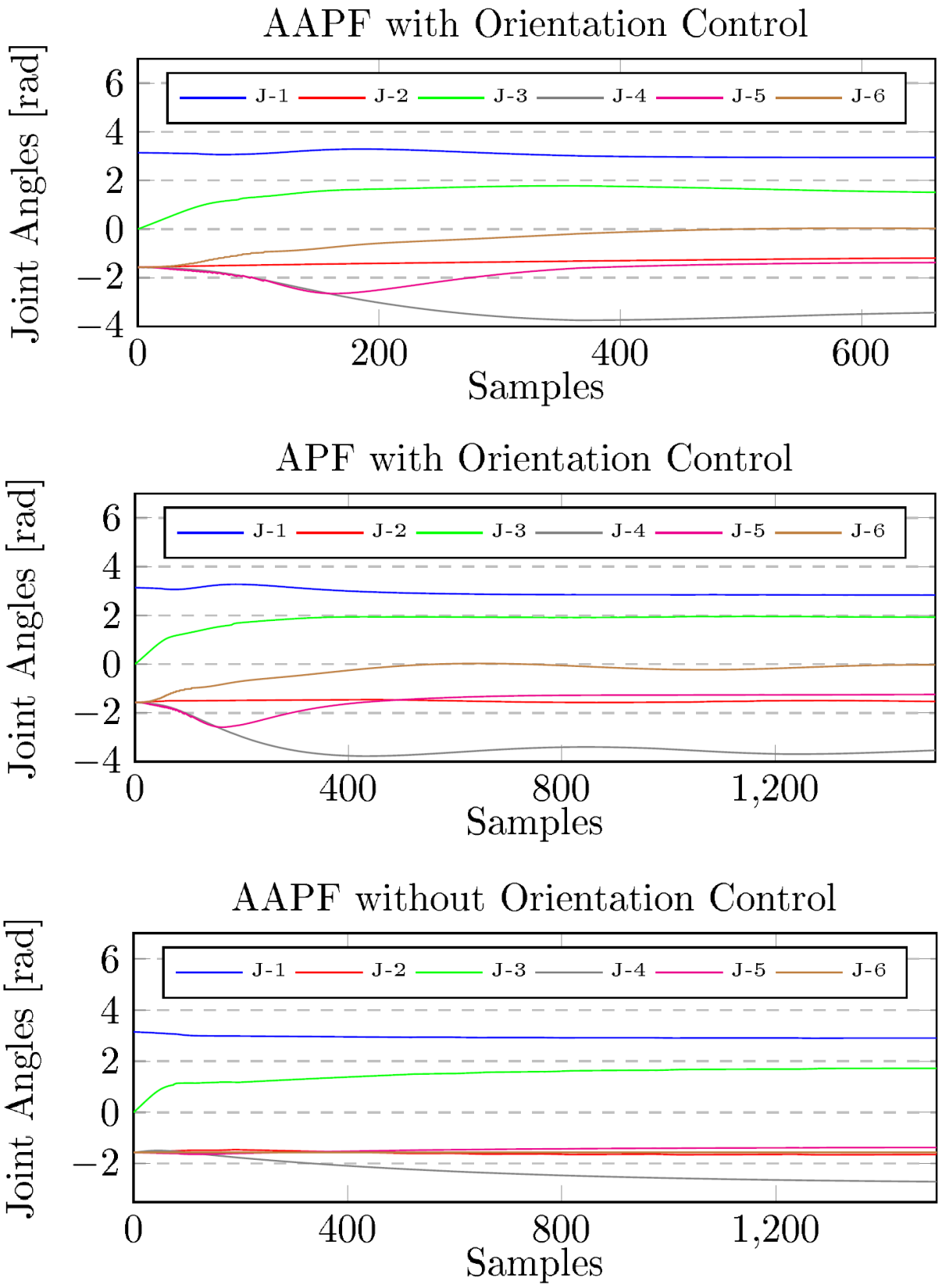


Figure 4.9: Comparison between UR5 joint angles using AAPF and APF with orientation control and AAPF without orientation control

Chapter 5

CONCLUSIONS

The collaborative robotics is a trend in the industry and, in this context, the environments in which the robots are being applied are increasingly more complex and dynamic. Thus manual and *ad hoc* solutions become not suitable for those scenarios and autonomous robots with real-time algorithms with fast response times becomes very important solutions. The methods proposed in this work are examples of this kind of algorithms that make possible the safe interaction among robots, humans and environment.

A novel method for end-effector orientation control which allows reaching a desired tool pose using the Jacobian matrix is presented. Results show that the desired final pose is achieved, but it only occurs in the final iterations of the planned path as the orientation is continually corrected along with all the other parts of the manipulator. This, in some cases, can cause problems to reach the final goal depending mainly in the initial robot position and type of area that must be reached. Prioritization of the tool orientation relative to the other joints can be a solution in some cases and also the intermediate pose goals selection that takes the tool to appropriate positions in some parts of the path in a way to ease the robot movement depending on the environment.

The typical local minimum problem GNRON is addressed and avoided by the implementation of the AAPF algorithm. It was shown that the use of this method reduces the jitter in narrow areas and allow the end-effector to reach the goal even if it is inside the obstacle influence area.

Even though the experimental results show positive results, the current method plans the path prior to the movement execution. Thus, the environment is assumed to be static. This way further improvements must be made so the robot can react to dynamic environments. This can be achieved by computer vision techniques for autonomous goals and obstacles recognition and modelling.

Also, the gain tuning can be an issue for this technique. Static gains were used for all the experiments made and they were determined empirically. Automatic and adaptative gain tuning can be applied so the performance can be improved. Other solutions to improve the APF planner performance is to use it with other techniques such as genetic algorithms, fuzzy logic and other.

BIBLIOGRAPHY

AKBARIPOUR, H.; MASEHIAN, E. Semi-lazy probabilistic roadmap: a parameter-tuned, resilient and robust path planning method for manipulator robots. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 89, n. 5-8, p. 1401–1430, 2017.

ANDERSEN, R. S. *Kinematics of a UR5.* 2018. ⟨http://rasmusan.blog.aau.dk/files/ur5_kinematics.pdf⟩. Accessed: 2020-12-09.

BYRNE, S.; NAEEM, W.; FERGUSON, S. An intelligent configuration-sampling based local motion planner for robotic manipulators. In: IEEE. *9th International Workshop on Robot Motion and Control*. [S.l.], 2013. p. 147–153.

COSTA, F. S.; NASSAR, S. M.; GUSMEROLI, S.; SCHULTZ, R.; CONCEIçãO, A. G. S.; XAVIER, M.; HESSEL, F.; DANTAS, M. A. R. Fasten iiot: An open real-time platform for vertical, horizontal and end-to-end integration. *Sensors*, v. 20, n. 19, 2020. ISSN 1424-8220. Disponível em: ⟨https://www.mdpi.com/1424-8220/20/19/5499⟩.

GE, S. S.; CUI, Y. J. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation*, IEEE, v. 16, n. 5, p. 615–620, 2000.

GUALTIERI, L.; RAUCH, E.; VIDONI, R. Emerging research fields in safety and ergonomics in industrial collaborative robotics: A systematic literature review. *Robotics and Computer-Integrated Manufacturing*, v. 67, p. 101998, 2021. ISSN 0736-5845. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S073658452030209X⟩.

HOLDOWSKY, J.; MAHTO, M.; RAYNOR, M.; COTTELEER, M. Inside the internet of things: A primer on the technologies building the iot. *Deloitte University Press*, 2015.

Isenberg, D. R. A potential field inspired approach to attitude motion planning with unit-quaternions. In: *SoutheastCon 2017*. [S.l.: s.n.], 2017. p. 1–7.

Kar, A. K.; Dhar, N. K.; Nawaz, S. S. F.; Chandola, R.; Verma, N. K. Automated guided vehicle navigation with obstacle avoidance in normal and guided environments. In: *2016 11th International Conference on Industrial and Information Systems (ICIIS)*. [S.l.: s.n.], 2016. p. 77–82.

KHATIB, O. Real-time obstacle avoidance for manipulators and mobile robots. In: *Autonomous robot vehicles*. [S.l.]: Springer, 1986. p. 396–404.

KIM, Y. H.; SON, W.-S.; PARK, J. B.; YOON, T. S. Smooth path planning by fusion of artificial potential field method and collision cone approach. In: EDP SCIENCES. *MATEC Web of Conferences*. [S.l.], 2016. v. 75, p. 05004.

LI, G.; TONG, S.; CONG, F.; YAMASHITA, A.; ASAMA, H. Improved artificial potential field-based simultaneous forward search method for robot path planning in complex environment. In: IEEE. *2015 IEEE/SICE International Symposium on System Integration (SII)*. [S.l.], 2015. p. 760–765.

LUO, J.; SU, W.; WANG, D. The improvement of the artificial potential field robot path planning based on 3-d space. In: IET. *International Conference on Automatic Control and Artificial Intelligence (ACAI 2012)*. [S.l.], 2012. p. 2128–2131.

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

PATLE, B.; Babu L, G.; PANDEY, A.; PARHI, D.; JAGADEESH, A. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, v. 15, n. 4, p. 582 – 606, 2019. ISSN 2214-9147. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S2214914718305130⟩.

Quiroz-Omaña, J. J.; Adorno, B. V. Whole-body control with (self) collision avoidance using vector field inequalities. *IEEE Robotics and Automation Letters*, v. 4, n. 4, p. 4048–4053, Oct 2019.

ROBOTS, U. *universal-robot ROS package*. 2019. ⟨https://github.com/ros-industrial/universal_robot⟩. Accessed: 2020-12-09.

ROBOTS, U. *UR5 collaborative robot arm — flexible and lightweight robot arm.* 2019. ⟨https://www.universal-robots.com/products/ur5-robot/⟩. Accessed: 2019-10-15.

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

SAFEEA M., B. R. . N. P. Collision avoidance of redundant robotic manipulators using newton's method. *J Intell Robot Syst*, v. 99, p. 673–681, 2020.

SINTOV, A. Goal state driven trajectory optimization. *Auton Robot*, v. 43, p. 631–648, 2019.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot modeling and control*. [S.l.]: wiley New York, 2006.

XIN, G.; QINGXUAN, J.; HANXU, S.; GANG, C.; QIANRU, Z.; YUKUN, Y. Real-time dynamic system to path tracking and collision avoidance for redundant robotic arms. *The Journal of China Universities of Posts and Telecommunications*, v. 23, n. 1, p. 73 – 96, 2016. ISSN 1005-8885. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1005888516600110⟩.

Zahroof, T.; Bylard, A.; Shageer, H.; Pavone, M. Perception-constrained robot manipulator planning for satellite servicing. In: *2019 IEEE Aerospace Conference*. [S.l.: s.n.], 2019. p. 1–10.

Zhang, H.; Wang, Y.; Zheng, J.; Yu, J. Path planning of industrial robot based on improved rrt algorithm in complex environments. *IEEE Access*, v. 6, p. 53296–53306, 2018.

ZHANG, Y.; LIU, Z.; CHANG, L. A new adaptive artificial potential field and rolling window method for mobile robot path planning. In: IEEE. *2017 29th Chinese Control And Decision Conference (CCDC)*. [S.l.], 2017. p. 7144–7148.

APPENDIX A

MAIN CODE

```
  #!/usr/bin/env python

# ROS import
import sys
import rospy
import tf
from tf.transformations import quaternion_from_euler, euler_from_matrix,
    euler_from_quaternion #, quaternion_matrix
from tf import TransformListener, TransformerROS
import actionlib

# Moveit Import
import moveit_commander
from moveit_python import *
# from moveit_commander.conversions import pose_to_list

# Msg Import
from moveit_msgs.msg import *
# from geometry_msgs.msg import *
from geometry_msgs.msg import PoseStamped, Point, Vector3, Pose
from std_msgs.msg import String, Header, ColorRGBA, Float64
from visualization_msgs.msg import Marker
from shape_msgs.msg import SolidPrimitive
from sensor_msgs.msg import JointState
from control_msgs.msg import *
from trajectory_msgs.msg import *

# Inverse Kinematics Import
from ur_inverse_kinematics import *

# Python Import
import numpy as np
from numpy import array, dot, pi
from numpy.linalg import det, norm
import csv
import argparse
import time
from itertools import izip_longest
import matplotlib.pyplot as plt
```

```
# Customized code
from get_geometric_jacobian import *
from get_ur5_position import *
from get_dist3D import *
import CPA
from CPA import *
from smooth_path import *

def get_param(name, value=None):
    private = "~%s" % name
    if rospy.has_param(private):
        return rospy.get_param(private)
    elif rospy.has_param(name):
        return rospy.get_param(name)
    else:
        return value

"""
    PARSE ARGS
"""

def parse_args():
    parser = argparse.ArgumentParser(description='AAPF_Orientation')
    # store_false assumes that variable is already true and is only set to
        false if is given in command terminal
    parser.add_argument('--AAPF', action='store_true', help='Choose AAPF
        instead of APF')
    parser.add_argument('--APF', action='store_true', help='Choose APF instead
        of AAPF')
    parser.add_argument('--OriON', action='store_true', help='Activate
        Orientation Control')
    parser.add_argument('--COMP', action='store_true', help='Compares distance
        to goal using APF, AAPF w/ and without ori control')
    parser.add_argument('--CSV', action='store_true', help='Write topics into
        a CSV file')
    parser.add_argument('--plot', action='store_true', help='Plot path to RVIZ
        through publish_trajectory.py (run this node first)')
    parser.add_argument('--plotPath', action='store_true', help='Plot path to
        RVIZ through publish_trajectory.py (run this node first)')
    parser.add_argument('--realUR5', action='store_true', help='Enable real
        UR5 controlling')
    args = parser.parse_args()
    return args

def print_joint_angles(args, vec, file, type):
```

```
    with
        open('/home/caio/3_Projetos_UR5/Project_IFAC_2019/src/custom_codes/csv_files/'
        + file + '.csv', mode='w') as employee_file:
        employee_writer = csv.writer(employee_file, delimiter=' ',
            quotechar='"', quoting=csv.QUOTE_MINIMAL)
        if type == 'joint':
            employee_writer.writerow(['Index', 'Joint1', 'Joint2', 'Joint3',
                'Joint4', 'Joint5', 'Joint6'])
            n = 0
            for position in vec:
                employee_writer.writerow([n, position[0], position[1],
                    position[2], position[3], position[4], position[5]])
                n+=1
        elif type == 'dist_to_goal' and args.COMP:
            employee_writer.writerow(['Index', 'dist_to_goal_AAPF_OriON',
                'dist_to_goal_APF_OriON', 'dist_to_goal_AAPF_OriOFF'])
            new_vec1 = [x[0] for x in vec[0]]
            new_vec2 = [x[0] for x in vec[1]]
            new_vec3 = [x[0] for x in vec[2]]
            new_index = [x for x in range(max(len(vec[0]),len(vec[1])))]
            rows = izip_longest(new_index, new_vec1, new_vec2, new_vec3,
                fillvalue='nan')
            employee_writer.writerows(rows)
        elif type == 'dist_to_goal':
            employee_writer.writerow(['Index', 'dist_to_goal'])
            n = 0
            for dist in vec:
                employee_writer.writerow([n, dist[0]])
                n+=1
        elif type == 'euler_angles':
            employee_writer.writerow(['Index', 'Roll', 'Pitch', 'Yaw'])
            roll, pitch, yaw = zip(*vec)
            new_index = [x for x in range(len(roll))]
            rows = izip_longest(new_index, roll, pitch, yaw, fillvalue='nan')
            employee_writer.writerows(rows)

def print_output(n, way_points, wayPointsSmoothed, dist_EOF_to_Goal):
    print("Dados dos CPAAs")
    print("Iterations: ", n)
    print("Way points: ", len(way_points))
    print("Way points smoothed: ", len(wayPointsSmoothed))
    print("Distance to goal: ", dist_EOF_to_Goal)

class MoveGroupPythonIntefaceTutorial(object):
    """MoveGroupPythonIntefaceTutorial"""

    def __init__(self, args):
```

```
        super(MoveGroupPythonIntefaceTutorial, self).__init__()

        moveit_commander.roscpp_initialize(sys.argv)
        rospy.init_node('move_group_python_interface_ur5_robot',
            anonymous=True)

        # self.display_trajectory_publisher =
            rospy.Publisher('/move_group/display_planned_path',
        #
            moveit_msgs.msg.DisplayTrajectory,
        #                                                 queue_size=20)

        self.pose = PoseStamped()
        # publish path or trajectory to publish_trajectory.py
        self.pose_publisher = rospy.Publisher('pose_publisher_tp',
            PoseStamped, queue_size=10)

        # Topico para publicar marcadores para o Rviz
        self.marker_publisher = rospy.Publisher('visualization_marker2',
            Marker, queue_size=100)

        self.tf = tf.TransformListener()
        self.scene = PlanningSceneInterface("base_link")
        self.marker = Marker()
        self.joint_states = JointState()
        self.joint_states.name = ['shoulder_pan_joint', 'shoulder_lift_joint',
            'elbow_joint', 'wrist_1_joint', 'wrist_2_joint',
                                'wrist_3_joint']

        # d1, S0, E0, a2, a3, d4, d45, d5, d6
        self.ur5_param = (0.089159, 0.13585, -0.1197, 0.425,
                        0.39225, 0.10915, 0.093, 0.09465, 0.0823 + 0.15)

        # Plot path inside while loop
        self.plot_path = True

        self.n = 1
        self.id = 100
        self.id2 = 130
        self.id_path = 200
        self.path_color = ColorRGBA(0.0, 0.0, 0.1, 0.8)
        self.diam_goal = [0.05]
        self.ptFinal = [[0.55, 0.0, 0.55]]

        self.client =
            actionlib.SimpleActionClient('arm_controller/follow_joint_trajectory',
            FollowJointTrajectoryAction)
```

```
        print "Waiting for server (gazebo)..."
        self.client.wait_for_server()
        print "Connected to server (gazebo)"

        # Obstacle positions
        oc = [-0.86, 0, 0.375] # Obstacle reference point - 3D printer
        d1 = -0.080
        s = 1
        self.obs_pos = [oc, np.add(oc, [s * 0.14, 0.0925, 0.255 + d1]),
            np.add(oc, [s * 0.14, 0.185, 0.255 + d1]), np.add(oc, [s * 0.14, 0,
            0.255 + d1]), np.add(oc, [s * 0.14, -0.0925, 0.255 + d1]),
            np.add(oc, [s * 0.14, -0.185, 0.255 + d1]),
                  np.add(oc, [s * 0.14, -0.185, 0.16 + d1]), np.add(oc, [s *
                      0.14, 0.185, 0.16 + d1]), np.add(oc, [s * 0.14, 0.0925,
                      0.05 + d1]), np.add(oc, [s * 0.14, 0.185, 0.05 + d1]),
                      np.add(oc, [s * 0.14, 0, 0.05 + d1]),
                  np.add(oc, [s * 0.14, -0.0925, 0.05 + d1]), np.add(oc, [s *
                      0.14, -0.185, 0.05 + d1])]
        self.diam_obs = [0.18] * len(self.obs_pos) # Main obstacle repulsive
            field
        self.diam_obs[0] = 0.3
        self.add_sphere(self.obs_pos, self.diam_obs, ColorRGBA(1.0, 0.0, 0.0,
            0.3))

        # CPA PARAMETERS
        self.eta = [0.00001 for i in range(6)] # Repulsive gain of each
            obstacle default: 0.00006

        if args.realUR5:
            self.clientreal =
                actionlib.SimpleActionClient('follow_joint_trajectory',
                FollowJointTrajectoryAction)
            print "Waiting for server (real)..."
            self.clientreal.wait_for_server()
            print "Connected to server (real)"

    """
    TH Matrix from joint angles
    """

    def matrix_from_joint_angles(self):
            th1, th2, th3, th4, th5, th6 = self.joint_states.position
            d1, S0, E0, a2, a3, d4, d45, d5, d6 = self.ur5_param

            matrix = [[-(sin(th1)*sin(th5) + cos(th1)*cos(th5)*cos(th2 + th3 +
                th4))*cos(th6) + sin(th6)*sin(th2 + th3 + th4)*cos(th1),
                (sin(th1)*sin(th5) + cos(th1)*cos(th5)*cos(th2 + th3 +
```

```
            th4))*sin(th6) + sin(th2 + th3 + th4)*cos(th1)*cos(th6),
            -sin(th1)*cos(th5) + sin(th5)*cos(th1)*cos(th2 + th3 + th4),
            -E0*sin(th1) - S0*sin(th1) + a2*cos(th1)*cos(th2) +
            a3*cos(th1)*cos(th2 + th3) - d45*sin(th1) - d5*sin(th2 + th3 +
            th4)*cos(th1) - d6*(sin(th1)*cos(th5) -
            sin(th5)*cos(th1)*cos(th2 + th3 + th4))],
            [-(sin(th1)*cos(th5)*cos(th2 + th3 + th4) -
            sin(th5)*cos(th1))*cos(th6) + sin(th1)*sin(th6)*sin(th2 + th3 +
            th4), (sin(th1)*cos(th5)*cos(th2 + th3 + th4) -
            sin(th5)*cos(th1))*sin(th6) + sin(th1)*sin(th2 + th3 +
            th4)*cos(th6), sin(th1)*sin(th5)*cos(th2 + th3 + th4) +
            cos(th1)*cos(th5), E0*cos(th1) + S0*cos(th1) +
            a2*sin(th1)*cos(th2) + a3*sin(th1)*cos(th2 + th3) +
            d45*cos(th1) - d5*sin(th1)*sin(th2 + th3 + th4) +
            d6*(sin(th1)*sin(th5)*cos(th2 + th3 + th4) +
            cos(th1)*cos(th5))], [sin(th6)*cos(th2 + th3 + th4) + sin(th2 +
            th3 + th4)*cos(th5)*cos(th6), -sin(th6)*sin(th2 + th3 +
            th4)*cos(th5) + cos(th6)*cos(th2 + th3 + th4),
            -sin(th5)*sin(th2 + th3 + th4), -a2*sin(th2) - a3*sin(th2 +
            th3) + d1 - d5*cos(th2 + th3 + th4) - d6*sin(th5)*sin(th2 + th3
            + th4)], [0, 0, 0, 1]]
        return matrix

"""
Also gets each frame position through lookupTransform
"""

def get_repulsive_cp(self, obs_pos, joint_values, CP_ur5_rep):
    """
    Get control point positions
    :param CP_ur5_rep: repulsive fields diameter on UR5
    :param joint_values: joint angles
    :param obs_pos: position of each obstacle
    :return: Control point positions and dist from each control point to
        each osbtacle
    """

    marker_lines = Marker()

    ur5_links = [
        "upper_arm_link",
        "forearm_link",
        "wrist_1_link",
        "wrist_2_link",
        "wrist_3_link",
        "grasping_link"
    ]
```

```
    CP_pos, CP_dist = [], []

    for i in range(len(ur5_links)):
        link_position = get_ur5_position(self.ur5_param, joint_values,
            ur5_links[i])
        CP_pos.append(link_position)
        CP_inter = []
        # self.add_sphere2(link_position, CP_ur5_rep[i], ColorRGBA(1.0,
            0.0, 0.0, 0.2)) # Plot UR5 repulsive fields
        # Calculates distance from link position to each obstacle and
            store it in CP_dist
        for y in range(len(obs_pos)):
            CP_inter.append(np.linalg.norm(link_position - obs_pos[y]))
        CP_dist.append(CP_inter)
    return CP_pos, CP_dist

"""
Adds the obstacles and repulsive control points on the robot
"""

def add_sphere(self, pose, diam, color):
    marker = Marker()
    marker.header.frame_id = "base_link"
    for i in range(len(pose)):
        marker.id = self.id
        marker.pose.position = Point(pose[i][0], pose[i][1], pose[i][2])
        marker.type = marker.SPHERE
        marker.action = marker.ADD
        marker.scale = Vector3(diam[i], diam[i], diam[i])
        marker.color = color
        self.marker_publisher.publish(marker)
        self.id += 1

"""
This function plot UR5 Repulsive Fields
"""

def add_sphere2(self, pose, diam, color):
    marker = Marker()
    if self.id2 == 137:
        self.id2 = 130
    marker.header.frame_id = "base_link"
    marker.id = self.id2
    marker.pose.position = Point(pose[0], pose[1], pose[2])
    marker.type = marker.SPHERE
    marker.action = marker.MODIFY
```

```
        marker.scale = Vector3(diam, diam, diam)
        marker.color = color
        self.marker_publisher.publish(marker)
        self.id2 += 1

    """
    Add Cylinder pose
    """
    def add_obstacles(self, name, height, radius, pose, orientation, r, g, b):
        scene = self.scene
        # addCylinder (self, name, height, radius, x, y, z, use_service=True)
        s = SolidPrimitive()
        s.dimensions = [height, radius] # [height, radius]
        s.type = s.CYLINDER

        ps = Pose()
        # ps.header.frame_id = "cylinder1"
        ps.position.x = pose[0]
        ps.position.y = pose[1]
        ps.position.z = pose[2]
        x, y, z, w = quaternion_from_euler(orientation[0], orientation[1],
            orientation[2])
        ps.orientation.x = x
        ps.orientation.y = y
        ps.orientation.z = z
        ps.orientation.w = w
        scene.addSolidPrimitive(name, s, ps)
        scene.setColor(name, r, g, b)

    """
    Plot robot's path to the RViz environment
    """

    def visualize_path_planned(self, path, G = 1.0, B = 0.0):
        self.marker.points.append(Point(path[0], path[1], path[2]))
        self.marker.header.frame_id = "base_link"
        self.marker.id = self.id_path
        self.marker.type = self.marker.LINE_STRIP
        self.marker.action = self.marker.ADD
        self.marker.scale = Vector3(0.008, 0.009, 0.1)
        self.marker.color = self.path_color
        self.marker_publisher.publish(self.marker)

    """
    Send final trajectory to gazebo or real UR5
    """
```

```
def move(self, way_points, target):
    g = FollowJointTrajectoryGoal()
    g.trajectory = JointTrajectory()
    g.trajectory.joint_names = self.joint_states.name

    # for joint in range(len(way_points)):
    hz = get_param("rate", 30) # 10hz
    r = rospy.Rate(hz)

    try:
        i = 0
        while not rospy.is_shutdown() and i < len(way_points):
            g.trajectory.points.append(JointTrajectoryPoint(positions=way_points[i],
                                                            velocities=[0] * 6,
                                                            time_from_start=rospy.Duration(0.05
                                                                * i + 5)))
                                                                #default 0.1*i +
                                                                5
            i += 1

        if target == "gazebo":
            self.client.send_goal(g)
            self.client.wait_for_result()
        elif target == "real":
            self.clientreal.send_goal(g)
            self.clientreal.wait_for_result()

    except KeyboardInterrupt:
        self.client.cancel_goal()
        self.clientreal.cancel_goal()
        raise
    except:
        raise

def CPA_loop(self, args, way_points, R, P, Y, AAPF_COMP = False, ORI_COMP
    = False):


    '# add_obstacles(name, height, radius, pose, orientation, r, g, b):'
    # self.add_obstacles("up", 0.54, 0.09, [-0.76, 0, 0.345], [1.5707,
        1.5707, 0], 1, 0, 0)
    # self.add_obstacles("bottom", 0.54, 0.09, [-0.76, 0, 0.55], [1.5707,
        1.5707, 0], 1, 0, 0)
    # self.add_obstacles("right", 0.35, 0.09, [-0.76, 0.185, 0.455], [0,
        0, 0], 1, 0, 0)
    # self.add_obstacles("left", 0.35, 0.09, [-0.76, -0.185, 0.455], [0,
        0, 0], 1, 0, 0)
```

```
self.add_sphere(self.ptFinal, self.diam_goal, ColorRGBA(0.0, 1.0, 0.0,
    1.0))

# apply obstacle colors to moveit
self.scene.sendColors()

# Final position
Displacement = [0.01, 0.01, 0.01]

# CPA Parameters
err = self.diam_goal[0] / 2 # Max error allowed
max_iter = 1500 # Max iterations
zeta = [0.5 for i in range(7)] # Attractive force gain of each obstacle
rho_0 = [i / 2 for i in self.diam_obs] # Influence distance of each
    obstacle
dist_att = 0.05 # Influence distance in workspace
dist_att_config = 0.2 # Influence distance in configuration space
alfa = 0.5 # Grad step of positioning - Default: 0.5
alfa_rot = 0.05 # Grad step of orientation - Default: 0.4
CP_ur5_rep = [0.15]*6 # Repulsive fields on UR5
CP_ur5_rep[-2] = 0.15
if args.AAPF and not args.OriON or (AAPF_COMP and not ORI_COMP):
    self.eta = [0.0006 for i in range(6)]

ptAtual = get_ur5_position(self.ur5_param, self.joint_states.position,
    "grasping_link")

hz = get_param("rate", 120)
r = rospy.Rate(hz)

dist_EOF_to_Goal = np.linalg.norm(ptAtual -
    np.asarray(self.ptFinal[0]))
dist_EOF_to_Goal_vec = dist_EOF_to_Goal

n = 0
err_ori = 1
corr = [R, P, Y]

# Get current orientation and position of grasping_link
ur5_joint_positions_vec = self.joint_states.position

joint_attractive_forces = np.zeros(6)
joint_rep_forces = np.zeros(6)
ori_atual_vec = np.zeros(3)
```

```
while (dist_EOF_to_Goal > err or abs(err_ori) > 0.02) and not
    rospy.is_shutdown() and n < max_iter:

    # Get UR5 Jacobian of each link
    Jacobian = get_geometric_jacobian(
        self.ur5_param, self.joint_states.position)

    # Get position and distance from each link to each obstacle
    CP_pos, CP_dist = self.get_repulsive_cp(self.obs_pos,
        self.joint_states.position, CP_ur5_rep)

    oriAtual = euler_from_matrix(self.matrix_from_joint_angles())
    oriAtual = [oriAtual[i] + corr[i] for i in range(len(corr))]

    # Get attractive linear and angular forces and repulsive forces
    joint_att_force_p, joint_att_force_w, joint_rep_force = \
        CPA.get_joint_forces(args, ptAtual, self.ptFinal, oriAtual,
            Displacement,
                        dist_EOF_to_Goal, Jacobian,
                            self.joint_states.position,
                            self.ur5_param, zeta,
                        self.eta, rho_0, dist_att, dist_att_config,
                            CP_dist, CP_pos, self.obs_pos,
                            CP_ur5_rep, AAPF_COMP)

    joint_rep_force = np.clip(joint_rep_force, -0.1, 0.1)

    # Joint angles UPDATE - Attractive force
    self.joint_states.position = self.joint_states.position + \
        alfa * joint_att_force_p[0]
    if args.OriON or ORI_COMP:
        self.joint_states.position = self.joint_states.position + \
            alfa_rot * joint_att_force_w[0]

    # Joint angles UPDATE - Repulsive force
    list = np.transpose(joint_rep_force[0]).tolist()

    for j in range(6):
        for i in range(6):
            self.joint_states.position[i] =
                self.joint_states.position[i] + \
                alfa * list[j][i]

    matrix = self.matrix_from_joint_angles()

    # Get current position of grasping_link
```

```
ptAtual = get_ur5_position(self.ur5_param,
    self.joint_states.position, "grasping_link")

if args.plotPath:
    # Visualize path planned in RVIZ
    self.visualize_path_planned(ptAtual)

# Get absolute orientation error
err_ori = np.sum(oriAtual)

# Get distance from EOF to goal
dist_EOF_to_Goal = np.linalg.norm(ptAtual -
    np.asarray(self.ptFinal))

# If true, publish topics to transform into csv later on
if args.CSV:
    ur5_joint_positions_vec = np.vstack((ur5_joint_positions_vec,
        self.joint_states.position))
    dist_EOF_to_Goal_vec = np.vstack((dist_EOF_to_Goal_vec,
        dist_EOF_to_Goal))
    joint_attractive_forces = np.vstack((joint_attractive_forces,
        joint_att_force_p))
    joint_rep_forces = np.vstack((joint_rep_forces, list[-1])) #
        list[-1] corresponds to rep forces on the last joint
    ori_atual_vec = np.vstack((ori_atual_vec, oriAtual))

# If true, publish topics to publish_trajectory.py in order to see
    the path in RVIZ
# if n % 10 is used to reduced the total number of waypoints
    generated by APF or AAPF
if args.plot:
    if n % 1 == 0:
        self.pose.pose.position.x = ptAtual[0]
        self.pose.pose.position.y = ptAtual[1]
        self.pose.pose.position.z = ptAtual[2]
        self.pose_publisher.publish(self.pose)
        # Save way points in order to send it to gazebo

way_points.append(self.joint_states.position)

try:
    r.sleep()
except rospy.exceptions.ROSTimeMovedBackwardsException:
    pass

n += 1
```

```
        return way_points, n, dist_EOF_to_Goal, ur5_joint_positions_vec,
            dist_EOF_to_Goal_vec, joint_attractive_forces, joint_rep_forces,
            ori_atual_vec

    def trajectory_execution(self, args, point, home_pos, way_points, traj):
        raw_input("' ========== Aperte enter para iniciar o algoritmo")

        # if ur5_robot_APF:
        #     ur5_robot_APF.scene.clear

        self.scene.clear()

        # Final positions
        self.ptFinal = point

        # Angle correction relative to base_link (from grasping_link)
        R, P, Y = 1.5707, 0, -1.5707

        if not args.COMP:
            way_points, n, dist_EOF_to_Goal, ur5_joint_positions_vec,
                dist_EOF_to_Goal_vec, joint_attractive_forces,
                joint_rep_forces, ori_atual_rep = \
                self.CPA_loop(args, way_points, R, P, Y)

            # Smooth path generated by AAPF
            wayPointsSmoothed = smooth_path(way_points)

            print_output(n, way_points, wayPointsSmoothed, dist_EOF_to_Goal)

            if args.OriON:
                oriStatus = 'OriON'
            else:
                oriStatus = 'OriOFF'

            if args.AAPF:
                APFStatus = 'AAPF'
            elif args.APF:
                APFStatus = 'APF'

            if args.CSV:
                print_joint_angles(args, ur5_joint_positions_vec, 'joint_' +
                    str(APFStatus) + '_' + str(oriStatus), 'joint')
                print_joint_angles(args, wayPointsSmoothed, 'joint_' +
                    'smoothed_' + str(APFStatus) + '_' + str(oriStatus),
                    'joint')
                print_joint_angles(args, dist_EOF_to_Goal_vec, 'dist_to_goal_'
                    + str(APFStatus) + '_' + str(oriStatus), 'dist_to_goal')
```

```
                print_joint_angles(args, joint_attractive_forces[1:],
                    'ATTFORCE_' + str(APFStatus) + '_' + str(oriStatus),
                    'joint')
                print_joint_angles(args, joint_rep_forces[1:], 'REPFORCE_' +
                    str(APFStatus) + '_' + str(oriStatus), 'joint')
                # plt.plot(np.linspace(0, len(joint_rep_forces),
                    len(joint_rep_forces)), joint_rep_forces)
                # plt.show()

                if args.OriON:
                    print_joint_angles(args, ori_atual_rep[1:], 'ORC_' +
                        str(APFStatus) + '_' + str(oriStatus), 'euler_angles')

            if args.plot:
                # choose the trajectory to display in RVIZ
                self.pose.header.frame_id = "trajectory"
                self.pose_publisher.publish(self.pose)

            if args.realUR5:
                raw_input("' ========== Aperte enter para enviar a trajetoria
                    para o UR5 !!!REAL!!! \n")
                self.move(wayPointsSmoothed, "real")
            else:
                raw_input("' ========== Press enter to send the trajectory to
                    Gazebo \n")
                self.move(wayPointsSmoothed, "gazebo")

        else:
            print("Generatin trajectory [AAPF: ON | Orientation Control: ON]")
            AAPF_COMP = True
            ORI_COMP = True
            _, _, _, _, dist_EOF_to_Goal_AAPF_OriON, _, _, _ =
                self.CPA_loop(args, way_points, R, P, Y, AAPF_COMP, ORI_COMP)
            dist_EOF_to_GOAL_traj = []
            dist_EOF_to_GOAL_traj.append(dist_EOF_to_Goal_AAPF_OriON)

            self.joint_states.position = home_pos
            self.scene.clear()
            print("Generatin trajectory [AAPF: OFF | Orientation Control: ON]")
            AAPF_COMP = False
            ORI_COMP = True
            _, _, _, _, dist_EOF_to_Goal_APF_OriON, _, _, _ =
                self.CPA_loop(args, way_points, R, P, Y, AAPF_COMP, ORI_COMP)
            dist_EOF_to_GOAL_traj.append(dist_EOF_to_Goal_APF_OriON)

            self.joint_states.position = home_pos
            self.scene.clear()
```

```
            print("Generatin trajectory [AAPF: ON | Orientation Control: OFF]")
            AAPF_COMP = True
            ORI_COMP = False
            _, _, _, _, dist_EOF_to_Goal_AAPF_OriOFF, _, _, _ =
                self.CPA_loop(args, way_points, R, P, Y, AAPF_COMP, ORI_COMP)
            dist_EOF_to_GOAL_traj.append(dist_EOF_to_Goal_AAPF_OriOFF)

            if args.CSV:
                print_joint_angles(args, dist_EOF_to_GOAL_traj,
                    'COMP_dist_to_goal', 'dist_to_goal')



def main(args):
    ur5_robot = MoveGroupPythonIntefaceTutorial(args)
    way_points = []
    ur5_robot.scene.clear()

    # UR5 Initial position
    home_pos = [3.14, -1.5707, 0, -1.5707, -1.5707, -1.5707] # default: [0,
        -1.5707, 0, -1.5707, -1.5707, -1.5707]
    ur5_robot.joint_states.position = home_pos
    ur5_robot.path_color = ColorRGBA(0.0, 1.0, 0.0, 0.8)
    way_points.append(ur5_robot.joint_states.position)
    ur5_robot.move(way_points, "gazebo")

    if args.realUR5:
        raw_input("' ========== Aperte enter para posicionar o UR5 !!REAL!!
            na posicao UP\n")
        ur5_robot.move((home_pos,), "real")

    '''
    Trajectory 1
    '''
    point = [[-0.80, 0, 0.45]]
    way_points = []
    ur5_robot.trajectory_execution(args, point, home_pos, way_points,
        'traj_1')#, ur5_robot_APF)

if __name__ == '__main__':
    try:
        arg = parse_args()
        main(arg)
    except rospy.ROSInterruptException:
        pass
    except KeyboardInterrupt:
```

```
        pass
```

APPENDIX B

APF CODE

```
#!/usr/bin/env python

# ROS import
import sys
import rospy
from tf.transformations import quaternion_from_euler, quaternion_matrix
import tf
from tf import TransformListener, Transformer
import publish_joint_states
from publish_joint_states import *

# Moveit Import
import moveit_commander
# import moveit_python
from moveit_commander.conversions import pose_to_list

# Msg Import
from moveit_msgs.msg import *
from moveit_msgs.srv import *
from geometry_msgs.msg import *
from std_msgs.msg import String, Header, ColorRGBA
from visualization_msgs.msg import Marker
from shape_msgs.msg import SolidPrimitive
from sensor_msgs.msg import JointState

# Inverse Kinematics Import
from ur_inverse_kinematics import *

# Python Import
import numpy as np
from numpy import array, dot, pi
from numpy.linalg import det, norm

# Customized code
from get_geometric_jacobian import *
from get_ur5_position import *
from show_HTM import *
from get_dist3D import *
# import CPA_classico
```

```
from CPA_classico import *


"""
### Use instructions
# First launch RViz:
roslaunch ur5_moveit_config demo.launch
# Second - run UR5_CPA file
rosrun UR5_CPA.py
"""

def get_param(name, value=None):
    private = "~%s" % name
    if rospy.has_param(private):
        return rospy.get_param(private)
    elif rospy.has_param(name):
        return rospy.get_param(name)
    else:
        return value

class MoveGroupPythonIntefaceTutorial(object):
    """MoveGroupPythonIntefaceTutorial"""

    def __init__(self):
        super(MoveGroupPythonIntefaceTutorial, self).__init__()

        moveit_commander.roscpp_initialize(sys.argv)
        rospy.init_node('move_group_python_interface_ur5_robot',
                        anonymous=True)

        self.display_trajectory_publisher =
            rospy.Publisher('/move_group/display_planned_path',
                                                    moveit_msgs.msg.DisplayTrajectory,
                                                    queue_size=20)
        rospy.sleep(0.5)

        # Topico para publicar marcadores para o Rviz
        self.marker_publisher = rospy.Publisher('visualization_marker',
            Marker, queue_size=100)
        rospy.sleep(0.5)

        self.tf = TransformListener()
        rospy.sleep(0.5)

        self.init_id_path = 13

        # Topico para publicar no /robot_state_publisher
```

```
        self.pub_joint_states = rospy.Publisher('joint_states_ur5',
            JointState, queue_size=50)


        self.marker = Marker()
        self.joint_states = JointState()
        self.joint_states.name = ['shoulder_pan_joint', 'shoulder_lift_joint',
            'elbow_joint', 'wrist_1_joint', 'wrist_2_joint',
    'wrist_3_joint']

        # d1, SO, EO, a2, a3, d4, d45, d5, d6
        self.ur5_param = (0.089159, 0.13585, -0.1197, 0.425, 0.39225, 0.10915,
            0.093, 0.09465, 0.0823 + 0.15)

    """
    Calculate the initial robot position - Used before CPA application
    Need to update: pass analytical homogeneous transformation to invKine
    """
    def get_ik(self, pose):
        matrix = tf.TransformerROS()
        # The orientation of /grasping_link will be constant
        q = quaternion_from_euler(1.5707, 1.5707, 0)

        matrix2 = matrix.fromTranslationRotation((pose[0]*(-1), pose[1]*(-1),
            pose[2]), (q[0], q[1], q[2], q[3]))
        # print "The quaternion representation is %s %s %s %s." % (q[0], q[1],
            q[2], q[3])

        rospy.loginfo(matrix2)
        th = invKine(matrix2)
        sol1 = th[:, 0].transpose()
        joint_values_from_ik = np.array(sol1)

        joint_values = joint_values_from_ik[0, :]

        return joint_values.tolist()

    """
    Also gets each frame position through lookupTransform
    """
    def get_repulsive_cp(self, obs_pos, joint_values, diam_rep_ur5):

        marker_lines = Marker()

        ur5_links = [
            "shoulder_link",
            "upper_arm_link",
```

```
            "forearm_link",
            "wrist_1_link",
            "wrist_2_link",
            "wrist_3_link",
            "grasping_link"
        ]

        cp_position, cp_distance = [], []
        for i in range(len(ur5_links)):
            # link_pose = get_ur5_position(self.ur5_param, joint_values,
                ur5_links[i])
            link_pose, _ = self.tf.lookupTransform("base_link", ur5_links[i],
                rospy.Time())
            cp_position.append(link_pose)
            cp_distance.append(np.linalg.norm(link_pose - np.asarray(obs_pos)))
            # print(cp_distance)
            marker_lines.points.append(Point(obs_pos[0], obs_pos[1],
                obs_pos[2]))
            marker_lines.points.append(Point(link_pose[0], link_pose[1],
                link_pose[2]))
            self.add_sphere(link_pose, i, diam_rep_ur5, ColorRGBA(1.0, 0.0,
                0.0, 0.3))

        return cp_position, cp_distance

    """
    Adds lines representing distances from obstacles to the robot control's
        point
    """
    def add_line(self, marker):
        marker.header.frame_id = "base_link"
        marker.type = marker.LINE_STRIP
        marker.action = marker.MODIFY
        marker.scale = Vector3(0.008, 0.009, 0.1)
        marker.color = ColorRGBA(0.0, 1.0, 0.0, 0.8)
        self.marker_publisher.publish(marker)

    """
    Adds the obstacles and repulsive control points on the robot
    """
    def add_sphere(self, pose, id, diam, color):
        marker = Marker()
        marker.header.frame_id = "base_link"
        marker.id = id
        marker.pose.position = Point(pose[0], pose[1], pose[2])
        marker.type = marker.SPHERE
        marker.action = marker.MODIFY
```

```
        marker.scale = Vector3(diam, diam, diam)
        marker.color = color
        self.marker_publisher.publish(marker)

    """
    Plot robot's path to the RViz environment
    """
    def visualize_path_planned(self, path):
        self.marker.points.append(Point(path[0], path[1], path[2]))
        self.marker.header.frame_id = "base_link"
        self.marker.id = 14
        self.marker.type = self.marker.LINE_STRIP
        self.marker.action = self.marker.ADD
        self.marker.scale = Vector3(0.008, 0.009, 0.1)
        self.marker.color = ColorRGBA(0.0, 1.0, 0.0, 0.8)
        self.marker_publisher.publish(self.marker)

    """
    Delete all markers in Rviz
    """
    def delete_markers(self):
        marker = Marker()
        marker.action = marker.DELETEALL
        self.marker_publisher.publish(marker)

def main():
    ur5_robot = MoveGroupPythonIntefaceTutorial()
    ur5_robot.delete_markers()

    ### UR5 Initial position
    raw_input("' ========== Aperte enter para posicionar o UR5 \n")
    ur5_robot.joint_states.position = [0, -1.5707, 0, -1.5707, 0, 0] # Posicao
        configurada no fake_controller_joint_states
    ur5_robot.pub_joint_states.publish(ur5_robot.joint_states)

    raw_input("' ========== Aperte enter para carregar o obstaculo e objetivo
        \n")
    ### Final and obstacle points
    obs_pos = [0.45, 0.4, 0.4]
    diam_obs = 0.4
    ur5_robot.add_sphere(obs_pos, 11, diam_obs, ColorRGBA(1.0, 0.0, 0.0, 0.5))

    ptFinal = [0.45, 0.3, 0.5]
    oriFinal = [0.01, 0.01, 0.01]
    diam_goal = 0.04
    ur5_robot.add_sphere(ptFinal, 13, diam_goal, ColorRGBA(0.0, 1.0, 0.0, 0.8))
```

```
### CPA Parameters
err = diam_goal/2
max_iter = 5000
zeta = [0.5 for i in range(7)]
eta = [0.005 for i in range(6)]
rho_0 = diam_obs
dist_att = 0.05
dist_att_config = 0.15
alfa = 0.5
alfa_rot = 0.2
diam_rep_ur5 = 0.15

raw_input("' ========== Pressione enter para posicionar o UR5 \n")
ur5_robot.joint_states.position = [0, -1.5707, 0, -1.5707, 0, 0] # Posicao
    configurada no fake_controller_joint_states
ur5_robot.pub_joint_states.publish(ur5_robot.joint_states)

### GET repulsive control point's position through LOOKUPTRANSFORM
CP_pos, CP_dist = ur5_robot.get_repulsive_cp(obs_pos,
    ur5_robot.joint_states.position, diam_rep_ur5)

### Parameters
CPAA_state = False
Orientation_state = True

hz = get_param("rate", 50) # 10hz
r = rospy.Rate(hz)

ptAtual, oriAtual = ur5_robot.tf.lookupTransform("base_link",
    "grasping_link", rospy.Time())

dist_EOF_to_Goal = np.linalg.norm(ptAtual - np.asarray(ptFinal))
n = 0

raw_input("' ========== Aperte enter para iniciar o algoritmo dos CPAs")
while dist_EOF_to_Goal > err and not rospy.is_shutdown():
    Jacobian = get_geometric_jacobian(ur5_robot.ur5_param,
        ur5_robot.joint_states.position)
    CP_pos, CP_dist = ur5_robot.get_repulsive_cp(obs_pos,
        ur5_robot.joint_states.position, diam_rep_ur5)

    joint_att_force_p, joint_att_force_w, joint_rep_force =
        CPA_classico.get_joint_forces(ptAtual, ptFinal, oriAtual, oriFinal,
    dist_EOF_to_Goal, Jacobian, ur5_robot.joint_states.position,
        ur5_robot.ur5_param, zeta,
    eta, rho_0, dist_att, dist_att_config, CP_dist, CP_pos, obs_pos,
        CPAA_state, diam_rep_ur5)
```

```
        # Joint angles UPDATE
        ur5_robot.joint_states.position = ur5_robot.joint_states.position +
            alfa*joint_att_force_p[0]
        if Orientation_state:
            ur5_robot.joint_states.position = ur5_robot.joint_states.position
                + alfa_rot*joint_att_force_w[0]

        list = np.transpose(joint_rep_force[0]).tolist()
        for j in range(6):
            for i in range(6):
                ur5_robot.joint_states.position[i] =
                    ur5_robot.joint_states.position[i] + alfa*list[j][i]
        ptAtual, oriAtual = ur5_robot.tf.lookupTransform("base_link",
            "grasping_link", rospy.Time())
        oriAtual += quaternion_from_euler(1.5707, 0, 0)

        if n % 10 == 0:
            ur5_robot.visualize_path_planned(ptAtual)
            print("Distance to the goal: " + str(dist_EOF_to_Goal))
        dist_EOF_to_Goal = np.linalg.norm(ptAtual - np.asarray(ptFinal))
        ur5_robot.pub_joint_states.publish(ur5_robot.joint_states)

        try:
            r.sleep()
        except rospy.exceptions.ROSTimeMovedBackwardsException:
            pass

        n += 1

if __name__ == '__main__':
    try:
        main()
    except rospy.ROSInterruptException:
        pass
    except KeyboardInterrupt:
        pass
```

APPENDIX C

GEOMETRIC JACOBIAN

```
from numpy import array, cos, sin
import numpy as np

"""
Anallytical jacobian
type:
    ur5_param: list
    joint_values: list
"""
def get_geometric_jacobian(ur5_param, joint_values):

    th1, th2, th3, th4, th5, th6 = joint_values
    d1, S0, E0, a2, a3, d4, d45, d5, d6 = ur5_param

    # J_1 = array([[0, 0, 0, 0, 0, 0, 0],
    #              [0, 0, 0, 0, 0, 0, 0],
    #              [0, 0, 0, 0, 0, 0, 0],
    #              [0, 0, 0, 0, 0, 0, 0],
    #              [1, 0, 0, 0, 0, 0, 0],
    #              [0, 0, 0, 0, 0, 0, 0]])

    J_2_T = ([[-S0*cos(th1), -S0*sin(th1), 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])


    J_3_T = ([[-E0*cos(th1) - S0*cos(th1) - a2*sin(th1)*cos(th2), -E0*sin(th1)
        - S0*sin(th1) + a2*cos(th1)*cos(th2), 0],
            [-a2*sin(th2)*cos(th1), -a2*sin(th1)*cos(th2), -a2*cos(th2)],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])
```

```
J_4_T = ([[-E0*cos(th1) - S0*cos(th1) - a2*sin(th1)*cos(th2) -
    a3*sin(th1)*cos(th2 + th3), -E0*sin(th1) - S0*sin(th1) +
    a2*cos(th1)*cos(th2) + a3*cos(th1)*cos(th2 + th3), 0],
        [(-a2*sin(th2) - a3*sin(th2 + th3))*cos(th1), (-a2*sin(th2) -
           a3*sin(th2 + th3))*sin(th1), -a2*cos(th2) - a3*cos(th2 + th3)],
        [-a3*sin(th2 + th3)*cos(th1), -a3*sin(th1)*sin(th2 + th3),
           -a3*cos(th2 + th3)],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])


J_5_T = ([[-E0*cos(th1) - S0*cos(th1) - a2*sin(th1)*cos(th2) -
    a3*sin(th1)*cos(th2 + th3) - d45*cos(th1), -E0*sin(th1) - S0*sin(th1)
    + a2*cos(th1)*cos(th2) + a3*cos(th1)*cos(th2 + th3) - d45*sin(th1), 0],
        [(-a2*sin(th2) - a3*sin(th2 + th3))*cos(th1), (-a2*sin(th2) -
           a3*sin(th2 + th3))*sin(th1), -a2*cos(th2) - a3*cos(th2 + th3)],
        [-a3*sin(th2 + th3)*cos(th1), -a3*sin(th1)*sin(th2 + th3),
           -a3*cos(th2 + th3)],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])


J_6_p_T = ([[-E0*cos(th1) - S0*cos(th1) - a2*sin(th1)*cos(th2) -
    a3*sin(th1)*cos(th2 + th3) - d45*cos(th1) + d5*sin(th1)*sin(th2 + th3
    + th4), -E0*sin(th1) - S0*sin(th1) + a2*cos(th1)*cos(th2) +
    a3*cos(th1)*cos(th2 + th3) - d45*sin(th1) - d5*sin(th2 + th3 +
    th4)*cos(th1), 0],
          [(-a2*sin(th2) - a3*sin(th2 + th3) - d5*cos(th2 + th3 +
             th4))*cos(th1), (-a2*sin(th2) - a3*sin(th2 + th3) -
             d5*cos(th2 + th3 + th4))*sin(th1), -a2*cos(th2) - a3*cos(th2
             + th3) + d5*sin(th2 + th3 + th4)],
          [(-a3*sin(th2 + th3) - d5*cos(th2 + th3 + th4))*cos(th1),
             (-a3*sin(th2 + th3) - d5*cos(th2 + th3 + th4))*sin(th1),
             -a3*cos(th2 + th3) + d5*sin(th2 + th3 + th4)],
          [-d5*cos(th1)*cos(th2 + th3 + th4), -d5*sin(th1)*cos(th2 + th3 +
             th4), d5*sin(th2 + th3 + th4)],
          [0, 0, 0],
          [0, 0, 0]])

gain_3 = 2.0
J_7_T = ([[-E0*cos(th1) - S0*cos(th1) - a2*sin(th1)*cos(th2) -
    a3*sin(th1)*cos(th2 + th3) - d45*cos(th1) + d5*sin(th1)*sin(th2 + th3
    + th4) - d6*(sin(th1)*sin(th5)*cos(th2 + th3 + th4) +
    cos(th1)*cos(th5)), -E0*sin(th1) - S0*sin(th1) + a2*cos(th1)*cos(th2)
    + a3*cos(th1)*cos(th2 + th3) - d45*sin(th1) - d5*sin(th2 + th3 +
```

```
        th4)*cos(th1) - d6*(sin(th1)*cos(th5) - sin(th5)*cos(th1)*cos(th2 +
        th3 + th4)), 0],
            [(-a2*sin(th2) - a3*sin(th2 + th3) - d5*cos(th2 + th3 + th4) -
               d6*sin(th5)*sin(th2 + th3 + th4))*cos(th1), (-a2*sin(th2) -
               a3*sin(th2 + th3) - d5*cos(th2 + th3 + th4) -
               d6*sin(th5)*sin(th2 + th3 + th4))*sin(th1), -a2*cos(th2) -
               a3*cos(th2 + th3) + d5*sin(th2 + th3 + th4) -
               d6*sin(th5)*cos(th2 + th3 + th4)],
            [((-a3*sin(th2 + th3) - d5*cos(th2 + th3 + th4) -
               d6*sin(th5)*sin(th2 + th3 + th4))*cos(th1))*gain_3,
               ((-a3*sin(th2 + th3) - d5*cos(th2 + th3 + th4) -
               d6*sin(th5)*sin(th2 + th3 + th4))*sin(th1))*gain_3,
               (-a3*cos(th2 + th3) + d5*sin(th2 + th3 + th4) -
               d6*sin(th5)*cos(th2 + th3 + th4))*gain_3],
            [(-d5*cos(th2 + th3 + th4) - d6*sin(th5)*sin(th2 + th3 +
               th4))*cos(th1), (-d5*cos(th2 + th3 + th4) - d6*sin(th5)*sin(th2
               + th3 + th4))*sin(th1), d5*sin(th2 + th3 + th4) -
               d6*sin(th5)*cos(th2 + th3 + th4)],
            [d6*(sin(th1)*sin(th5) + cos(th1)*cos(th5)*cos(th2 + th3 + th4)),
               d6*(sin(th1)*cos(th5)*cos(th2 + th3 + th4) -
               sin(th5)*cos(th1)), -d6*sin(th2 + th3 + th4)*cos(th5)],
            [0, 0, 0]])


    J_6_w_T = ([[0, 1, 0],
             [-sin(th1), cos(th1), 0],
             [-sin(th1), cos(th1), 0],
             [-sin(th1), cos(th1), 0],
             [-sin(th2 + th3 + th4)*cos(th1), -sin(th1)*sin(th2 + th3 + th4),
                -cos(th2 + th3 + th4)],
             [-sin(th1)*cos(th5) + sin(th5)*cos(th1)*cos(th2 + th3 + th4),
                sin(th1)*sin(th5)*cos(th2 + th3 + th4) + cos(th1)*cos(th5),
                -sin(th5)*sin(th2 + th3 + th4)]])

    return (J_2_T, J_3_T, J_4_T, J_5_T, J_6_p_T, J_7_T, J_6_w_T)
```