

**LEVERAGING RUST FOR REAL TIME LOW COMPUTATIONAL
ROBOTICS**

by

GUILHEM MATHIEUX
(M.S., School of Computing)

**A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE (BY RESEARCH)**

in

COMPUTER SCIENCE

in the

GRADUATE DIVISION

of the

NATIONAL UNIVERSITY OF SINGAPORE

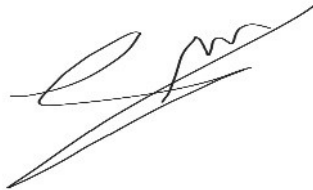
2024

Supervisor:
Professor SHAO Lin

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

A handwritten signature in black ink, consisting of a stylized 'G' followed by a series of loops and a long horizontal stroke.

Guilhem Mathieux

11 November 2024

Acknowledgments

I would like to extend my sincere gratitude to all who have supported me throughout this Master Thesis.

First I would like thank my advisor, Professor Lin Shao, for his constructive feedback and insights that guided me on this journey.

I thank the National University of Singapore and their double degree programme for giving me the opportunity to do this master's thesis.

I would also like to thank my fellow students for their help in this process. Special thanks to Yang Gang for his insights at the beginning of the project.

My deepest gratitude to Télécom Paris and the amazing professors there that equipped me with a strong base of knowledge and skills that were crucial during this research.

Thank you to my friends both back in France and here in Singapore for brightening my life, I truly appreciate each and every one of you.

Lastly, I extend my heartfelt thanks to my family for their unwavering support and belief in me. Their love and encouragement throughout my life have given me the tools to accomplish this endeavour.

Thank you all.

Contents

Acknowledgments	i
Abstract	iv
List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Thesis Synopsis	2
2 Background	3
2.1 Inverse Kinematics	3
2.1.1 General	3
2.1.2 RangedIK	5
2.2 Motion Planning	7
2.2.1 Sampling based motion planning	7
2.3 Rust	8
2.3.1 General advantages of Rust	9
2.3.2 Rust in robotics	9
3 Methodology	11
3.1 Modularity	11
3.1.1 Parameterized objectives	11
3.1.2 Configuration file	12
3.2 Optimizations	13
3.2.1 Frame Computations	13
3.2.2 Partial Gradient	13

3.3	Motion Planning	14
3.4	Visualization framework	15
4	Experimental Result	16
4.1	Experimental Setup	16
4.1.1	Objectives description	16
4.1.2	Other improvements	19
4.2	Results	21
4.2.1	Accuracy	21
4.2.2	Time performance	28
4.3	Limitations	30
4.3.1	IK limitations	30
4.3.2	Motion planning limitations	31
4.3.3	Experimental limitations	31
5	Conclusion and Future Work	33
	Bibliography	35

Abstract

Leveraging rust for real time low computational Robotics

by

Guilhem Mathieux

Master of science (by research) in Computer Science

National University of Singapore

The increasing complexity of modern robotic arms has led to significant increases in computational time, primarily due to the growing dimensionality of the problems they must solve. While this is manageable for applications that can pre-compute movements on powerful computers, it poses challenges for real-time applications, resulting in excessive latency.

In this thesis, we present an efficient, low-computation method for generating manipulator movements by formulating the pose generating task as an *optimization problem* and utilizing *Rapid exploring Random Trees* in order to determine a viable path to the intended goal. The entire project is implemented in Rust, leveraging its safety features and performance advantages.

In order to demonstrate the viability of our system, we conducted extensive empirical testing on a common task: a pre-grasp motion. It allowed us to confirm the ability to generate accurate poses and motions while maintaining low computation times. Our work aims to serve as a milestone for future research in this area and includes tools to facilitate further development, such as a visualization framework and benchmarks.

List of Figures

2.1	Inverse kinematic methods families	4
2.2	Cost functions, reproduced from [34] with authorization	7
3.1	Example of the parameters of an objective in the configuration file . . .	12
3.2	Partial frame example for gradient computation	14
3.3	Simulation control view	15
4.1	<i>UR5</i> with the Robotiq85 gripper attachment	17
4.2	<i>XARM6</i> with the Robotiq85 gripper attachment	17
4.3	Comparison of the cost functions for the Z axis position matching objective	18
4.4	Objective comparison: without (left) vs with (right) horizontal alignment	18
4.5	Objective comparison: without (left) vs with (right) horizontal orientation	19
4.6	Two pose grasp approach - intermediate pose (left) and final grasp pose (right)	20
4.7	Cost visualization for <i>XARM6</i> over the X and Y axis at Z=0.3m . . .	22
4.8	Cost visualization over the X and Y axis shown in 3D	22
4.9	The <i>XARM6</i> in the zero configuration	23
4.10	Cost visualization for <i>XARM6</i> over the X and Y axis at Z=0.3m with the zero starting position	24
4.11	Cost visualization for <i>XARM6</i> over the Y and Z axis at X=0.3m . . .	25
4.12	Cost visualization over the Y and Z axis shown in 3D	25
4.13	Additional motion steps needed for <i>XARM6</i> over the X and Y axis at Z=0.3m	26
4.14	Additional motion steps needed for <i>XARM6</i> over the X and Y axis at Z=0.3m with the zero starting position	27

4.15	Distance (in radians) of the C-Space path for <i>XARM6</i> over the X and Y axis at Z=0.3m	29
4.16	Distance (in radians) of the C-Space path for <i>XARM6</i> over the X and Y axis at Z=0.3m with the zero starting position	30

List of Tables

2.1	Rust crates with application in robotics	9
4.1	Empirical performance	28

Chapter 1

Introduction

Robotics is a booming field, at the frontier between the digital world and the physical world. It is at the core of many current projects, as the digital world becomes more and more entwined with our daily lives. Since the explosion of generative AI, it is considered by many as the next frontier for automation.

Although it is a well explored field, several aspects and key issues are still active topics of discussion. The problem of determining a joint configuration to position the end-effector in a given pose, commonly referred to as inverse kinematics (IK), has been largely addressed. However, many solvers scale badly with large degrees of freedom and don't adapt to additional constraints effectively. Numerous techniques exist, each with their own advantages and disadvantages. As a result, intuitively simple movements like grasp are still being researched and improved on.

Modern robotic manipulators are increasingly designed with higher degrees of freedom, which presents a significant challenge known as the “curse of dimensionality”¹, a 6 dimensions problem being exponentially harder to solve than a 3 dimensions problem. One potential solution is to enhance computational power in order to compensate the increase in number of computations needed, though this often comes at the expense of increased energy consumption and might not be viable on the long term.

This thesis focuses instead on low computation optimizations by attempting to

¹The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings. The expression was coined by Richard E. Bellman when considering problems in dynamic programming [4]

provide a program that can run on a smaller processor and solves those problems in real-time. This system is not designed to provide the best possible trajectory or optimal pose, but simply feasible solutions that are, in most cases, acceptable.

We will investigate how the usage of the Rust programming language can be an effective tool for such endeavour in robotics, as well as how to convert the IK problem to an optimization problem and then establish a possible path to the desired solution. We will focus on a pre-grasp motion for our experimental evaluation. By doing so, we aim to contribute to low level robotics and give a starting point for further development of such projects in Rust.

1.1 Thesis Synopsis

The structure of this research project is organized as follows: In Chapter 2, we conduct a literature review in inverse kinematics and motion planning as well as discuss Rust and its applications in robotics. Chapter 3 details the innovations and improvements of this research. Chapter 4 describes the experimental evaluation for the pre-grasp motion of a bottle. We conclude the entire thesis as well as discuss further directions for future research in Chapter 5.

Chapter 2

Background

2.1 Inverse Kinematics

2.1.1 General

Inverse kinematics (IK) is the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain ¹ in a given position and orientation relative to the start of the chain [19].

It is widely used in Robotics and Computer graphics, and is complex problem with currently three major types of competing solving methods: closed form methods, numerical methods and data driven methods.

¹In mechanical engineering, a kinematic chain is an assembly of rigid bodies connected by joints to provide constrained motion that is the mathematical model for a mechanical system [24]. As the word chain suggests, the rigid bodies, or links, are constrained by their connections to other links. An example is the simple open chain formed by links connected in series, like a serial robot manipulator.

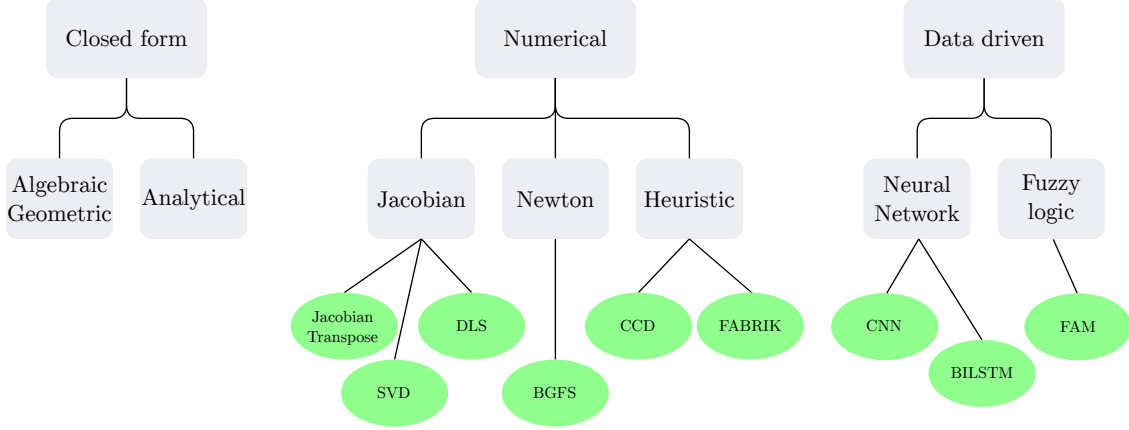


Figure 2.1: Inverse kinematic methods families

2.1.1.1 Closed form methods

Closed form methods (or analytical methods) aim to find exact solutions to mathematical expression. The problem is formulated as a mathematical model derived from the manipulator geometric properties. Solvers based on this approach usually do not suffer from singularity problems and are robust [3]. However, since they are inherently specific to a manipulator they cannot be generalized to all manipulators, as some arm configuration may have no solutions. Additionally, the solutions are independent of the initial pose and can hardly be constrained by additional objectives.

2.1.1.2 Numerical methods

Numerical methods attempt to reach a satisfactory solution by iteratively minimizing a cost function, providing an approximation of the solution. This approach can be subdivided in three families: Jacobian, Newton and Heuristic.

The Jacobian family of numerical methods use the Jacobian matrix to find a linear approximation to the IK problem. This requires computing the inverse jacobian which presents a computationally heavy requirement so most solvers avoid this calculation. One type uses the *Jacobian transpose* instead, which is an easier computation but requires many iterations before convergence. Other rely on an approximation of the inverse matrix using techniques like *Singular Value Decomposition (SVD)* or *Damped Least Square (DLS)*.

CHAPTER 2. BACKGROUND

This family typically doesn't allow for constraints. Most techniques offer trade offs between computational complexity and unpredictable joint configurations.

The Newton family of numerical methods contrary to the Jacobian family, avoid inverse matrix calculation, instead solving convex optimization problems with stochastic objectives and reaching a local minimum using gradient descent algorithms. The *Broyden, Fletcher, Gold-farb and Shanno (BFGS [7])* method is the most widely used.

These types of algorithms typically reach a solution in much fewer iterations but can be difficult to implement and have high computational cost per iterations.

The Heuristic family of numerical methods use simple iterative operations to reach a local minima. The two most popular are the *Cyclic Coordinate Descent (CCD [35])* algorithm and the *Forward And Backward Reaching Inverse Kinematics (FABRIK [2])* algorithm. These benefit from their simplicity but are limited in objectives and constraints (for instance it requires extra steps for orientation constraints).

2.1.1.3 Data driven methods

Data driven *IK* solvers like *Convolutional Neural Networks (CNNs)* or *Bidirectional Long-Short Term Memory (BiLSTM)* [33] have been proved to be feasible. Such models however, remain computationally expensive to train, requiring many iterations in order to achieve sufficient machine learning. The training is also specific to a given manipulator, making them less readily adaptable to a large range of manipulators.

2.1.2 RangedIK

The approach we will focus on and improve in the rest of this thesis is the *RangedIK* algorithm, a type of numerical heuristic *IK* solver. It was first introduced as “RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion” 2018 by Rakita *et al.* [22]. It uses a formulation of the *IK* problem as a weighted-sum non-linear optimization problem. This allows for competing objectives making it a very versatile *IK* solver. The original *RelaxedIK* project was further augmented with collision avoidance in *CollisionIK* [23] and ranged objectives in *RangedIK* [34]. All three projects are open source, focused on real time execution and built in Rust.

We will focus on *RangedIK* since, as of today, it is the more mature version of the algorithm with largest number of features.

The solver takes an End-Effector pose (position and orientation) and an initial pose and finds a feasible joint configuration that can be constrained by objectives². Objectives here range from the common end-effector position and orientation matching to more advanced objectives like optimizing manipulability, minimizing jerk, self-collision avoidance and more. This objective or task approach follows Nakamura *et al.*, 1987 [14]. These authors demonstrated it allows more constraints than the traditional end-effector pose matching.

Moreover this approach enables the developer to combine objectives: *RangedIK* improved on *RelaxedIK* by combining three types of objectives in a single unified framework: (1) a precise goal with limited flexibility, (2) a range of with equally valid goals and (3) a combination of 1 and 2 with a range of acceptable goals but with a preferred goal. This is achieved through three cost functions:

1. Groove for the precise goal relying on a gaussian function
2. Swamp for ranged goal relying on a wall function
3. Swamp Groove for the ranged goal with preferred value relying on a combination of the gaussian and wall functions.

These cost functions also incorporate a polynomial function to provide sufficient derivative when far away from the goal. All the functions can be extensively configured to adapt to specific objectives as shown in Figure 2.2.

The optimization problem generated using this technique is generalizable and a solution can be reached by most solvers but the *RangedIK* prototype uses the *Proximal Averaged Newton-type Method (PANOC [27])*. It has the benefit of being widely used and is implemented in Rust [26].

RangedIK offers a fast and efficient solution for motion synthesis. Its iterative approach, starting from an initial configuration, ensures smooth trajectory tracking. By applying the algorithm to numerous points along a desired trajectory, it generates a sequence of closely related configurations, resulting in fluid motion.

²In the rest of this thesis we will use the term objectives to describe the individual constraints and the term task to describe the higher level goal like grasping.

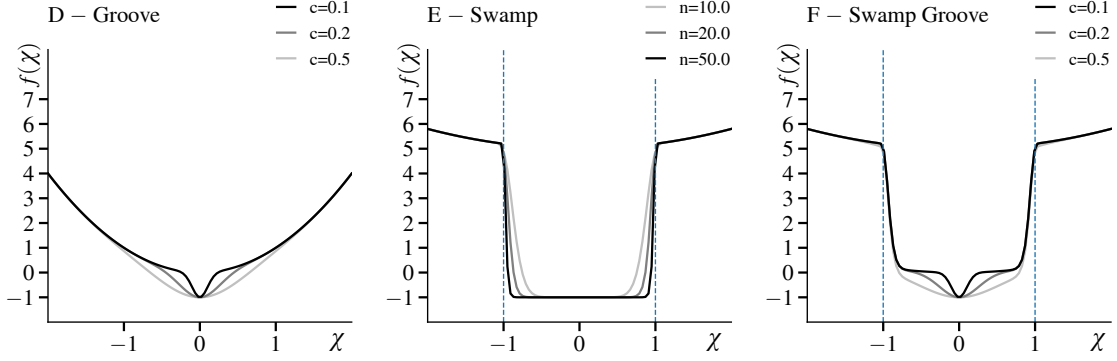


Figure 2.2: Cost functions, reproduced from [34] with authorization

2.2 Motion Planning

Motion planning refers to the process of finding a path between two robot configurations. It is a fundamental aspect of robotics and is often used in conjunction with inverse kinematics to generate feasible motion from the initial pose to the IK generated pose.

This task is a path finding problem -a common problem- with the added complexity of higher dimensionality. Instead of navigating a 3D workspace, the planning occurs within the Configuration Space (C-Space [12]), which represents all possible positions and orientations of the robot.

As a result, the problem quickly becomes too computationally expensive for traditional graph search algorithms such as *Dijkstra* [5] and *A** [8] as the number of joints in a manipulator increases. Many alternatives exist and are described in [36, 11], from neural networks to optimization based method to utilizing convex sets to produce smooth trajectories [1], but we will focus on sampling based methods as they are widely used today and have proven their effectiveness.

2.2.1 Sampling based motion planning

Sampling based algorithms randomly sample points in a fixed workspace (usually fixed by the joint limits). These algorithms can typically produce a feasible path quickly and can then improve the path with further sampling. They will however never ensure optimality of the result, usually generating sub-optimal paths.

The two most commonly used are the *probabilistic roadmap method (PRM)* and the *rapidly exploring random trees (RRT)*.

PRM was introduced by Kavraki *et al.* in [9]. It focuses on static workspaces and uses a learning phase to construct a probabilistic roadmap. The graph obtained during this phase can then be utilized to guide subsequent queries. The roadmap can also be augmented during the query phase adding an interesting learning aspect. This method is less suited for our experiments as we typically have an evolving workspace and couldn't benefit from the learning phase.

RRT was first introduced in 2001 by LaValle and Kuffner Jr in [10]. It incrementally constructs a tree that explores the workspace quickly and uniformly. Each randomly sampled point is connected to nearest reachable point until a path from the starting point to the end goal is reached. It is widely used in robotics for motion planning as it is efficient in high dimensionality problems. Variants of the algorithm have also been developed to improve certain aspects:

- *RRT-bidirectional* provides even faster path finding by running exploring trees simultaneously from both the starting and the end point
- *RRT** incorporates a rewiring step. This enables it to find shorter, more efficient paths, especially as the number of samples grows.

Since our priority is speed, we will use *RRT-bidirectional*.

2.3 Rust

Rust [13, 28] developed at Mozilla Research is a programming language emphasizing, at its core, performance and safety. It is getting more and more popularity as a more modern alternative to C++ without sacrificing on performance and low level control.

It is a statically typed language that doesn't enforce a programming paradigm, but was heavily influenced by ideas from functional programming like immutability, pattern matching and algebraic data types to name a few.

2.3.1 General advantages of Rust

The type system and borrow checker are two core elements of Rust. They make the compiler much more useful by catching a lot of common issues (data-races, data safety). As such, most errors are caught during the compiling stage without the need for any additional testing. As a consequence of this major advantage, a compiling program in Rust will provide much stronger guarantees of safety than most other programming languages. This is not restricting because any code requiring memory manipulation that can't be proven to be safe by the compiler, can be isolated as an unsafe part of the program without losing the safety guarantees on the remainder of the program.

Rust also benefits from a powerful package manager, *Cargo* [30], making dependencies in Rust much easier to install, upgrade and specify than other languages that rely on external tools. the *Cargo* ecosystem is very developed, with a lot of open source crates (libraries) and more projects being launched on a daily basis.

Other aspects like built-in concurrency support, precise memory control, pattern matching make it a greatly versatile language, that can be (and is being) used in a large variety of domains, from web services to systems programming.

2.3.2 Rust in robotics

There is an increasing community support for robotics in Rust in the form of open source crates. Table 2.1 presents the main ones we have used in this project.

Crate name	Description	Source
nalgebra	Linear algebra library	[6]
OpEn	Optimization Engine solvers	[26]
OpenRR	Rust robotic platform	[15]
→ k	Kinematics library	[16]
→ urdf-rs	URDF parser	[17]
→ urdf-vis	3D URDF visualizer	[18]

Table 2.1: Rust crates with application in robotics

The previously mentioned advantages of Rust in terms of run-time performance are vital for real-time robotics applications. As a compiled language, Rust can be seamlessly ported to resource-constrained embedded systems, taking full ad-

CHAPTER 2. BACKGROUND

vantage of target-specific optimizations. In performance-critical environments, a bare-metal approach—where no operating system is used—can eliminate OS overhead, maximizing system efficiency. Rust is particularly well-suited for bare-metal environments, offering low-level control and fine-grained memory management, all while maintaining safety and performance.

This makes Rust a key part of future robotics development.

Chapter 3

Methodology

The core of this project has been to provide a unified and optimized robotics framework in Rust. To achieve this goal, we have added new developments to the *RangedIK* library with further optimizations, increased capabilities and included a motion planning algorithm. All code further discussed in the rest of this thesis is provided as an open source implementation ¹

3.1 Modularity

3.1.1 Parameterized objectives

The goal of this program is to provide a system adaptable to a large range of tasks. As a result, it requires adaptability, mainly on the objectives. For instance, a cylindrical object pre-grasp and a rectangle object pre-grasp will require different objective configuration. Objective construction was then modified to be completely dynamic. We created a list of objectives and for each one users specify the weight of the objective and the loss function parameters.

As of now, the objectives are created based on the configuration at the start of the program and not modified afterwards but it could be easily modified to change the objectives during code execution. One could also add a few selected tasks in the form of a list of objectives built during the initialization phase that could then be selected for the IK calculation at different stages. For example we could imagine loading both a top grasp and horizontal grasp tasks and switching from one to the

¹github.com/guimath/relaxed_ik_pp

other when one is not feasible. This opens the possibility to combine tasks with completely conflicting objectives that wouldn't result in a good solution if run as single problem (e.g. combining side-grasp and top-grasp).

Establishing parameters for the objectives didn't result in significantly longer run times but could be further optimized without losing the general versatility of the system by using procedural macros [29] to parse a configuration and build the objectives at compile time. This is an area open for trial which could result in even faster initialization phase and could provide some optimizations on the loss functions computations.

3.1.2 Configuration file

To improve the configuration file, we switched from the *YAML* format used in *RangedIK* to the *TOML* format. *TOML* (Tom's Obvious Minimal Language) [21, 20] is a popular configuration file format that is for example used as the manifest file for Rust package manager *cargo* [30]. As its name suggests it is a very simple format that is clear, easily readable and editable by humans and machines providing a good alternative to more heavy syntax formats like *YAML*.

This file can be quickly modified to suit a given project needs and can also be enlarged with more fields if required for minimal code changes (usually simply changing the rust structure that corresponds to it should suffice as the parsing is derived from the structure description but additional logic might be needed on edge cases).

As of now, the file is set up to specify objectives configuration, the unified robot description format (*URDF*) file of the manipulator, the used joints of the manipulator and their starting values (among other).

```
1 | [objectives.z_pos]
2 | func= {Swamp={l_bound= -0.1, u_bound= 0.1, f1= 2.0, f2= 0.2, p1= 20}}
3 | weight= 70
```

Figure 3.1: Example of the parameters of an objective in the configuration file

3.2 Optimizations

A key focus of the project was to enhance the system’s responsiveness to minimize downtime in real-time environments. This involved refining the code and implementing several optimizations.

3.2.1 Frame Computations

All fixed translation or rotation in a serial manipulator are computed during the initialization phase. Previously, fixed joints would be computed as a normal joint with fixed value, forcing the system to compute the offsets every time. This optimization is somewhat minor as it is uncommon to have many fixed joints in the *URDF* description but it can substantially improve run time on a larger Degrees of Freedom (DoF) manipulator where some joints are fixed to limit the dimensionality of the problem.

It also doesn’t require lengthy computation during the initialization stage so it offers benefits without any downsides.

3.2.2 Partial Gradient

During the IK phase, multiple cost gradients are computed to direct the gradient descent. That involves assessing how a minor change in each joint affects the overall cost. Calculating the gradient requires computing all joint frames and the objective components, multiplied by the number of joints, making the frame calculation very frequent.

The underlying idea is that for serial manipulators, parts of the calculations are redundant and can be memorized.

The optimization then only applies to serial manipulators: We first compute the frames at the current joint configuration and then reuse this pre-computed frame for all joints leading up to the one with the delta.

In theory, for an arm of n -DoF, this translates to a complexity of $\mathcal{O}(n \log n)$ instead of the original $\mathcal{O}(n^2)$. In practice the improvements are not as drastic but are still consequential, achieving about a 2x speedup factor in frame calculation for a 6-DoF manipulators instead of the expected ≈ 3.3 x factor.

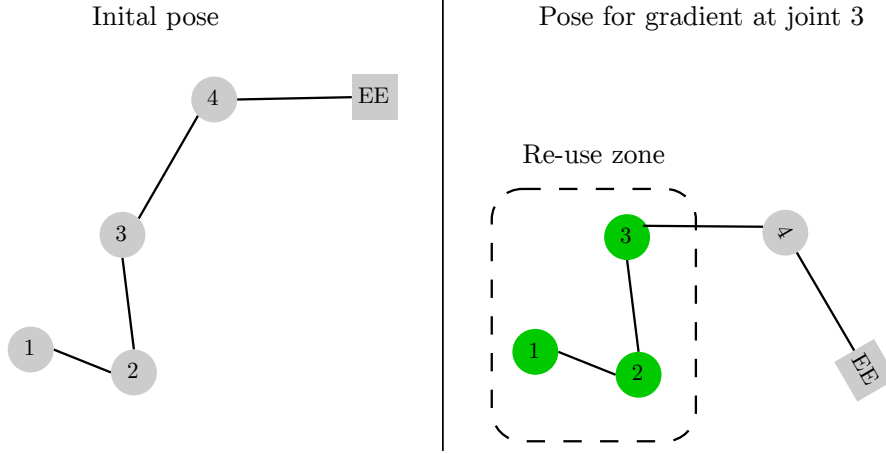


Figure 3.2: Partial frame example for gradient computation

3.3 Motion Planning

We added a motion planning algorithm directly to the library, allowing the library to be used completely standalone from other robotics software. The goal is to be able to run all the control on an embedded target thus integrating the motion planning algorithm is crucial as many motion planners software are not designed for such systems.

We decided on the *RRT* algorithm, more precisely the *RRT-bidirectional* algorithm. As mentioned in § 2.2.1, this path finding algorithm is efficient, fast and widely adopted. It has an implementation in rust by *OpenRR* project. For the implementation, we integrated the *openrr_planner* which abstracts the actual *RRT* algorithm, creating from the joint limits the random sample function and from the obstacles description the collision detection. We only use the motion planning part of this library to get from one configuration to the other as the pose generation is done in our IK solver.

This uses the same libraries for robot representation and parsing of *URDF* files, offering a guaranty of a uniform interpretation of our workspace environment description.

3.4 Visualization framework

Simulation is a crucial part of robotic development. It is always helpful to visualize a rendering of the motions of the manipulator in its scene to verify that the *URDF* is correct. We used extensively such a visualization device to test our system, to visualize grasp pose, motion and to tune our parameters.

To achieve this, we created a framework utilizing *urdf-viz*, an open source library for 3D robot visualization. This framework offers essential controls in simulation, enabling users to move the target in 3D, compute an IK, show the generated pose or a full move at the given target and reset the manipulator to initial pose, among other features. The code is designed to be adaptable to project specific needs, other commands can be easily added as needed.

All virtual visualizations of manipulators presented in this thesis were generated using this framework.

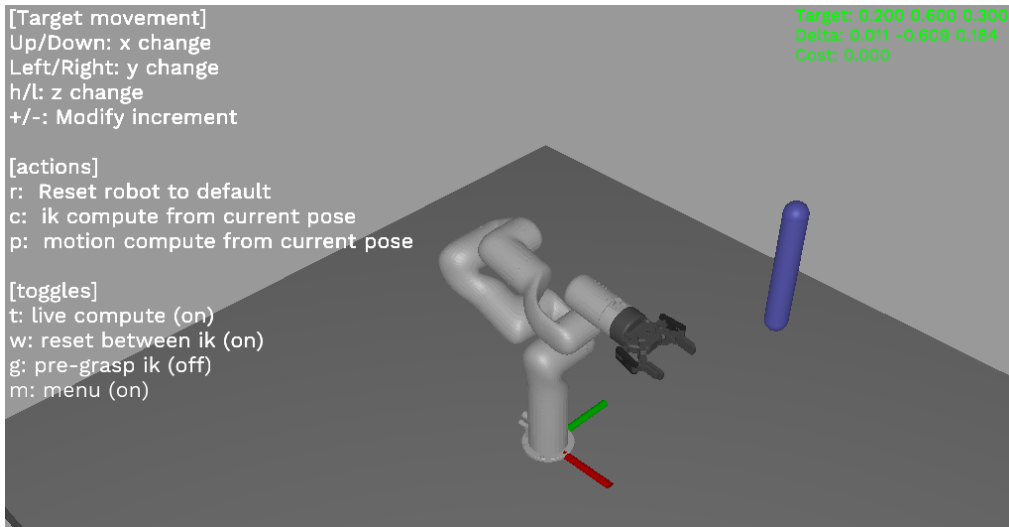


Figure 3.3: Simulation control view

Chapter 4

Experimental Result

4.1 Experimental Setup

For the experimental prototype, we focused on a pre-grasp motion. Our task is to take a manipulator description, the starting joint position, and the target object position and to generate a complete movement plan which will enable the arm to reach the object for grasping, while ensuring there are no collisions with the surrounding environment. To demonstrate the capability of our system, we used a bottle (represented by a cylinder) with many acceptable grasp pose.

All our algorithms are designed to be flexible and not specific to any one robotic arm, allowing them to adapt to a wide variety of robotic arms with different configurations and DOF. We tested two widely used 6-DOF robotic arms: the *UR5* by *Universal Robots* [32] and the *XARM6* by *Ufactory* [31]. Each was combined with the *robotiq85* gripper [25] as it is among the most popular two finger gripper, but again, any gripper should be easily added.

Most of our evaluation was conducted in simulation, with some additional testing carried out in real life on the *XARM6*.

4.1.1 Objectives description

The point of the objectives approach is to be versatile adapting to completely different tasks, so task specific objectives were created for our pre-grasp. We focused on a side grasp task, for which we didn't use orientation matching objectives as many different orientations could produce good grasp poses.

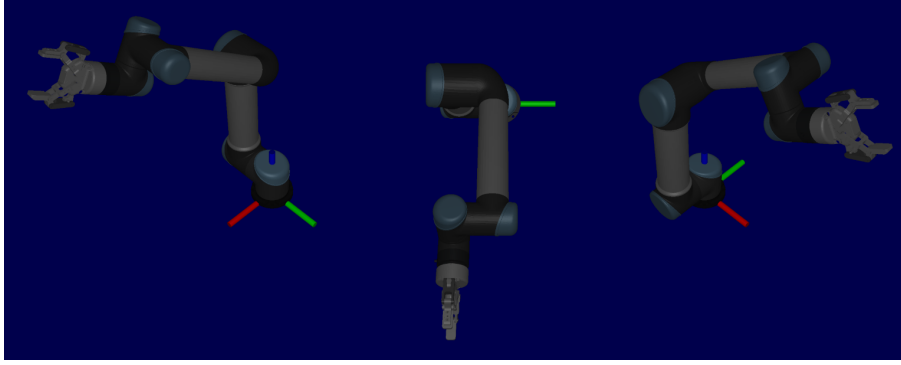


Figure 4.1: *UR5* with the Robotiq85 gripper attachment

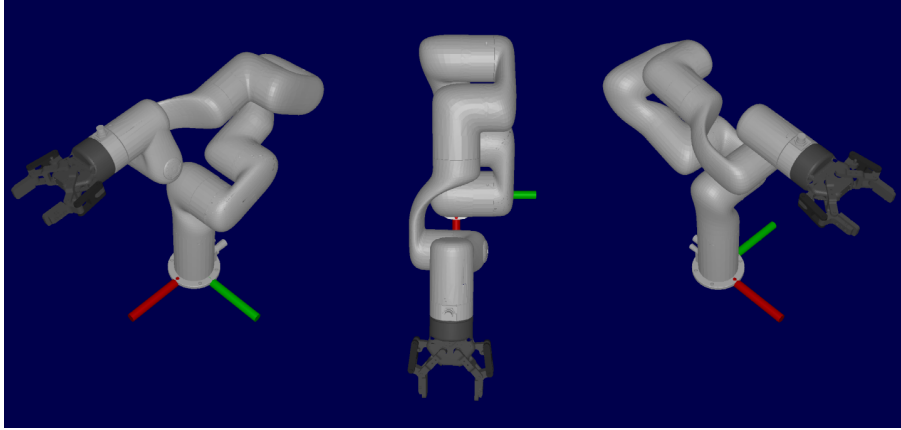


Figure 4.2: *XARM6* with the Robotiq85 gripper attachment

There were three main changes to the *RangedIK* base objectives.

- (1) Making the z axis position of the end effector have a range of acceptable values giving more flexibility to the robot.

For a side grasp of a bottle, the gripper doesn't have to be perfectly centered in the Z axis. This can be accomplished by changing the cost function without modifying the position matching objective code, meaning that only the configuration file needs modification.

For those objectives, the cost function is applied to the delta between the goal and the end-effector at the given axis. A range of acceptable positions around the center can then be implemented by using a Swamp function instead of a Groove function. The two functions used for position matching are compared in Figure 4.3.

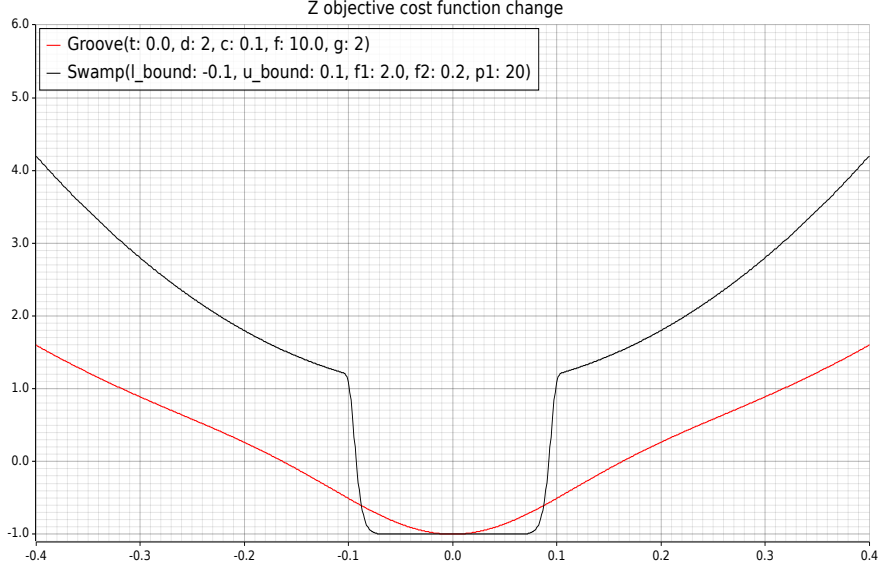


Figure 4.3: Comparison of the cost functions for the Z axis position matching objective

(2) Adding an alignment objective.

We are considering an upright bottle so in order to facilitate an easy grasp, the end-effector should be positioned perpendicular to the bottle. This requirement can be encapsulated by a new objective using the delta between the z-positions of the end-effector and the preceding link as input for a groove cost function centered at zero. Since the frames for each joint have already been computed for self-collision avoidance, the additional computation for this objective is minimal.

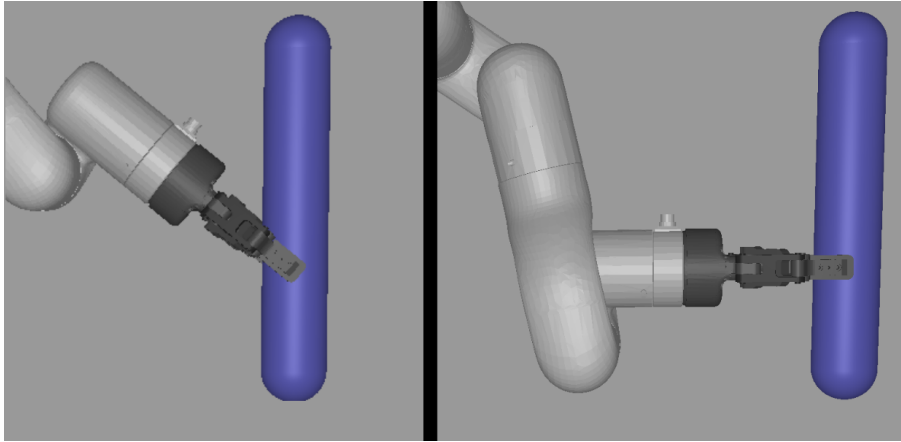


Figure 4.4: Objective comparison: without (left) vs with (right) horizontal alignment

(3) Adding a horizontal gripper orientation objective.

This could vary depending on the type of gripper used (e.g. might not be needed for a suction based griper). In the case of a two finger gripper, we want to have each finger on either side of the bottle. To achieve this, we constrain the orientation of the end-effector, more precisely we use the y euler angle as input for a groove function centered on 0.

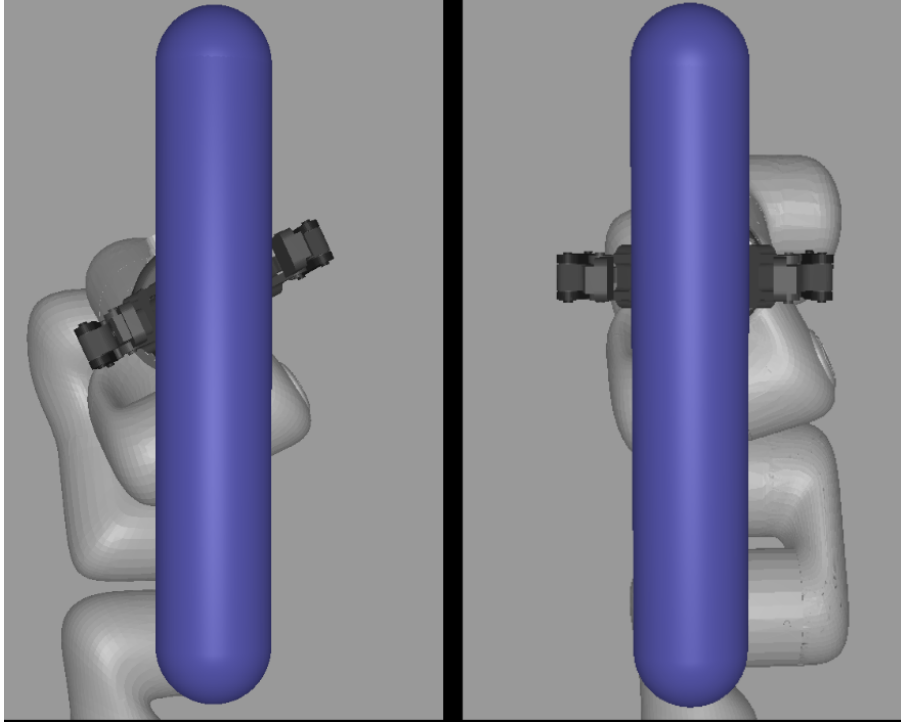


Figure 4.5: Objective comparison: without (left) vs with (right) horizontal orientation

4.1.2 Other improvements

In order to limit the complexity of the motion planning, we actually compute two IK poses: one slightly away from the grasping object, and then the actual grasp pose using the previous pose as the starting pose. The first pose is computed with the same objectives as the final one, with an additional distance on the end effector center point. This distance is entered as a parameter and for the purpose of the test on the *XARM6*, we use 5cm.

This approach is used because the initial pose is typically easier to reach, as it is positioned further away from the grasping object and has fewer collision constraints.

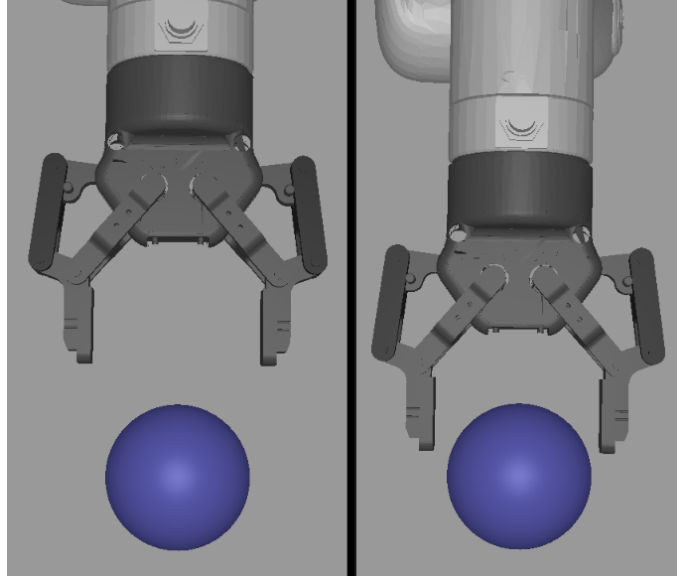


Figure 4.6: Two pose grasp approach - intermediate pose (left) and final grasp pose (right)

Reaching the final pose from the intermediate pose is then straightforward since minor movements don't require additional motion planning steps [34]. The second IK solve is also much faster to solve as less iterations are needed with less joint modifications.

4.2 Results

To evaluate our algorithm we have two main metrics: accuracy and efficiency. Accuracy refers to the quality of the grasp pose generated and the motion. Efficiency refers to the code run time and the movement time.

All evaluations were performed on an 12th gen Intel Core i5-1240P mobile chip with 16 GB RAM.

4.2.1 Accuracy

Accuracy is not easily quantifiable. In order to obtain a metric for our results, we used the global cost of the reached pose for the IK accuracy. By comparing the global cost to the minimum possible cost, we have an approximation of the accuracy of the generated pose.

Regarding movement, we have selected the full completion of movement as success factor and the number of additional steps needed to reach it.

Because we want the system to be reliable, we tested it on a wide area of 3D space. For this we sampled slices of 3D space, with more then 50000 points of target positions, using the same initial pose for each calculation.

4.2.1.1 Pose accuracy

To visualize the results of the space scanning, we produced the heatmap in Figure 4.7. Each point of the heatmap represents the cost at the second ik, meaning the final pose, just before the grasp. It is shown in scale from green to red. The scale has no units as it measures global cost, or more precisely the cost differential with the minimum possible cost of -246.20. The scales caps at 50, which in our experiments is the limit for a decent grasp pose. We ignore any point further than 1 meter from the base of the robotic arm, as any object above this threshold is out of grasp range. It is also shown in 3D space with the manipulator in Figure 4.8 for a clearer spatial understanding. There are a around 75000 sampled points in the graph.

CHAPTER 4. EXPERIMENTAL RESULT

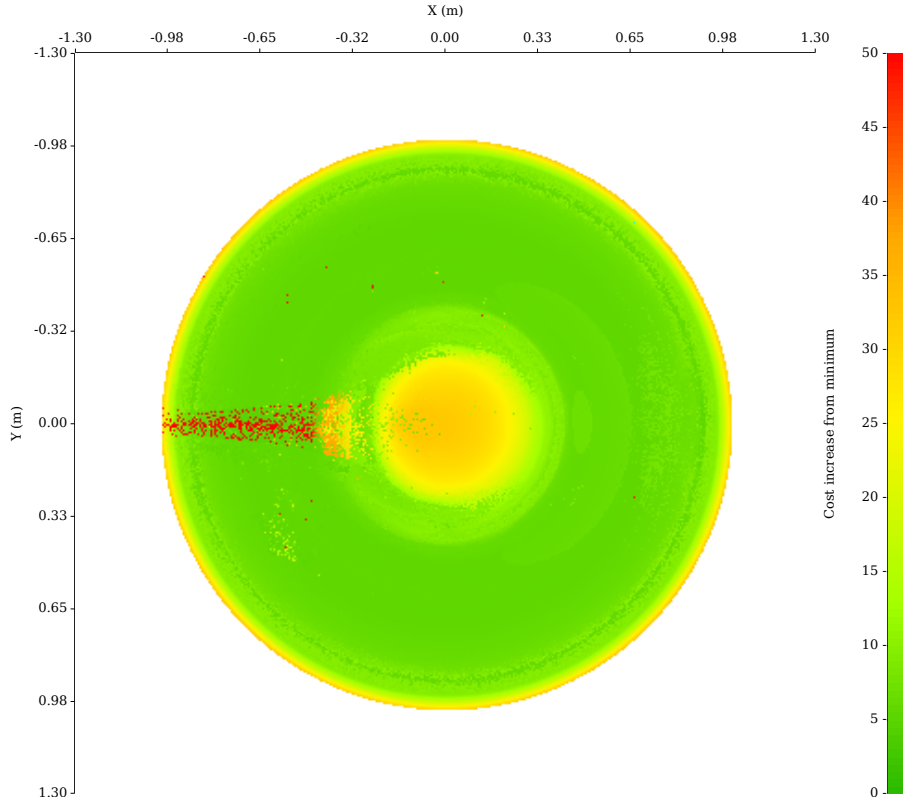


Figure 4.7: Cost visualization for *XARM6* over the X and Y axis at $Z=0.3\text{m}$



Figure 4.8: Cost visualization over the X and Y axis shown in 3D

Figure 4.7 shows us that the large majority of the range of motion produces a satisfying grasp pose. There are two main problematic areas:

(1) the center of the graph. This is expected as the lack of manipulability so close to the base is a known limitation of robotic arm. This is not specific to our

system and is not an overall problem anyway since the object to be picked up is typically not in this zone.

(2) The back of the graph. Directly behind the robot is another challenging area, requiring the most changes in joint configuration, which makes the optimization problem much harder to solve. This in turn raises the question of the initial pose. Indeed changing the starting state of the problem change the result. Figure 4.7 was obtained with the initial pose of $[0.0, -1.2, -0.15, 0.1, -0.1, 1.57]$ (in radiants), shown in Figure 4.2 which we will refer to as the *XARM6-startpose1*. We have tested other starting poses, most notably the *XARM6-startpose0* , shown in Figure 4.9, the pose reached with all joint values set to zero.

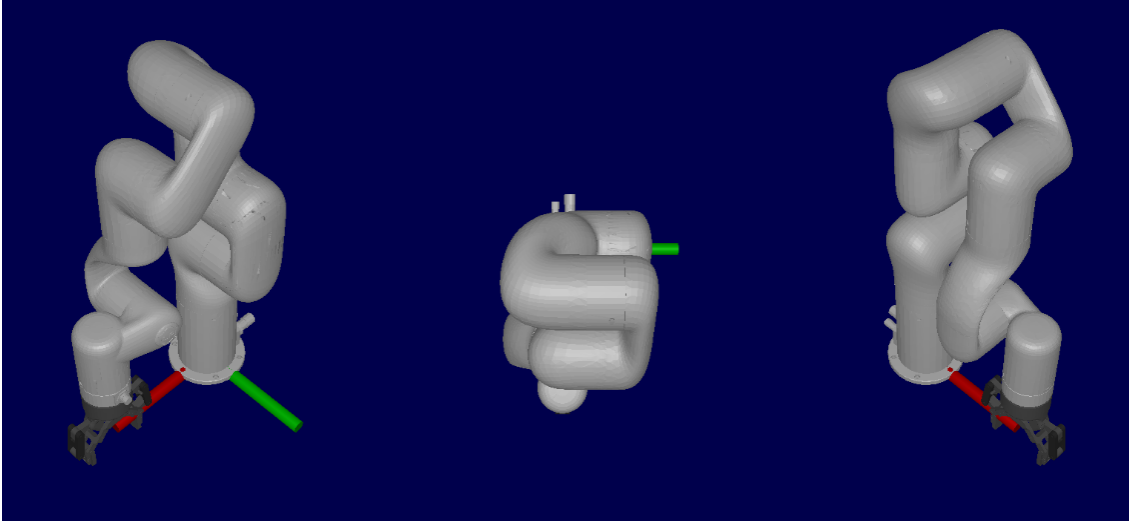


Figure 4.9: The *XARM6* in the zero configuration

A similar heatmap, using the *XARM6-startpose0* as the starting position is shown in Figure 4.10. This starting pose may limit the system's ability to reach a wider range of targets, but it can still achieve a good grasp within the semicircle in front of the robot. Our testing suggests that similar guarantees apply to most acceptable starting positions. This is not too limiting as one can generally expect the target of the grasp to be in front of the robot.

CHAPTER 4. EXPERIMENTAL RESULT

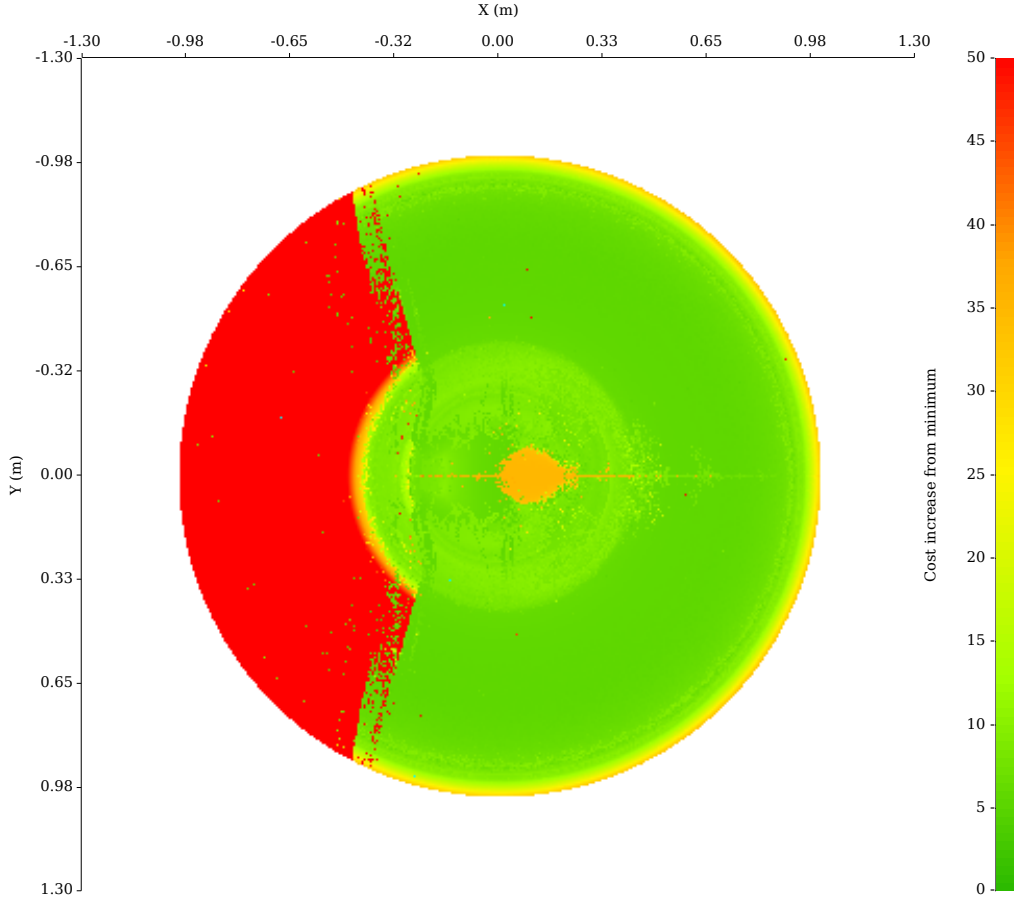


Figure 4.10: Cost visualization for *XARM6* over the X and Y axis at $Z=0.3\text{m}$ with the zero starting position

We also get good results with different heights as seen in Figure 4.11

The quality of the grasp pose generated was also visually empirically tested, by utilizing our visualization framework to quickly sample wide a range of random poses. This visual inspection confirms the previous findings.

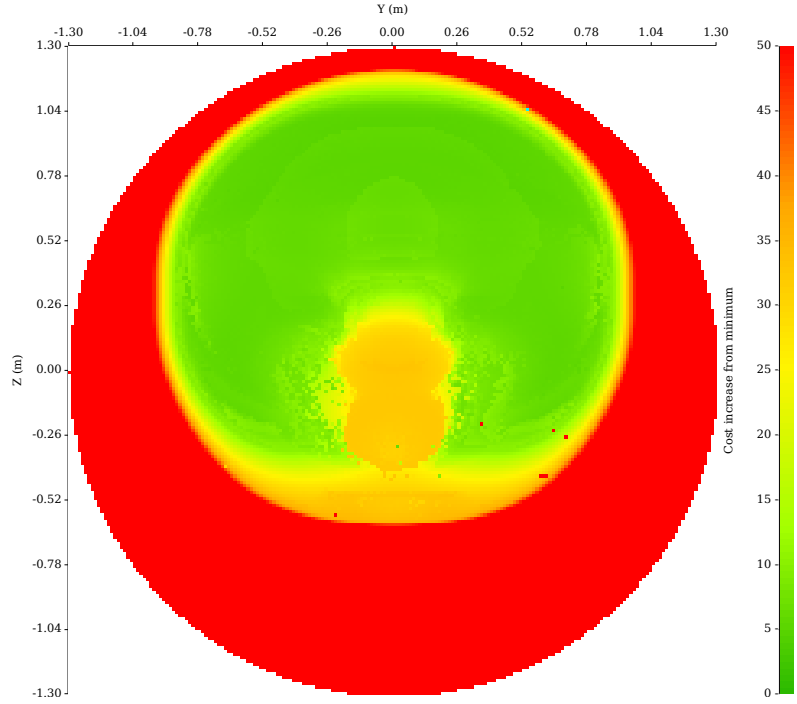


Figure 4.11: Cost visualization for *XARM6* over the Y and Z axis at $X=0.3\text{m}$

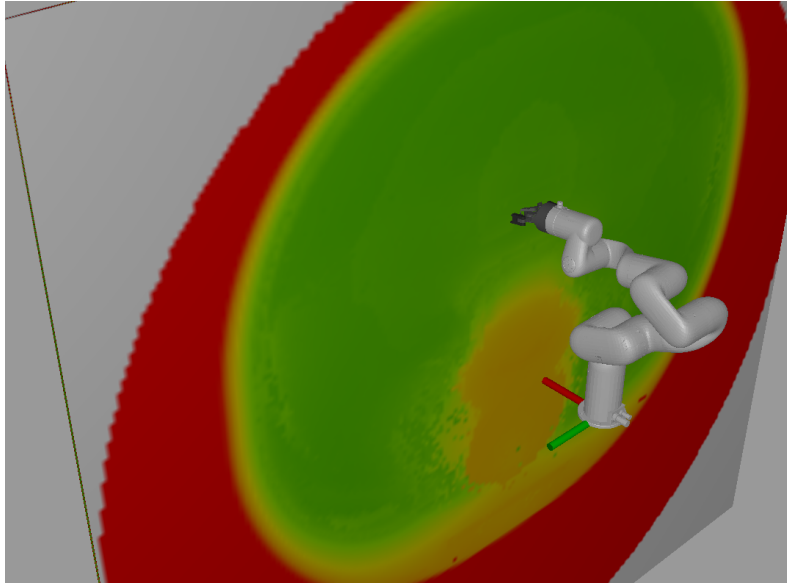


Figure 4.12: Cost visualization over the Y and Z axis shown in 3D

4.2.1.2 Motion Accuracy

AS was done to show the pose accuracy, we can visualize the motion accuracy using a heatmap. The color is no longer a representation of the cost of the pose

CHAPTER 4. EXPERIMENTAL RESULT

but instead a representation of the number of additional steps required to reach the grasp pose. Thanks to the two pose approach, most targets can be reached without any additional steps, so in only three steps. But some poses need steps in between, usually to avoid collisions.

As before, we limit the graph to the range of motion of robot, ignoring any point further than 1 meter or closer than 0.2 m from the base.

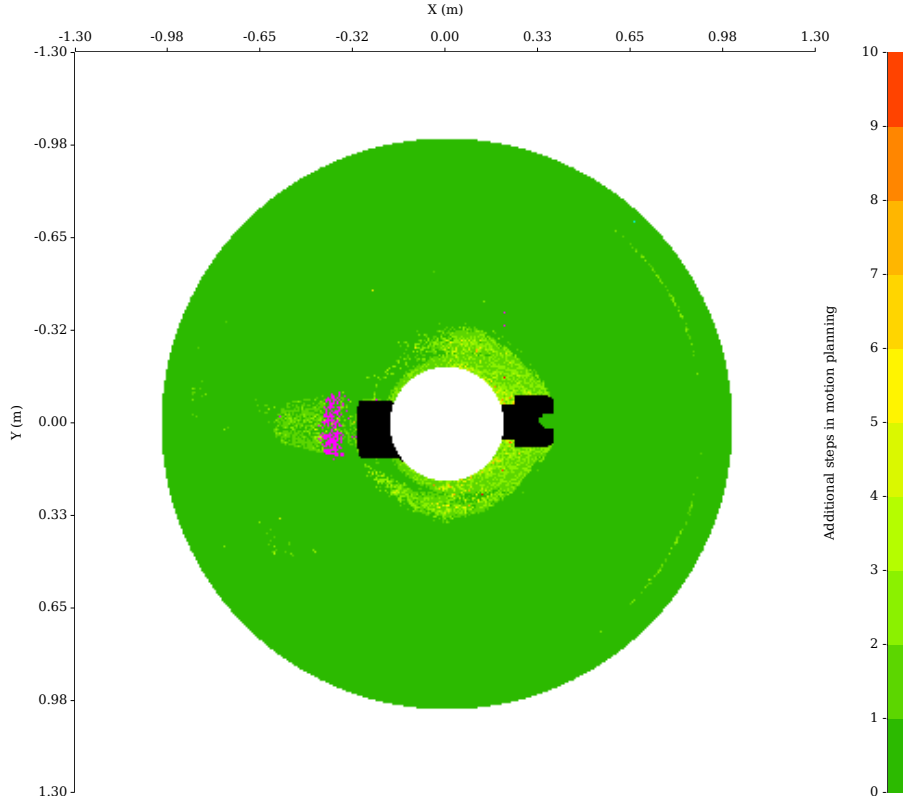


Figure 4.13: Additional motion steps needed for *XARM6* over the X and Y axis at $Z=0.3\text{m}$

Figure 4.13 shows that the vast majority of the targets are reached without extra steps. This is by definition one of the shortest C-Space path to go from the initial pose to the intermediate pose and finally to the grasp pose and is generally close to the shortest path from initial pose directly to the grasp pose.

Even when extra steps are needed, only a few are required and the algorithm still produces a working path which allows to get to the grasp pose for all targets in the range of motion (except for some points excluded because of collision between the object and the robot).

CHAPTER 4. EXPERIMENTAL RESULT

This is also holds true for other pose including the *XARM6-startpose0* as can be seen in Figure 4.14 where only edge cases require additional motion planning steps.

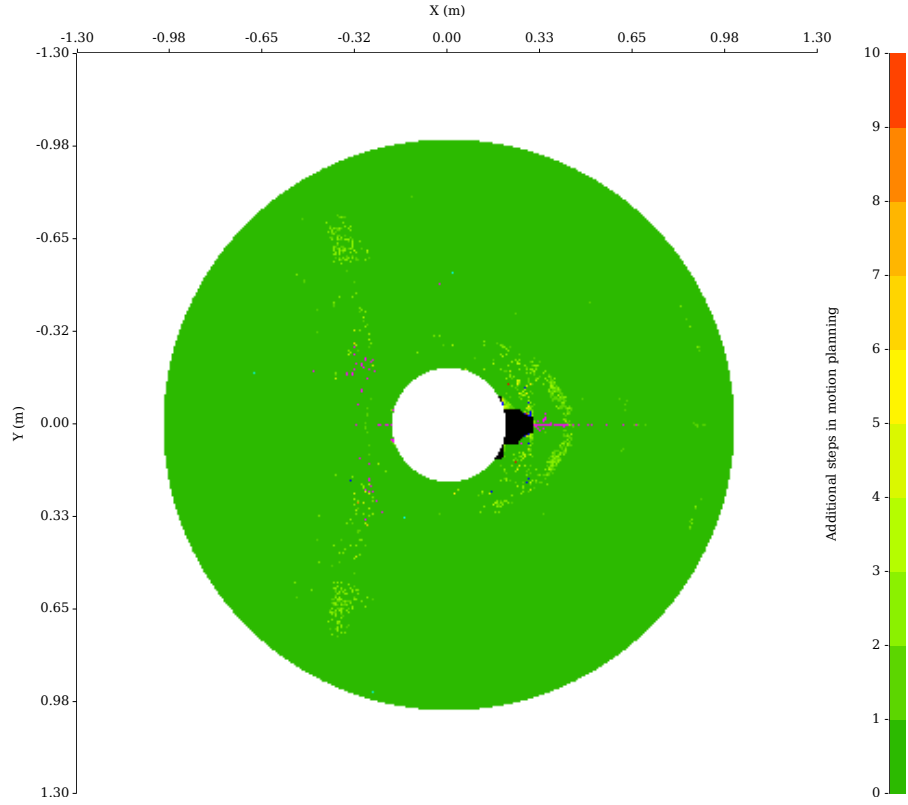


Figure 4.14: Additional motion steps needed for *XARM6* over the X and Y axis at $Z=0.3\text{m}$ with the zero starting position

4.2.2 Time performance

4.2.2.1 Computation time

Results discussed in this section would require further testing, and could be further improved by target specific or compile level optimizations already mentioned in § 3.2.

This program has not yet been tested on a embedded hardware which is the target use case of the system. Therefore we have chosen to provide an initial assessment of the computation time performance using a regular computer processor.

Arm	Single IK grasp pose	Two IK grasp pose	Motion plan
<i>XARM6</i>	$1.92 \pm 0.59\text{ms}$	$2.50 \pm 0.92\text{ms}$	$9.31 \pm 69.99\text{ms}$
<i>UR5</i>	$2.42 \pm 0.55\text{ms}$	$2.76 \pm 0.62\text{ms}$	$13.82 \pm 64.71\text{ms}$

Table 4.1: Empirical performance

These values have been obtained over a large (around 75000 individual samples) sampling pool of target within the 3D range defined by the coordinates (0, -1, 0.3) to (1, 1, 0.6).

Pose generation exhibits a negligible latency, consistently completing in less than 3 milliseconds. As for motion planning, the vast majority of the targets position are handled in under 10ms (92% for the *XARM6*, 90% for the *UR5*), but some more complicated motions require a lot of RRT sampling to find a path, taking sometimes more than 1 second to complete. This is very rare in our sampling range with 65 positions resulting in more than 1 second for the *XARM6*, so less than 0.1% of the time. This is still a limitation of the motion planning algorithm when low latency is an important requirement of the system.

4.2.2.2 Run time

The time needed to reach the grasp pose can vary depending on the manipulator controller used. To avoid such a difficulty, we chose to measure the distance of the path obtained by the algorithm which should be closely linked to the actual run time. To measure it we used the configuration distance of the generated path, calculated as the sum of deltas of each following steps.

CHAPTER 4. EXPERIMENTAL RESULT

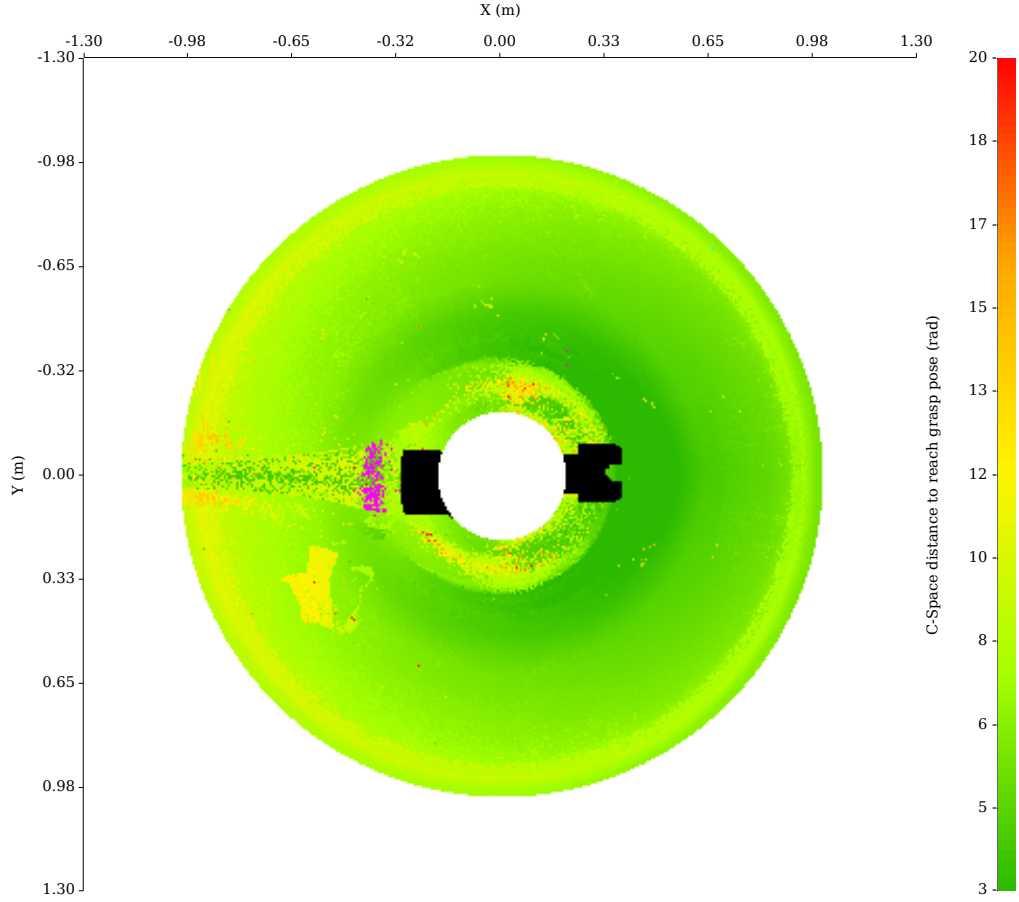


Figure 4.15: Distance (in radians) of the C-Space path for *XARM6* over the X and Y axis at $Z=0.3\text{m}$

In both starting poses, the distance obtained is relatively uniform, increasing when the target further away from the starting pose. Since most targets don't require additional steps this is expected and confirms that the path obtained is sufficiently optimized and should be completed in a competitive time compared to other algorithms.

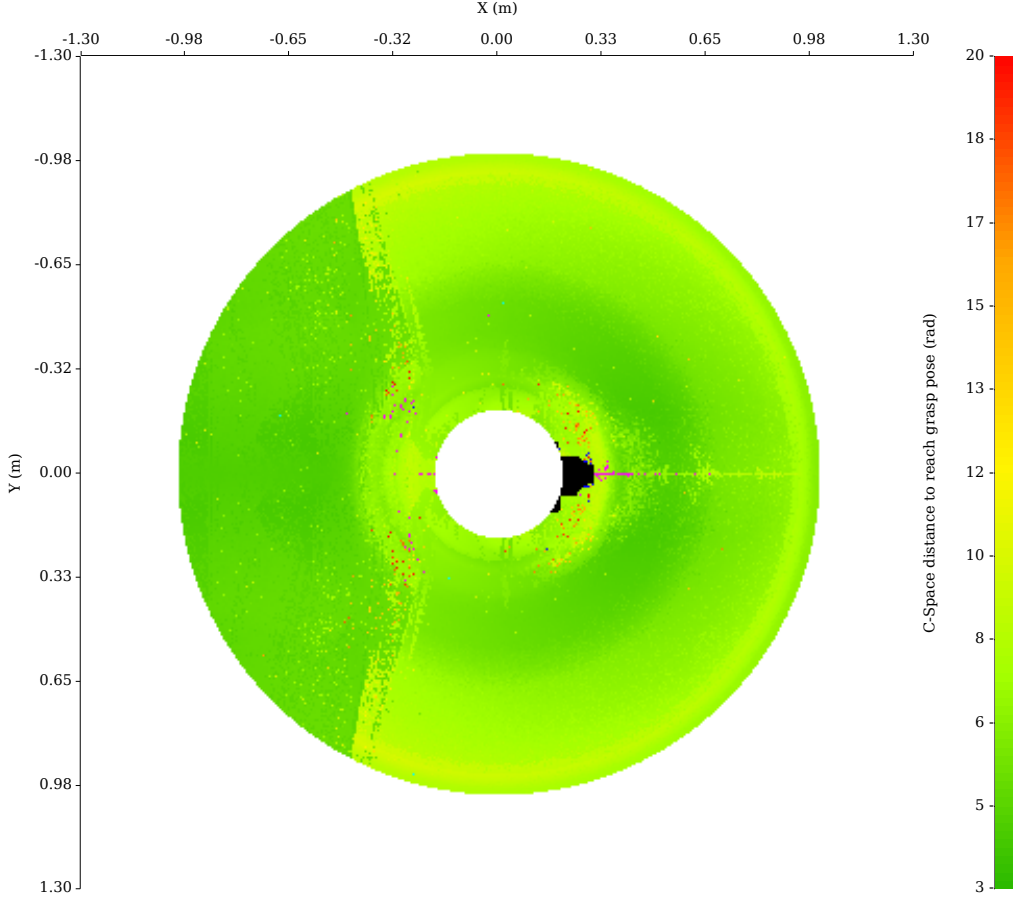


Figure 4.16: Distance (in radians) of the C-Space path for *XARM6* over the X and Y axis at $Z=0.3\text{m}$ with the zero starting position

4.3 Limitations

4.3.1 IK limitations

In the context of Inverse Kinematics (IK), using objectives means that each task must be formulated as a mathematical expression, which is not always straightforward. Moreover, fine-tuning the parameters of these objectives is often required, and such tuning does not always generalize well across different manipulators. For instance, using a swamp loss for matching the z-position improves performance on the *XARM6*, but can cause convergence issues when applied to the *UR5*.

While adopting more generic objectives, which can be slightly adjusted based on the specific task (e.g., the length of the bottle in our example), may alleviate some

of these challenges, the system will still be limited to a set of predefined tasks. This can restrict the flexibility of the IK approach.

Additionally, for complex grasps that require significant movement from the robot, the farther the solution is from the initial configuration, the more challenging it becomes for the IK to find a feasible solution. As a result, the system performs best when targeting positions that are close to the robot’s starting configuration, with the iterative-based model converging quickly in such cases. This implies that for some initial poses, certain areas of the configuration space may not produce feasible grasp poses. To address these limitations, a hybrid approach combining a traditional IK solver with a motion planner could be considered for edge cases, improving the robustness of the system.

4.3.2 Motion planning limitations

For motion planning, the long runtime required for certain configurations is one of the most limiting factors. As discussed in § 4.2.2.2, while such cases are rare, they can significantly increase the overall latency of the system. This issue is primarily due to the complexity of the path finding problem in these scenarios, where the object (e.g., the bottle) is often too close to the arm’s initial position, necessitating many steps to avoid collisions. Although this challenge seems inevitable, it should still be highlighted as a limitation.

Additionally, the generated motions could benefit from smoothing to improve trajectory quality. However, since smoothing typically incurs a higher computational cost, it may not be feasible within the scope of this project.

4.3.3 Experimental limitations

The example of the bottle pre-grasp used to test our system is based on several simplifications:

- We consider only an upright bottle aligned with the robot’s reference frame. While this configuration is common in many scenarios, real-world situations may involve bottles in different orientations.
- We assume the bottle’s position is known exactly. In a dynamic environment,

CHAPTER 4. EXPERIMENTAL RESULT

however, the bottle's position would typically be provided by computer vision systems, which often come with some degree of error.

- The environment used for testing is relatively simple, with only a ground and the bottle as obstacles. In more complex real-world settings, the presence of additional obstacles could significantly increase the difficulty of the task.

Chapter 5

Conclusion and Future Work

In this thesis, we addressed the growing challenges associated with the increasing complexity of robotic arm movements, particularly in the context of real-time applications. Our purpose was to explore the currently available methods for robotic arm manipulation. By using *RelaxedIK* and *RRT* we were able to demonstrate that this method makes it possible to solve common goal in a precise and above all computation time-efficient way.

Through this endeavour, we made a direct contribution to enriching the *RelaxedIK* library with :

- code improvements
- various optimization
- a motion planning algorithm (*RRT*) integration
- the creation of mostly automatized benchmarks
- the creation of visualization framework

We showcased a few of Rust’s capabilities, in reliability and performance. It is a true alternative to C++ for low level robotics and will undoubtedly become increasingly popular in this field. Like any other programming language, it requires some getting used to, especially when coming from higher level languages like python but the compiler being so helpful with nicely formatted and easy to understand errors makes it a lot nicer to learn than other languages.

We hope the development tools provided in this study such as the visualization or benchmark framework can facilitate further use of this language.

CHAPTER 5. CONCLUSION AND FUTURE WORK

Our empirical test demonstrated that the proposed system is able to produce satisfying pre-grasp poses and motion planning while maintaining minimal computational times. This finding validates our approach and highlights the potential for real-time embedded application where quick decision making is critical. However two main aspect should be explored further in future research:

(1) Implementing our system in an embedded application.

A real-time embedded application is the desired use case for this system, as it demands low-level control and efficient code execution. It would be a first step towards developing a more realistic prototype, allowing for further optimizations that have yet to be implemented in the current version and could improve the system’s performance even further. Such further work would enable additional testing, in particular by enabling the measurement of power consumption and run-time metrics that provide valuable insights into the system’s efficiency.

(2) Diversifying the tasks.

Creating different and task specific objectives to increase the versatility of the system would be required to increase its range of usages. Similarly a constrained motion planning algorithm could help improve the adaptability of our system to difficult tasks as many motions might require additional constraints rather than only avoiding collisions. This could also be done directly with *RelaxedIK* with many intermediate pose generated.

Overall this would probably require making the objectives selection more dynamic to accommodate different scenarios.

We believe this work has demonstrated the potential of our system and established a milestone for future advancements in low-level robotics using Rust.

Bibliography

- [1] A. Amice, H. Dai, P. Werner, A. Zhang, and R. Tedrake, “Finding and optimizing certified, collision-free regions in configuration space for robot manipulators”, in *International Workshop on the Algorithmic Foundations of Robotics*, Springer, 2022, pp. 328–348.
- [2] A. Aristidou and J. Lasenby, “Fabrik: A fast, iterative solver for the inverse kinematics problem”, *Graphical Models*, vol. 73, no. 5, pp. 243–260, 2011, ISSN: 1524-0703. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1524070311000178>.
- [3] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, “Inverse kinematics techniques in computer graphics: A survey”, in *Computer graphics forum*, Wiley Online Library, vol. 37, 2018, pp. 35–58.
- [4] R. Bellman, R. Corporation, and K. M. R. Collection, “Dynamic Programming”, (Rand Corporation research study). Princeton University Press, 1957, ISBN: 9780691079516. [Online]. Available: <https://books.google.com.sg/books?id=wdtoPwAACAAJ>.
- [5] E. W. Dijkstra, “A note on two problems in connexion with graphs”, in *Edsger Wybe Dijkstra: his life, work, and legacy*, 2022, pp. 287–290.
- [6] Dimforge, “Nalgebra: Linear algebra library for the rust programming language”, [Online; accessed 28-october-2024]. [Online]. Available: <https://docs.rs/nalgebra/latest/nalgebra/>.
- [7] R. Fletcher, “Practical methods of optimization”, John Wiley & Sons, 2000.
- [8] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths”, *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

BIBLIOGRAPHY

- [9] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”, *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [10] S. M. LaValle and J. J. Kuffner Jr, “Randomized kinodynamic planning”, *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.
- [11] S. Liu and P. Liu, “A review of motion planning algorithms for robotic arm systems”, in *RiTA 2020: Proceedings of the 8th International Conference on Robot Intelligence Technology and Applications*, Springer, 2021, pp. 56–66.
- [12] T. Lozano-Perez, “Spatial planning: A configuration space approach”, Springer, 1990.
- [13] N. D. Matsakis and F. S. Klock, “The rust language”, *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 103–104, 2014.
- [14] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, “Task-priority based redundancy control of robot manipulators”, *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, 1987.
- [15] OpenRR, “An open rust robotics platform: Openrr”, [Online; accessed 28-october-2024]. [Online]. Available: <https://github.com/openrr/openrr>.
- [16] OpenRR, “K: Kinematics library for rust-lang”, [Online; accessed 28-october-2024]. [Online]. Available: <https://docs.rs/k/latest/k/>.
- [17] OpenRR, “Urdf-rs: Urdf parser for rust”, [Online; accessed 28-october-2024]. [Online]. Available: <https://docs.rs/urdf-rs/latest/urdf-rs/>.
- [18] OpenRR, “Urdf-viz: Visualize urdf file”, [Online; accessed 28-october-2024]. [Online]. Available: https://docs.rs/urdf-viz/latest/urdf_viz/.
- [19] D. L. Pieper, “The kinematics of manipulators under computer control”, Stanford University, 1969.
- [20] T. Preston-Werner, “Toml: Tom’s obvious minimal language”, [Online; accessed 25-october-2024], 2021. [Online]. Available: <https://toml.io/en/>.
- [21] T. Preston-Werner and P. Gedam, “Toml v1.0.0 specification”, [Online; accessed 25-october-2024], 2021. [Online]. Available: <https://toml.io/en/v1.0.0>.

BIBLIOGRAPHY

- [22] D. Rakita, B. Mutlu, and M. Gleicher, “RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion”, in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, 2018.
- [23] D. Rakita, H. Shi, B. Mutlu, and M. Gleicher, “Collisionik: A per-instant pose optimization method for generating robot motions with environment collision avoidance”, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 9995–10 001.
- [24] F. Reuleaux, “The kinematics of machinery,(trans. and annotated by abw kennedy)”, 1963.
- [25] Robotiq, “Robotiq 2f-85 product page”, 2024. [Online]. Available: <https://robotiq.com/products/adaptive-grippers#Two-Finger-Gripper>.
- [26] P. Sopasakis, E. Fresk, and P. Patrinos, “OpEn: Code generation for embedded nonconvex optimization”, in *IFAC World Congress*, Berlin, 2020.
- [27] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos, “A simple and efficient algorithm for nonlinear model predictive control”, in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, IEEE, 2017, pp. 1939–1944.
- [28] The Rust team, “The Rust programming language”, [Online; accessed 27-october-2024], 2017. [Online]. Available: <https://doc.rust-lang.org>.
- [29] The Rust team, “Procedural macros - the rust reference”, [Online; accessed 20-october-2024]. [Online]. Available: <https://doc.rust-lang.org/beta/reference/procedural-macros.html>.
- [30] The Rust team, “The manifest format - the cargo book”, [Online; accessed 20-october-2024]. [Online]. Available: <https://doc.rust-lang.org/cargo/reference/manifest.html>.
- [31] UFactory, “Xarm6 product page”, 2024. [Online]. Available: <https://www.ufactory.cc/xarm-collaborative-robot/>.
- [32] Universal Robots, “Ur5 product page”, 2024. [Online]. Available: <https://www.universal-robots.com/products/ur5-robot/>.

BIBLIOGRAPHY

- [33] N. Wagaa, H. Kallel, and N. Mellouli, “Analytical and deep learning approaches for solving the inverse kinematic problem of a high degrees of freedom robotic arm”, *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106 301, 2023.
- [34] Y. Wang, P. Praveena, D. Rakita, and M. Gleicher, “Rangedik: An optimization-based robot motion generation method for ranged-goal tasks”,, pp. 9700–9706, 2023.
- [35] S. J. Wright, “Coordinate descent algorithms”, *Mathematical programming*, vol. 151, no. 1, pp. 3–34, 2015.
- [36] C. Zhou, B. Huang, and P. Fränti, “A review of motion planning algorithms for intelligent robots”, *Journal of Intelligent Manufacturing*, vol. 33, no. 2, pp. 387–424, 2022.