A Combined Approach for Motion Learning and Obstacle Avoidance

ROBOTICS SEMESTER PROJECT

LEARNING ALGORITHMS AND SYSTEMS LABORATORY

STUDENT: CHRISTOPHER J. STOCKER
SUPERVISOR: LUKAS HUBER
PROFESSOR: AUDE BILLARD

Abstract

As human-robot interaction becomes increasingly prevalent through collaborative robots, new safety requirements are needed to ensure the user's protection at all times. This means that robots need to adapt to human behavior, and not the other way around. To achieve this in human-inhabited environments they need to be robust to perturbances and be able to reevaluate in milliseconds its new path to avoid a collision.

This report analyzes the development of an experimental setup for full-body obstacle avoidance applied to a robot arm. The obstacle avoidance problem is adapted from a closed-form approach utilized to constrain flow within a given volume and around objects. This approach ensures the flow converges and stops at a single fixed point. This technique can converge towards an attractor whilst avoiding both static and dynamics obstacles. This has never before been applied to a robot working in 3D space. For this reason, the experimental setup and simulation environment were built on a virtual replica of the Franka Emika robot arm.

Acknowledgments

I would first like to express my sincere gratitude to Prof. Aude Billard for allowing me to follow this project in her lab.

My work could not have been made possible without the help and support of my project supervisor Lukas Huber. When bugs seemed to multiply he was always kind to lend his generous time, quick feedback, and patience.

Similarly, I would like to thank Dominique Reber for his kindness to help debug the AICA dockers and for sharing his key observations.

Contents

1 Introduction

1.1 Background

Figure 1: Collaborative robot assisting operator.
Image Source: Journal of Advanced Manufacturing Technology

The Annual Report of the White House and Council of Economic Advisers foresee that millions of low-paying jobs are at risk of being taken over by automation [1]. Robots will be a disruptive force in the years to come, however, what if instead of replacing humans they were here to empower us.

Collaborative robots seek to combine the synergies between humans and robots. Industrial robots are typically able to carry high payloads, be extremely precise, and work continuously on repetitive tasks, while human workers are easily able to change to new tasks, use problem-solving skills, and react to new situations. What if in the future, the role of robots is to become complementing actors to the human condition by aiding the body in strength, the mind in calculation, and fatigue with automation.

New developments are pushing automation beyond traditional manufacturing and further into new industries. Thanks to new advancements in machine learning, computer vision, and motion planning, more and more sectors are finding new ways to integrate collaborative robots into their workforce. These advancements place collaborative robots at the center stage in one of the fastest-growing robotic markets [2]. As a result, collaborative robots are increasingly becoming adopted by small business owners.

Although automation will put some jobs at risk, the next generation of robots will not necessarily replace humans but enhance them by combining the strengths of both.

1.2 Problem Statement

Figure 2: Robot caging for human safety.
Image Source: ABB

It's been said the future lies in taking the robot out of the human, meaning, liberating humans from monotonous robotic tasks. However, the reverse is also true, the future lies in putting human qualities into robots. Big productivity improvements could be made if collaborative robots could work alongside people to perform routine work whose actions are still best done by human minds. Today one of their major limitation has been their safety to work around humans.

Industrial robot have been designed to operate inside a cage, well away from human workers as per Figure 3. The research towards collaborative robots has looked at how to make these new generations of robots safe enough to eliminate the cage.

One way to make robots safer is to make them slower. Although improving safety, their slow movements can hamper productivity as a bottleneck to the entire production process. A second solution has been to equip robots with force sensors at the joints. These sensors immediately halt the robot the moment it has felt a collision. Although effective, the work is an abrupt sequence of stop and go as the robot has no means of perceiving a collision until it has happened. Lastly, proximity scanners have been added to robots to improve their awareness of where people are, but again have been used so far to modulate the speed of the robot's movement but not its direction.

Changing the direction of the robot's movement is defined by its navigation control. Robots navigating in human-inhabited environments can encounter countless static and dynamic obstacles. For this reason navigation in human environments combines navigation control with dynamic obstacle avoidance. Collaborative robots should be able to navigate in rapidly changing environments and react to perturbances in the length of milliseconds without compromising the safety of the user. This makes it clear that robots should not only have an excellent awareness of their environment but also the built-in protocols to react accordingly.

1.3 Literature Background

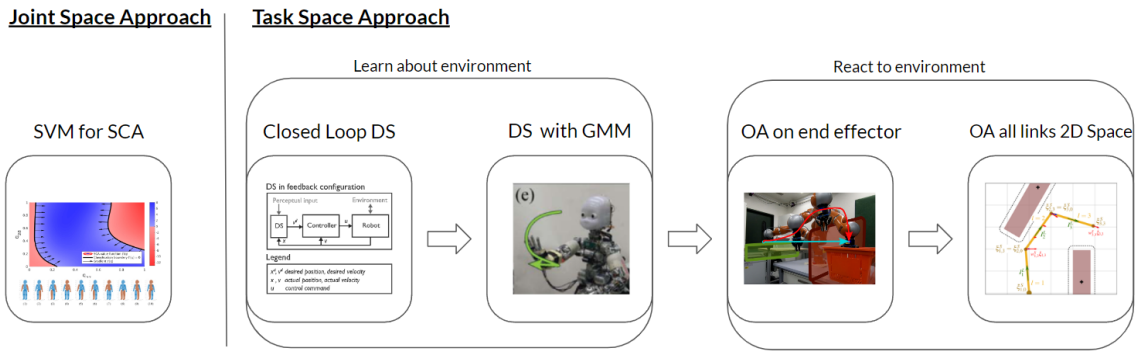


Figure 3: Review of different self-collision (SCA) and obstacle avoidance (OA) in task space and joint space.

The literature background on motion planning and obstacle avoidance has largely been dominated by computer sciences. Researchers aim at devising stable algorithms with well-understood completeness. For this reason, many have tackled the whole problem of obstacle and self collision avoidance by embracing tools of machine learning. To give a breadth of the state of the art this literature background gives a current example of real-time self-collision avoidance in joint space. The reviewed approaches dive into machine learning tools that although immensely beneficial may be subject to qualitative heuristics. The the approach my semester project took is inspired by principles of movement from fluid dynamics and control theory acting in the task space. To understand the framework of my semester project this review brushes up on the string of papers that build this new approach for dynamic obstacle avoidance in task space and offer an interesting new possibility for dynamic obstacle avoidance in human environments.

Self-Collision Avoidance in Joint Space

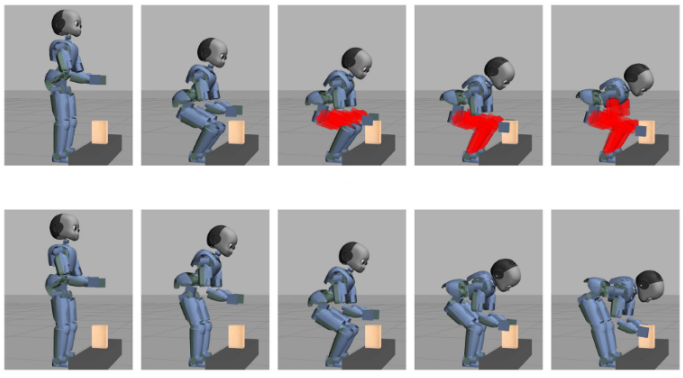


Figure 4: Self collision avoidance learning for iCub robot model.

As mentioned before machine learning has proven to offer new tools for real-time self-collision avoidance. One example of this was the use of support vector machines (SVM) to learn the boundary functions of a robot in the joint space. This was done for the high-Dof humanoid robot iCub. This was done by decomposing the robot's 29-dimensional joint space into independent

sub-models of lower dimensionality. By taking advantage of symmetry the number of independent models to be learned was reduced. To learn the boundary condition classic SVM formulation was taken by using the kernel trick to lift the data to higher dimensional space and there find a separation of the hyperplane in feature space [3].

Although the sub-model system had an average accuracy of 96% for predicting colliding vs. non-colliding self configurations, the accuracy fell to 86% when looking at the full-body model. This is one of the major drawbacks of this approach as model separation may not be evident and requires some qualitative heuristics specific to the iCub robot. The model only needs one extensive computation and afterward provides a reliable, efficient, and real-time self-collision detection.

Self Collision and Obstacle Avoidance in Task Space

Dynamic systems have emerged as a popular framework for representing robot motion. It lends itself well to supervised and reinforcement learning. However, one major drawback to traditional dynamic systems is its open-loop configuration. The end goal of our robot is to have built-in mechanisms that ensure human safety by allowing it to react to the environment. To do so, it must be able to react to physical perturbation thus demanding a closed-loop controller that feeds back the state of the robot to the dynamic system. One way to close the control loop is by introducing a damping matrix into the system to allow passive velocity feedback control and the concept of energy tanks to allow velocity feedback control when the desired dynamics are non-conservative [4].

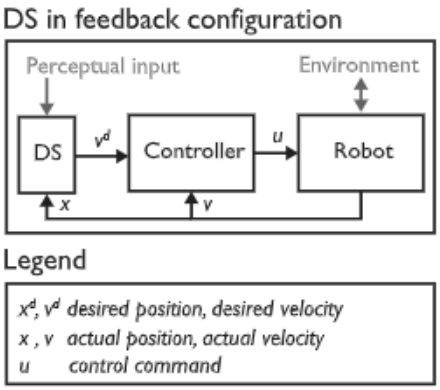


Figure 5: Closed-Loop dynamic system.

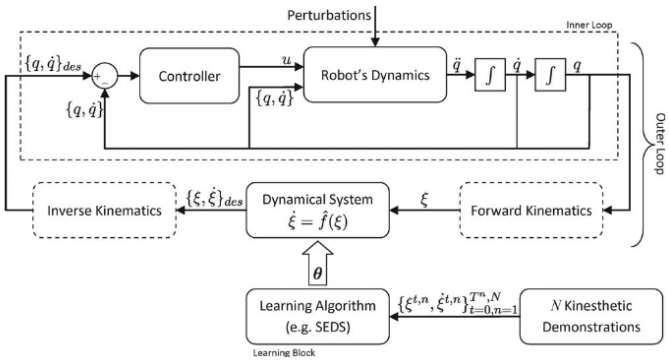
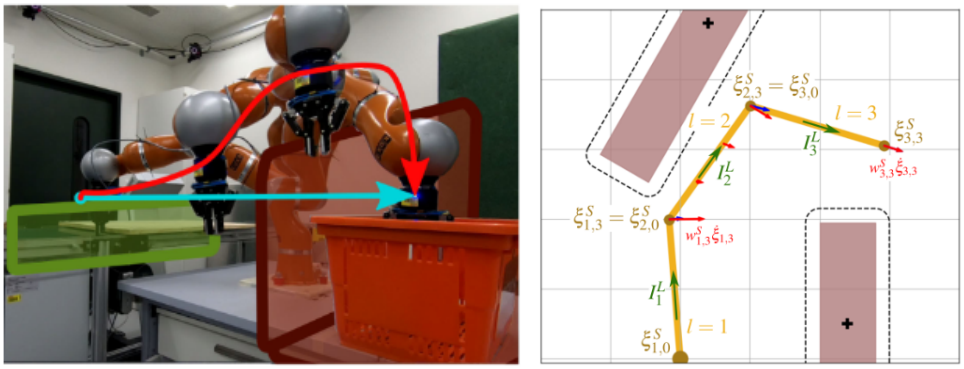


Figure 6: SEDS motion learning configuration on dynamic systems.

The next obvious question is if machine learning tools can be applied to this nonlinear dynamical system. As mentioned before the goal is to give some level of cognition back to the robot to learn from its environment. For this, a new learning method was proposed called *Stable Estimator of Dynamical Systems* (SEDS). This method has proven to closely follow the demonstration while reaching the target. This offers a framework for fast learning from a small set of demonstrations. SEDS achieves this by statistically encoding a dynamical motion using Gaussian Mixture Models [5]. Since the end goal is to use a dynamical system-based motion learning, the next two papers present the groundwork for the obstacle avoidance framework that was used for this semester project.

Moreover, the framework of the robot must be able to also react when to multiple dynamic obstacles and static obstacles. This obstacle avoidance framework has been applied to a real robot but only implementing obstacle avoidance on the end effector. This has been demonstrated by taking inspiration from harmonic potential fields [6]. This presents a simple, fast, and closed-loop method, however, it works with a linear dynamics system.

The final step is then to expand the use to a nonlinear dynamic system that uses obstacle avoidance in all the segments of the robot not just the end effector. This brings me to the final paper of this review that focuses on obstacle avoidance and dynamic systems for the complete robot arm. The goal would be to have a framework that is capable of using non-linear dynamic systems and obstacle avoidance that works on the entire robot. The proof of simulation of a robot link in 2D space has been demonstrated [7]. My task for my semester project was to expand robot simulation from a two-dimensional environment to a three-dimensional environment using (Franka Emika), a robot arm with six degrees of freedom.



(a) OA only on end-effector. (b) OA on entire robot link.

Figure 7: (Left) OA on the end effector alone. (Right) OA on two dimensional arm link.

2 Preliminaries

Preliminaries Organization: In Section 2.1 *Obstacle Avoidance*l summarize the the mathematical equations used for obstacle avoidance. In Section 2.2 *Dynamic Environments* I summarize and explain how these are applied to the robotic arm. The formulation for these equations is an agglomeration of focus papers (*"Avoidance of convex and concave obstacles with convergence ensured through contraction"* [6] and *Avoiding Dense and Dynamic Obstacles in Enclosed Spaces: Application to Moving in Crowds) [7]*.

2.1 Obstacle Avoidance Formulation

First of all the motion towards a target goal must be defined before establishing the obstacle avoidance. This direct dynamics is expressed by the following linear dynamical system:

$$\mathbf{f}(\xi) = -k\left(\xi - \xi^a\right) \tag{1}$$

Where ξ is a Cartesian position, $k \in \mathbb{R}$ is a scaling parameter, and ξ^a as an end goal (attractor). The attractor is illustrated in Figure 8 by a star.

Obstacles are described by first classifying the entire space by a distance function $\Gamma(\xi)$ into three different points: free points, interior points, and margin points. Figure 8 depicts the outline of this margin by a dotted point line. These points are mathematically defined as follows:

$$\begin{aligned} &\text{Free points:} && X^f = \left\{\xi \in \mathbb{R}^d : \Gamma(\xi) > 1\right\} \\ &\text{Margin points:} && X^b = \left\{\xi \in \mathbb{R}^d : \Gamma(\xi) = 1\right\} \\ &\text{Interior points:} && X^o = \left\{\xi \in \mathbb{R}^d \backslash \left(X^f \cup X^b\right)\right\} \end{aligned} \tag{2}$$

Each obstacle is described by a reference point ξ_i^r located in the interior of the obstacle and a reference vector $\mathbf{r}_i(\xi)$. This point is illustrated in Figure 8 as a cross.

$$\mathbf{r}_i(\xi) = \left(\xi - \xi_i^r\right) / \left\|\xi - \xi_i^r\right\| \quad \forall \xi \in \mathbb{R}^d \backslash \xi_i^r \tag{3}$$

The distance function is described as the ratio between the distance to the reference point over the distance between the local margin to the reference point.

$$\Gamma^o(\xi) = \left(\left\|\xi - \xi^r\right\| / \left\|\xi^b - \xi^r\right\|\right)^{2p} \quad \forall \xi \in \mathbb{R}^d \backslash \xi^r \tag{4}$$

where ξ^b is defined as:

$$\xi^b = b\mathbf{r}(\xi) + \xi^r \quad \text{such that} \quad b > 0, \xi^b \in X^b \tag{5}$$

Obstacle Avoidance Through Modulation

To obtain an obstacle avoidance in real-time a dynamic modulation matrix is applied to the dynamical system function $\mathbf{f}(\xi)$.

$$\dot{\xi} = \mathbf{M}(\xi)\mathbf{f}(\xi) \quad \text{with} \quad \mathbf{M}(\xi) = \mathbf{E}(\xi)\mathbf{D}(\xi)\mathbf{E}(\xi)^{-1} \tag{6}$$

As can be carefully observed this modulation matrix is formed by two matrices: the basis matrix $\mathbf{E}$ and the diagonal matrix $\mathbf{D}$. The matrix is built by taking the orthonormal tangent vectors $\mathbf{e}_i(\xi)$ at the margin (5).

$$\mathbf{E}(\xi) = \left[\mathbf{r}(\xi)\mathbf{e}_1(\xi)\dots\mathbf{e}_{d-1}(\xi)\right] \tag{7}$$

The diagonal matrix D is formed by choosing a radial eigenvalue and tangent eigenvalues:

$$\mathbf{D}(\xi) = \text{diag}\left(\lambda^r(\xi), \lambda^e(\xi), \dots, \lambda^e(\xi)\right) \tag{8}$$

with the eigenvalues equal to:

$$\lambda^r(\xi) = 1 - 1/\Gamma(\xi) \quad \lambda^e(\xi) = 1 + 1/\Gamma(\xi) \tag{9}$$

Surface Friction and Repulsive Parameter

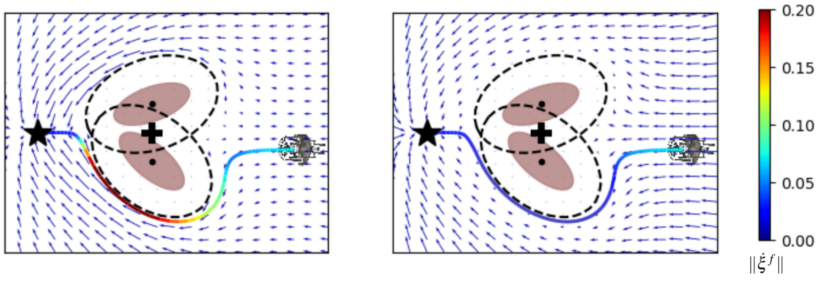


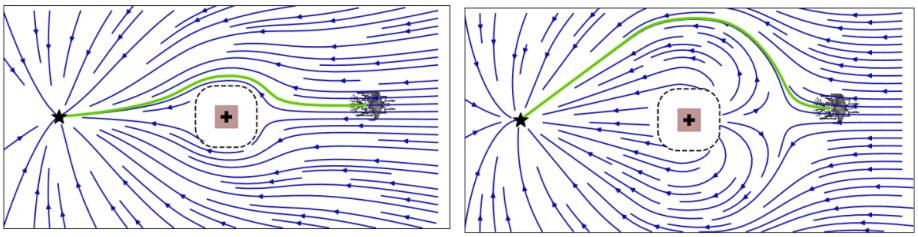
Figure 8: (Left) Flow without fictional force.(Right) Flow with frictional force.

The eigenvalues in the tangent direction chosen for the diagonal matrix (8) are inspired by fluid dynamics representation of the flow of an in-compressible fluid. In fluid dynamics, to keep the flow rate constant, the fluid's velocity must increase when pushed around objects. To avoid the end effector speeding up around the object the parameter $\dot{\xi}^f$ is added to replicate surface friction. This safety mechanism is important since it ensures the slowing down of the end effector as it gets closer to objects.

$$\dot{\xi}^f = \lambda^f(\xi)\frac{\|\mathbf{f}(\xi)\|}{\|\dot{\xi}\|}\dot{\xi} \quad \text{with} \quad \lambda^f(\xi) = 1 - 1/\Gamma(\xi) \tag{10}$$

Furthermore, the agent should not only slow down to obstacles but if possible wish to avoid them. For this, a repulsion component $\lambda^r(\xi)$ is added to keep a minimum distance between the agent and the obstacle. The repulsion can be modulated through a repulsion coefficient $c^{\text{rep}} \geq 1$.

$$\lambda^r(\xi) = \begin{cases} 1 - \left(c^{\text{rep}}/\Gamma(\xi)\right)^{1/\rho} & \text{if } \langle \mathbf{f}(\xi), \mathbf{r} \rangle < 0 \\ 1 & \text{otherwise} \end{cases} \tag{11}$$



(a) No effect of repulsion $c^{\text{rep}} = 1$ (b) Effect of repulsion $c^{\text{rep}} = 2$

Figure 9: Effect of repulsion around an obstacle.

2.2 Dynamic Environments

In environments with moving obstacles the reference frame changes from static to dynamic. This means the movement of the robot must be taken relative to the movement of the object. The system's modulation changes with respect to its relative velocity $\dot{\xi}$. The total local relative velocity $\dot{\tilde{\xi}}^{\text{tot}}$ is subtracted from the dynamic system and is added towards global velocity.

$$\dot{\xi} = \mathbf{M}(\xi) \left(\mathbf{f}(\xi) - \dot{\tilde{\xi}}^{\text{tot}} \right) + \dot{\tilde{\xi}}^{\text{tot}} \tag{12}$$

The total velocity is calculated as the sum of all local relative velocities with their corresponding dynamic weights w_o^l.

$$\dot{\tilde{\xi}}_{tot} = \sum_{o=1}^{N_o} w_o^l \dot{\tilde{\xi}}_o \tag{13}$$

The dynamic weights are defined as a function of the distance to $\Gamma(\xi)$. If the distance is close then that link carries a larger weight.

$$w_o^l = \frac{w_o^l}{\sum_o \tilde{w}_o^l} \quad \text{with } \tilde{w}_o^l = \frac{1}{\Gamma_o(\xi) - 1} \quad \forall \Gamma_o(\xi) > 1 \tag{14}$$

Similarly, the calculation for the relative velocity of a moving obstacle is:

$$\dot{\tilde{\xi}}^v = \dot{\xi}^{L,v} + \dot{\xi}^{R,v} \times \tilde{\xi} \tag{15}$$

Where the linear velocity $\dot{\xi}^{L,v}$, and angular velocity $\dot{\xi}^{R,v}$ are calculated relative to the center point of the obstacle ξ^c, with a relative position defined as $\tilde{\xi} = \xi - \xi c$

The velocity of each agent is limited to its maximum velocity $v^{\max}$.

$$v^n = \langle \dot{\tilde{\xi}}, \mathbf{n}(\xi) \rangle < v^{\max} \quad \text{as} \quad \Gamma(\xi) \to 1 \tag{16}$$

The agent must also prioritize moving away from the obstacle over following the desired path when it is too close to an obstacle. In doing so we introduce a safe velocity $\dot{\xi}^s$ that prioritizes avoidance in critical situations.

$$\dot{\xi}^s = \begin{cases} v^n \mathbf{n}(\xi) + v^e \mathbf{e}(\xi) & \text{if } \left\langle \frac{\dot{\xi}}{\|\dot{\xi}\|}, \mathbf{n}(\xi) \right\rangle < \frac{v^n}{v^{\max}} \\ v^{\max} \dot{\xi} / \|\dot{\xi}\| & \text{else if } \|\dot{\xi}\| > v^{\max} \\ \dot{\xi} & \text{else } \|\dot{\xi}\| = v^{\max} \end{cases} \tag{17}$$

The purpose of $\dot{\xi}^s$ is to prioritize obstacle avoidance over desired motion. In order to do this the it is necessary to modify the velocity all whilst taking into account real life constraints such as the maximal velocity $v^{\max}$.

To modulate the desired motion and repulsion of the agent two velocity components are introduced: a normal velocity to the object v^n, and a tangent velocity v^e. The magnitude of the tangent velocity is defined as $v^e = \sqrt{(v^{\max})^2 - (v^n)^2}$. The normal velocity is capped to $v^{\max}$, so when this is largest the tangential velocity v^e is null. In the case where the agent is far from an obstacle then $\|\dot{\xi}\| = v^{\max}$ and the trajectory sticks to the original DS.

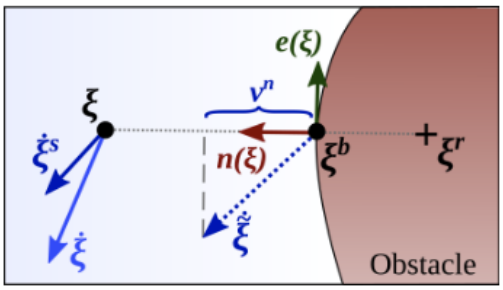


Figure 10: Depiction of new safe velocity $\dot{\xi}^s$.

Obstacle Avoidance with Robots

The algorithm described thus far has been used to represent a point mass. As part of my semester project, the idea was to extend this notion into 3D space. The point mass can be extended to a sphere by generating a margin around all robot links large enough to cover the shape of the robot. The representation of this is shown in Figure 11.

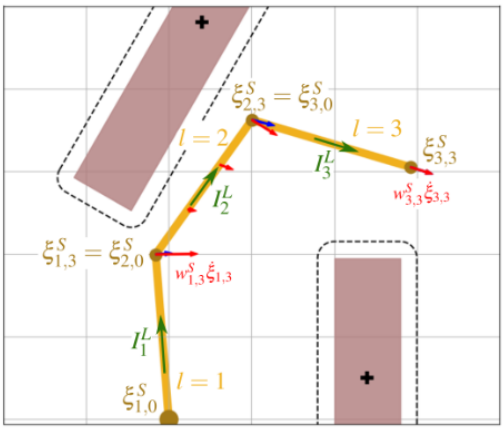


Figure 11: 2D-space model parameters with three link robot arm.

Take the robot end-effector's position as ξ. The desired velocity towards the attractor is noted as $\dot{\xi}$. The inverse-kinematic is used to calculate the goal command in joint space:

$$\dot{\mathbf{q}}^g = \hat{\mathbf{J}}^\dagger(\mathbf{q})\dot{\xi} \tag{18}$$

$\hat{\mathbf{J}}(\mathbf{q})$ is the Jacobian of the robot arm with respect to the position, and $\mathbf{J}^\dagger$ is its inverse.

Danger Field

Consider now a $\Gamma^d(\xi)$ danger field. This field measures the danger an obstacle has to colliding with a robot based on the distance between them.

$$\Gamma^d(\xi) = \min_{o \in [1..N^{\text{obs}}]} \Gamma_o(\xi) \tag{19}$$

The goal command ($\dot{q}^g$) and the avoidance command ($\dot{q}^m$) continuously update their weights based on the proximity of the robot to the obstacle. This field is illustrated in Figure 12.

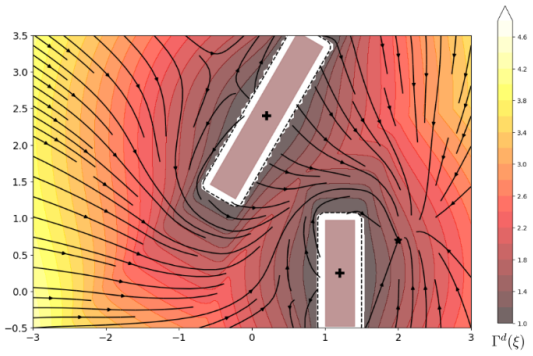


Figure 12: Illustration of the danger field and the direction in 2D space.

Link Avoidance

To ensure that each link avoids collision with its environment a specific avoidance command $\dot{q}^m$ is assigned to each link. To expand the obstacle avoidance to a robotic arm the introduction of N^S section points $\xi^S_{l,s}$ for each link l and $s \in [1...N^S]$. At each section point the dynamical system is evaluated, and paired with section point weight w^S_s. The section point gives priority movement to the sections in highest danger to collide.

For each link the linear and angular avoidance velocity are calculated as follows:

$$\mathbf{v}^L = \sum^{N^S} w^S_s \dot{\xi} S, s \text{ and } \omega^L = \sum_{l=1}^{N^S} w^S_s \left(\xi^S_{l,0} - \xi^S_{l,s}\right) \times \left(\mathbf{v}^S_s - \mathbf{v}^L\right) \tag{20}$$

where $\xi^S_{l,0}$ is the position of the root of a link l and $\dot{\xi}^S_{l,s}$ is the dynamical system-based avoidance described in Section 2.1 *Obstacle Avoidance Formulation*.

Joint Modulation

Using the inverse kinematics we can find the desired joint command. Where $\mathbf{J}\left(\mathbf{q}, \xi^S_{l,0}\right)$ is the Jacobian up the root of the link l, and $\mathbf{J}^\dagger(\mathbf{q})$ is the inverse of the Jacobian.

$$\dot{\mathbf{q}}^m = \mathbf{J}^\dagger \left(\mathbf{q}, \xi^S_{l,0}\right) \begin{bmatrix} \mathbf{v}^L \\ \omega^L \end{bmatrix} \tag{21}$$

This process of evaluating every link starts from the bottom of the robotic arm and is iterated over all the links (N^L). The initial joint control is defined as:

$$\dot{\mathbf{q}}^c \leftarrow \left(1 - \sum_{l=1}^{N^L} w^L_l\right) \dot{\mathbf{q}}^g \tag{22}$$

The first element of the joint command is updated with respect to the avoidance velocity calculated in (21).

$$\dot{\mathbf{q}}^c_{[1]} \leftarrow \dot{\mathbf{q}}^c_{[1]} + w^L_1 \dot{\mathbf{q}}^m_{[1]} \tag{23}$$

The joint command $\dot{q}^c$ is not the same as the idea goal command $\dot{q}^g$ as an effect of the obstacle avoidance at each link. The difference at link l is expressed as:

$$\mathbf{v}^{\Delta} = \hat{\mathbf{J}}_l(\mathbf{q}) \left(\dot{\mathbf{q}}^g_{1:l]} - \dot{\mathbf{q}}^c_{1:l]} \right) \quad \forall l > 1 \tag{24}$$

where $\dot{\mathbf{q}}^c_{[:l]}$ is the current control command calculated up link $l-1$, and $\hat{\mathbf{J}}_l(\mathbf{q})$ is the arm Jacobian with respect to the position up to the link l.

In this scenario the joint speed of the link l that best accounts for this difference is:

$$\dot{q}^{\Delta} = \left\langle \omega^{\Delta}, \mathbf{I}^{\omega} \right\rangle \quad \text{with } \omega^{\Delta} = \mathbf{v}^{\Delta} \times \mathbf{I}^{L} \tag{25}$$

Where $\mathbf{I}^{\omega}$ is the direction of rotation of the joint actuating link and $\mathbf{I}^{L}$ is the direction pointing from one joint to the next. This variable can be seen in Figure 11

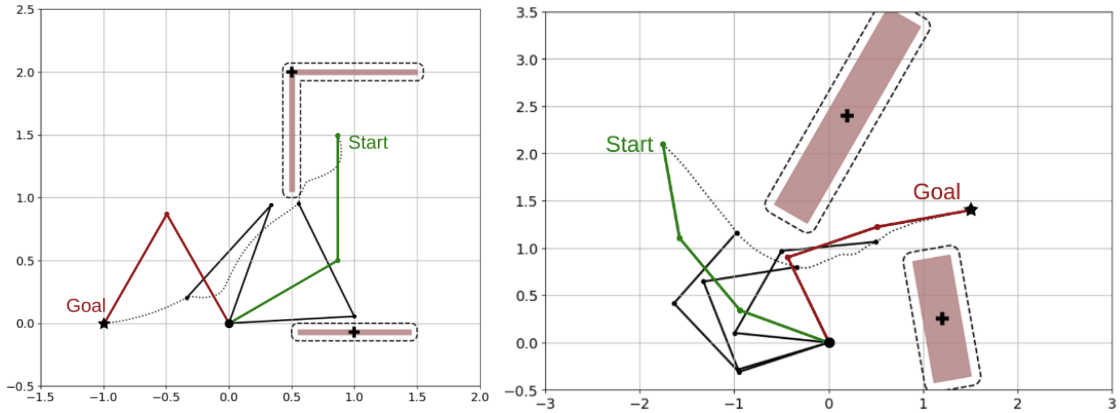
Joint Control Update

The velocity of each joint is adapted one by one starting first with the joint at the base of the arm and up towards the end-effector. The control command $\dot{q}^c$ is updated first with the correction term from (25) The correction term is calculated for each $q^c_{[l]}$ in order that the motion of the previous links are taken into account.

$$\dot{\mathbf{q}}^c_{[l]} \leftarrow \dot{\mathbf{q}}^c_{[l]} + \dot{q}^{\Delta} \sum_{i=1}^{l-1} w_i^L \quad \forall l > 1 \tag{26}$$

Lastly, the modulation command $\dot{q}^m$ from (21) is applied to all the underlying joints.

The algorithm was applied to two planar robotic arms with different degrees of freedom Figure 13. The different time sequences from start to goal can be observed.



(a) Two degrees of freedom.

(b) Three degrees of freedom.

Figure 13: Time sequence of link arms moving in 2D-Space

3 Methodology

In this section, I will describe the necessary setup that was built to test, develop, and run the new algorithm in a three-dimensional environment. The goal was to take the existing algorithms described in Section 2 for obstacle avoidance and expand them to work in 3D space. Unfortunately, the transition was more complicated than expected and was not yet fully implemented. However, a completely new testing environment was built to test and further develop this end goal.

For the transformation from a 2D space to a 3D space bring new constraints to the model. Most notably volume. The control points are now defined as control sphere. These control spheres now need to take the volume of the control joints as to indicate where the robot arm can and cannot move. The number of control spheres directly affect the granularity of motion, more spheres better granularity, but also more calculations. In order to limit the computational cost these control spheres could be increased in size to reduce the number of spheres, but would cover the volume of the robot less tightly, meaning there might be a reduction in the volume of the workspace.

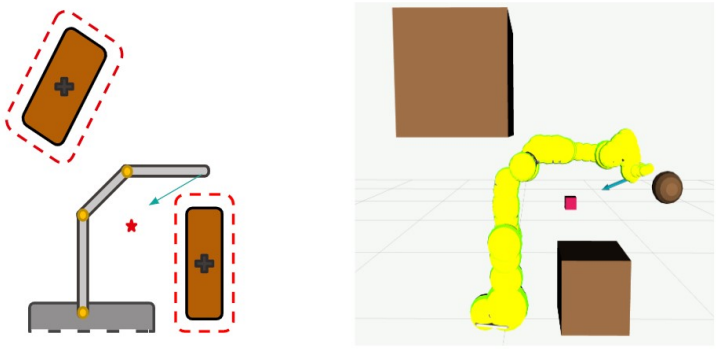


Figure 14: 2D to 3D with schematic..

3.1 Simulation Environment

First the robotic arm Franka Emika was chosen as the robotic subject upon which the algorithm would be tested. Past work in the lab had built an accurate bridge between the Franka Emika and the simulation software PyBullet. This minimizes the later discrepancies that could incur when transitioning from simulation to real life.

Second, a visualization environment was needed for the representation of the dynamic obstacles, their reference and margin points (Equation 2 & 4), as well as the new formulation of the control point and attractor position (Equation 3 & 1). To this end, the visualization software Rviz was used as it provided a computationally light environment to run the animation.

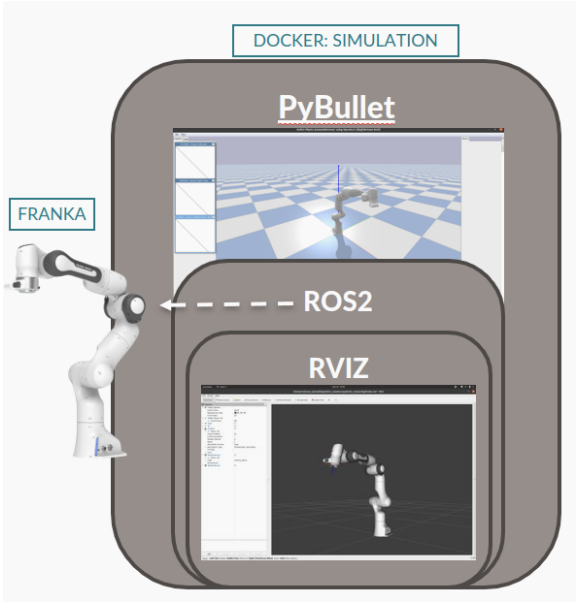


Figure 15: Simulation Environment Setup

Third, ROS2 was selected as the software framework for package management and message transfer between processes. The main simulation process at play was the control and visualization of the robotic arm. It is important to note that despite its low latency ROS is not a real-time operating system (RTOS) [8]. ROS2 addresses the lack of real-time systems with modern libraries for real-time code and embedded system hardware, a feature that was used in our code. One drawback to ROS2 is that its relatively recent deployment has left its online documentation more sparse.

Typically in ROS, you run different modules on several terminals. Our system reduces the need for open terminals by taking advantage of the built ROS2 system for multi-threading. We only utilized one open terminal to run all the simulation and control commands. This is achieved by leveraging ROS2 definition of *executors*. Executors are used to enable the use of one or multiple threads. The advantage of this is that multi-threading allows us to perform real-time computing.

Real-time computing is important in our robotic system since its responsiveness to the environment is crucial for it to avoid obstacles. In motion critical applications, a delay of one or two milliseconds in the system can be enough to cause a serious navigation failure. It should be noted however that the executor overhead in terms of CPU and memory usage is more significant.

Lastly, the simulation and visualization back-end were encapsulated together in an AICA developed docker container. This container nicely encapsuled the application with all the necessary packages that bridged both the Pybullet and Rviz software together.

3.2 Libraries

This project investigates the unification of motion learning and obstacle avoidance. Two accomplish two key libraries were needed to be combined: the Control Library and the Obstacle Avoidance Library. To facilitate the creation of the obstacle avoidance algorithm in 3D the control libraries were chosen from the LASA GitHub repository Control Library[1]. The obstacle avoidance library was coded by me to develop the 3D testing environment. The github repository to my contribution can be found in the github repo. Semester Project LASA [2].

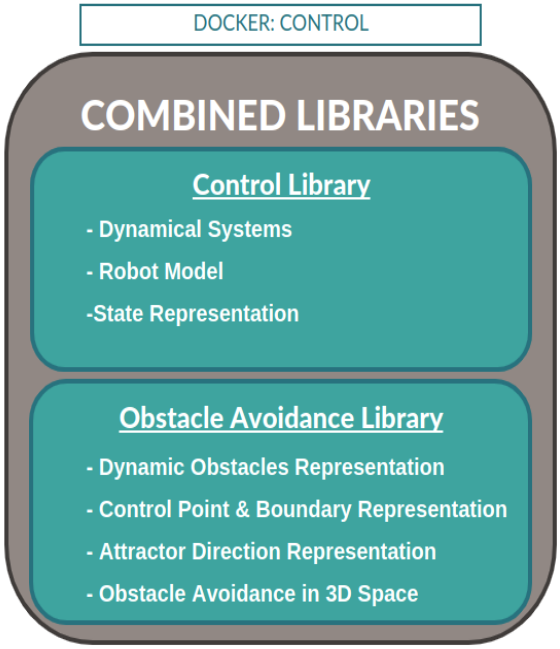


Figure 16: Control and OA Libraries

The **Control Library** was pulled from the LASA's GitHub repository. This library provided the necessary rigid-body algorithm necessary to solve the kinematic and dynamic problems (Equation 1 &18). Furthermore it provides a set of classes to represent states in Cartesian or Joint Space. The classes unify the position, velocity, acceleration and force variable into a consistent internal representation that can later be shared with the Obstacle Avoidance Library.

The second library that was needed was the **Obstacle Avoidance Library**. This library was already proven successful in 2D space but needed to be expanded to work in 3D space. This library builds the 3D environment to test and evaluate the 3D implementation of the 2D obstacle avoidance algorithm. The unification of these two libraries is essential as the final objective is to investigate the unification of obstacle avoidance and motion learning, both of which are built on dynamical systems.

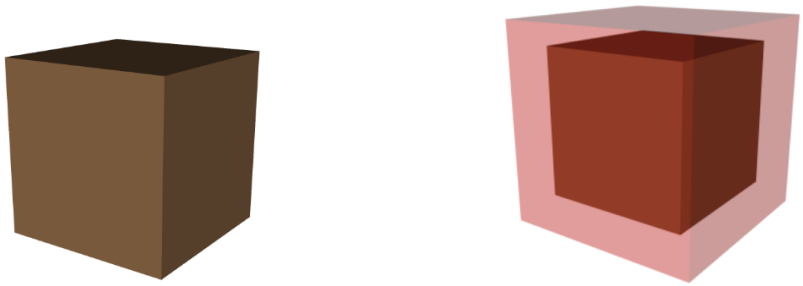
[1] https://github.com/dbdxnuliba/control-libraries_epfl-lasa
[2] https://github.com/ChristopherJulien/Semester_Project_Lasa

4 Results

The following section showcases the three-dimensional adaptations that were made on the Rviz software to test the expanded OA algorithm. The changes were made possible using the framework described above in Section 3. The 3D environment was successfully built using the Rviz as a visualization platform.

4.0.1 Static Obstacles and Margins

Human environments are filled with static obstacles like wall, floors, tables, and ceilings. The 3D algorithm must be aware of these obstacles and avoid them with a safety margin. The safety parameter can be set by how large one sets the margin point (red box), or by the repulsion coefficient (c^{rep}) described in (11).



(a) Static box obstacle.

(b) Static obstacle with margin.

Figure 17: Static obstacle without (left) and with (right) variable margin.

4.0.2 Control Points and Robot Arm Representation

(a) Control Sphere

(b) Robot arm with respective control spheres.

Figure 18: Example of control spheres and segment placement on robot arm.

4.0.3 Dynamic Obstacles

As mentioned before, robots in human-inhabited environments are subject to dynamic perturbances. A person may turn around and walk in the path of motion of the robotic arm. To always ensure human safety, the robot must be able to react to this dynamic change. A test scenario was recreated using spherical balls as moving obstacles. These were given an oscillatory motion through the robot's navigation path. Regardless of which arm link is in course of collision, the robot should adapt its configuration to avoid getting hit.

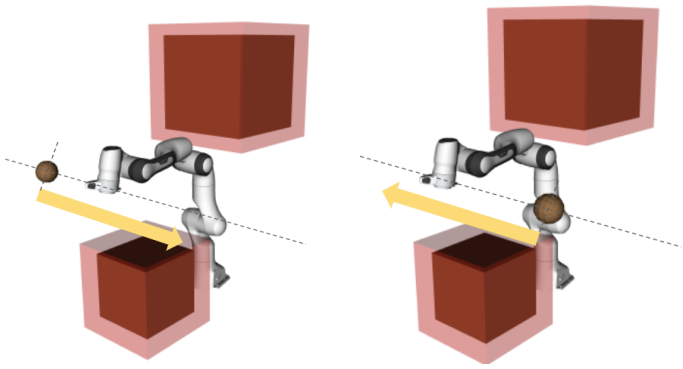


Figure 19: Example of spherical dynamical obstacle.

Attractor and Attractor Vector

The end-effector should always try to reach the target goal (attractor). For this reason, when there is no obstacle present the motion of the end effector should align as closely as possible to the direction of the attractor vector (blue arrow). In our 3D simulation, the attractor is represented as the small pink box.

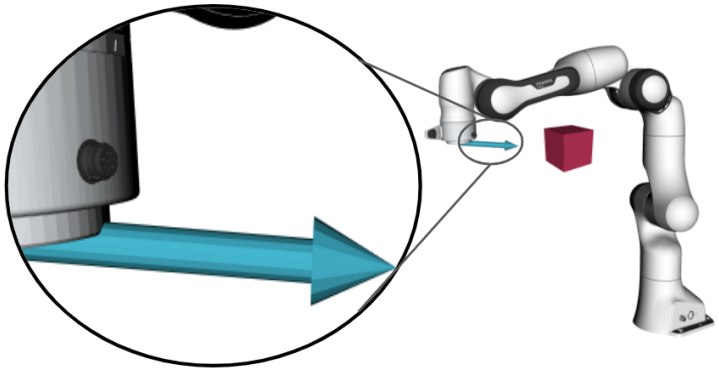


Figure 20: Attractor direction (blue vector) points to the attractor location (pink box).

The Final 3D Setup

Illustrated below we have the final simulation (PyBullet) and visualization (Rviz) software working together. The bridge between the simulation and visualization can be observed. As the robot arm moves to the right on the Pybullet software its 3D replica moves on the Rviz platform while keeping in place all control points and adapting the attractor vector.

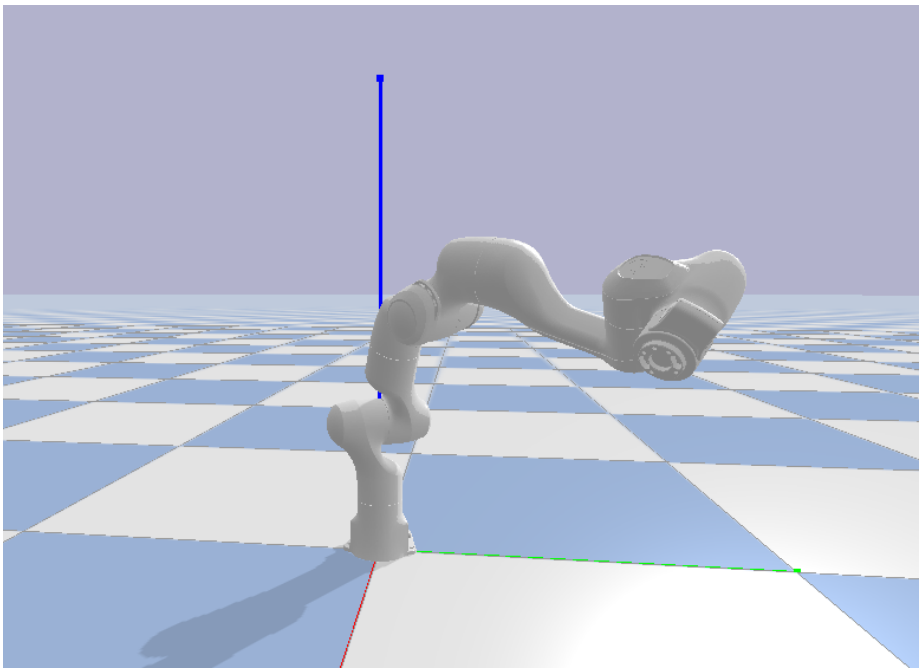


Figure 21: The 3D simulation on PyBullet.

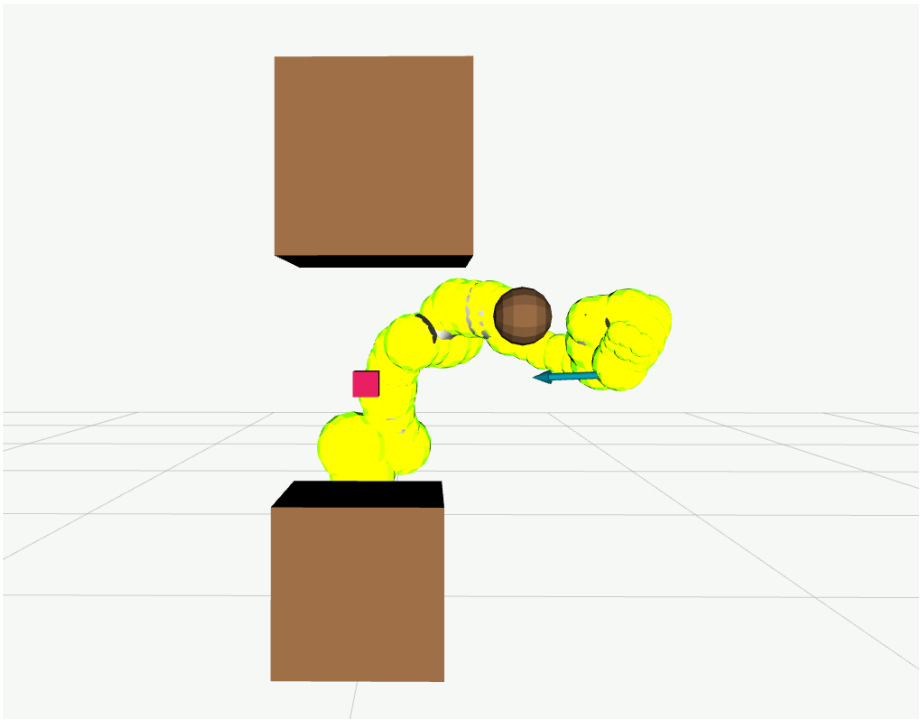


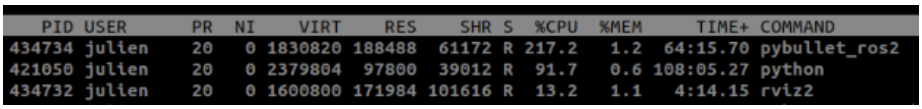
Figure 22: The final 3D visualization on the Rviz.

5 Discussion

The 3D avoidance algorithm is missing to test the simulation on the created 3D environment. The development was made in parallel as the variables and representation of the 3D object were dependent on the visualisation environment.

However, the goals for the simulation framework were reached. The framework's final version of the PyBullet's API class allows to control and monitor the robot in simulation. PyBullet also allows for the implementation of available sub-classes to interface Franka Emika robot ROS2 packages for controlling the real robot. This allows for direct transfer of the code to the real robot, facilitate the future step of transferring the algorithm once ready.

The bridge between PyBullet and Rviz software allows for simple visualization of the obstacles and keeps the processing power low. The computational strain came from ROS2's Real-Time Operating System. RTOS put a heavier burden on the processing power since tasks are now time-sensitive. Despite this constraint, it was possible to simulate on a laptop with (12 CPU(s)). PyBullet used 217.2% CPU while 13.2% Rviz and 91.7% Python. A faster processing power (or dedicated GPU) will be needed once the 3D algorithm is implemented and more obstacles are added to the simulation.

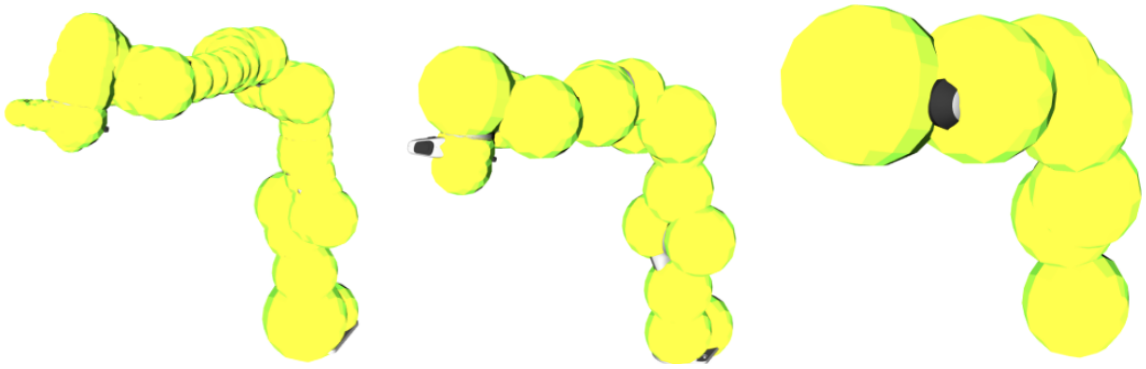


PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
434734	julien	20	0	1830820	188488	61172	R	217.2	1.2	64:15.70	pybullet_ros2
421050	julien	20	0	2379804	97800	39012	R	91.7	0.6	108:05.27	python
434732	julien	20	0	1600800	171984	101616	R	13.2	1.1	4:14.15	rviz2

Figure 23: Comparison of CPU load.

$$\text{Run Time} = \text{Algorithm Loop Execution Time} \times \text{Number of Control Spheres} \times \text{Number of Obstacles}$$

The run time is dependent on the number of obstacles, the number of control spheres, and the algorithm's loop execution time. Depending on the number of obstacles and their velocities one framework with little control spheres could perform better than one with many. For this reason, three different control sphere setups were modeled to test the effect of each one speed, obstacle avoidance, and computational power.



(a) Small control spheres. (b) Medium control spheres. (c) Large control spheres.

Figure 24: Different radius and number of control spheres on the Franka Emika robot.

It is clear that looking at control spheres alone the more numerous the spheres the better the movement acuity however this comes at the expense of larger processing power. Inversely larger control spheres would diminish its movement acuity but could reduce response time. More spheres give also better avoidance granularity by providing individual segments with more evasive motion. The number of control spheres and the approximate total volume of each setup is listed below.

Size of Control Spheres	**Number of Control Spheres**	**Approximate Total Volume[m^3]**
Small control spheres	48	3.334
Medium control spheres	17	0.308
Large control spheres	7	0.792

5.0.1 Next Steps

The next step is to test the 3D algorithm on the 3D simulation environment. The first scenario should test the simplest case scenario of obstacle avoidance with only one static object. This could be a good scenario for calibrating the tuning parameter of the algorithm (i.e. repulsion parameter (Equation 11) and margin parameter (Equation 5)). The next step would be to introduce dynamic obstacles and see how the algorithm performs to varying speeds of the obstacle and varying number of control spheres.

6 Conclusion

The next generation of collaborative robot requires cognisant and reactive automation. These robots should empower human beings and not intimidate them. Hard-coded safety features must make the robot aware of the obstacles in its environment, anticipate collisions and react beforehand to prevent them from occurring all in the laps of a few milliseconds.

This report focused on a nonlinear dynamics system-based algorithm for local navigation. The mathematical frameworks of the avoidance algorithm were explained. The algorithm success on a 2D robotic link, made the next logical step to test it in a 3D setting on a full robotic arm.

To test this both a simulation environment and new avoidance algorithms needed to be made. This project focused on the simulation environment. The software framework for the visualization and robot control were shown. New ROS2 development tools were used to run the visualization on Rviz in real-time operating system (RTOS). The new 3D experimental setup was built with both static and dynamic obstacles. The robot segments were configured with control spheres to be utilized on the obstacle avoidance algorithm. Additionally, the attractor and attractor vector were added to the simulation's environment.

The final success of the project is dependant on a 3D simulation environment as well as the 3D algorithm adaptation. Now that the first half has been built the next steps would be to utilize the new 3D environment to build the analogous mathematical functions from the 2D algorithm (Section 2 *Preliminaries*) to work now in 3D space. Then tests should be performed on the the optimal number of control spheres and their corresponding positioning. These results should give insights as to how many control spheres are needed to reduce computational latency whilst ensuring sufficient reaction time for obstacle avoidance.

A Appendix

The resulting algorithm implemented from Section 2.2 and Section 2.1 is outlined below.

Algorithm

A.1 Algorithm Pseudocode

Algorithm 1: Joint Control Command for a Robot Arm

Input: $N^L, N^S, f(\xi)$ obstacle-environment

Output: $\dot{\mathbf{q}}^c$

$\xi_o^r \forall o \in 1..N^{\text{obs}}$

$\dot{\mathbf{q}}^g \leftarrow (\mathbf{J}(\mathbf{q}))^{\dagger}\dot{\xi}$ (19)

for $l = 1$ **to** N^L **do**

  for $s = 1$ **to** N^S **do**

    $\Gamma\left(\xi_{s,l}\right)$ (19)(20)

    w^{Γ}

  end for

  w_l^L

end for

$\dot{\mathbf{q}}^c \leftarrow \left(1 - \sum_l w_l^L\right)\dot{\mathbf{q}}^g$ (21)

$\dot{\mathbf{q}}^c_{[1]} \leftarrow \dot{\mathbf{q}}^c_{[1]} + w_1^L \dot{\mathbf{q}}^m_{[1]}$ (22)

for $l = 2$ **to** N^L **do**

  if $w_l^L > 0$ **then**

    for $s = 1$ **to** N^S **do**

      compute w_s^S

    end for

    compute $\mathbf{v}_l^L, \boldsymbol{\omega}_l^L$

    compute $\dot{\mathbf{q}}_l^m$

    $\dot{\mathbf{q}}^c_{11:l]} \leftarrow \dot{\mathbf{q}}^c_{[1:l]} + w_l^L \dot{\mathbf{q}}^m_{[1:l}$ (24)

  end if

  $\mathbf{v}^{\Delta} \leftarrow \mathbf{J}_l(\mathbf{q})\left(\dot{\mathbf{q}}^g_{[1:l]} - \dot{\mathbf{q}}^c_{[1:l]}\right)$ (25)

  $\omega^{\Delta} \leftarrow \mathbf{v}^{\Delta} \times \mathbf{I}^L$ (26)

  $\dot{q}^{\Delta} \leftarrow \left\langle \omega^{\Delta}, \mathbf{I}^{\omega} \right\rangle$ (26)

  $\dot{\mathbf{q}}^c_{[l]} \leftarrow \dot{\mathbf{q}}^c_{[l]} + \dot{q}^{\Delta} \sum_i w_i^L$ (26)

end for

References

[1] Executive Office of the President of the United States of America, “Artificial Intelligence, Automation, and the Economy,” *WhiteHouse*, no. December, p. 55, 2016.

[2] McKinsey, “Industrial robotics - Insights into the sector’s grwoth dynamics,” *Mckinsey & Company*, no. July, pp. 1–33, 2019.

[3] M. Koptev, N. Figueroa, and A. Billard, “Real-Time Self-Collision Avoidance in Joint Space for Humanoid Robots,” *IEEE Robotics and Automation Letters*, vol. 6, pp. 1240–1247, 4 2021.

[4] K. Kronander and A. Billard, “Passive Interaction Control With Dynamical Systems,” *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 106–113, 2016.

[5] S. M. Khansari-zadeh and A. Billard, “With Gaussian Mixture Models,” vol. 27, no. 5, pp. 943–957, 2011.

[6] L. Huber, A. Billard, and J.-j. Slotine, “Convergence Ensured Through Contraction,” vol. 4, no. 2, pp. 1462–1469, 2019.

[7] L. Huber, J.-J. Slotine, and A. Billard, “Avoiding Dense and Dynamic Obstacles in Enclosed Spaces: Application to Moving in Crowds,” pp. 1–20, 2021.

[8] “ROS/Introduction - ROS Wiki.”