DEGREE PROJECT IN MECHANICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Superquadrics-augmented Rapidly-exploring Random Trees

YARED EFREM AFEWORK

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Superquadrics Augmented Rapidly-exploring Random Trees

YARED EFREM AFEWORK

M.Sc. Engineering Design, Mechatronics Track
Date: June 12, 2019
Supervisor: Pouya Mahdavipour Vahdati *pouyamv@kth.se*
Examiner: DeJiu Chen *chen@md.kth.se*
Swedish title: N-tegradsytförstärkta Raskt-utforskande Slumpmässiga Träd
School of Industrial Technology and Management

KTH Industriell teknik och management

Examensarbete MMK [TRITA-ITM-EX 2019:394**]**

Raskt-utforskande Slumpmässiga Träd med N:tegradsytor

Yared Efrem Afework

Godkänt	Examinator	Handledare
2019-06-03	DeJiu Chen	Pouya Mahdavipour Vahdati
	Uppdragsgivare	Kontaktperson
	N/A	Rickard Svensson

Sammanfattning

Nyckelord: Superquadrics, RRT, GJK, Kollisionsdetektering

Detta examensarbete undersökte fördelarna och nackdelarna med att använda n:tegradsytor (NY) för att utföra kollisionsdetektion i algoritmen Raskt-utforskande Slumpmässiga Träd (RST). RST används typiskt för planeringen av system med relativt många frihetsgrader. En etablerad metod för kollisionsdetektion, Gilbert-Johnson-Keerthi-algoritmen (GJK), implementerades även i jämförelsesyfte. Då GJK-algoritmens största flaskhals ligger i kollisionsdetektionen är detta ett intressant ämne att efterforska.

Den NY-baserade kollinsdetektionsmetoden jämfördes med den GJK-baserade metoden både kvantitativt och kvalitativt. Kvalitativt jämfördes beräkningshastighet och minnesåtagande, medan de kvalitativt jämfördes i deras förmåga att representera godtyckliga geometriska former. På ett högre plan diskuterades det även hur lämpliga de är för att modellera en robotarm med 6 stycken frihetsgrader. RST-algoritmen jämfördes även med en annan planeringsalgoritm, A*. Framförallt fokuserade diskussionen kring planering av system med relativt många frihetsgrader.

I det fall inga kollisioner fanns presterade GJK-algoritmen ungefär lika bra som NY algoritmerna i att fastslå detta, utifrån beräkningshastighet. Men när det kom till att upptäcka existerande kollisioner presterade GJK-algoritmen sämre. Minnesmässigt använder GJK-algoritmen mer minne, då den kräver att båda objekten är explicitrepresenterade (dvs, som ett punktmoln), medan man med en NY-metod endast behöver representera ena objektet explicit och den andra implicit.

Gällande förmågan att representera godtyckliga geometriska former är NY-baserade metoder bättre. Till skillnad från GJK som är begränsad till konvexa mängder kan NY uppta icke-konvexa former, exempelvis flottyrmunkformade supertoroider. En metod som använder GJK-algoritmen skulle behöva bygga upp icke-konvexa former med flera mindre konvexa komponenter. NY-metoden är således bättre för att modellera robotarmar med 6 frihetsgrader. Det är dock i praktiken lättare att implementera GJK-metoden då den endast kräver punktmoln, medan NY kräver parametrar som måste bestämmas eller finjusteras.

RST-algoritmen implementerades sist, utformad så att kollisionsdetektionsmetoderna är utbytbara. Det var dock inte möjligt att dra slutsatser utifrån det testdata som erhölls, ty RST-algoritmens slumpmässiga karaktär. RST-algoritmen kan ses som en multiplikator som endast förstorar de inneboende egenskaperna hos kollisionsdetektionsmetoderna.

KTH Industrial Engineering and Management

M.Sc. Thesis MMK [TRITA-ITM-EX 2019:394**]**

Superquadrics-augmented Rapidly-exploring Random Trees

Yared Efrem Afework

Approved	Examiner	Supervisor
2019-06-03	DeJiu Chen	Pouya Mahdavipour Vahdati
	Commissioner	Contact person
	N/A	Rickard Svensson

Abstract

Keywords: Superquadrics, RRT, GJK, Collision Detection

This thesis work investigated the advantages and disadvantages of using superquadrics (SQ) to do the collision-checking part of the Rapidly-exploring Random Trees (RRT) motion planning algorithm for higher Degree of Freedom (DoF) motion planning, comparing it with an established proximity querying method known as the Gilbert-Johnson-Keerthi (GJK) algorithm. In the RRT algorithm, collision detection is the main bottleneck, making this topic interesting to research.

The SQ-based collision detection method was compared to the GJK algorithm both qualitatively and quantitatively, comparing computational speed, memory requirements, as well as the ability to handle arbitrary shapes. Furthermore, how appropriate they are in modelling a 6 DoF arm was analyzed. A qualitative comparison between the RRT algorithm and the A* algorithm was also provided, comparing their suitability for searching in higher dimensional spaces.

When there were no collisions the SQ-based algorithms performed roughly at parity with the GJK algorithm in terms of computational speed. However, when a collision had occurred, the SQ-based algorithms were able to return a positive faster than the GJK algorithm, outperforming it. From a memory standpoint the SQ-based algorithms required less memory as they could leverage the explicit and implicit representations of the SQ objects, whereas the GJK algorithm requires both objects being checked for collision to be explicitly represented as convex sets of points.

Regarding handling arbitrary shapes, the SQ-based algorithms have an advantage in that they can allow for certain non-convex shapes to be. Conversely, the GJK algorithm is limited to convex shapes. The GJK algorithm would thus require more geometric primitives to accurately capture the same non-convex shape. Thus, it can be concluded that the SQ-based method is more suitable for modelling a 6 DoF arm. However, a GJK-based collision detection module would in most cases be a lot more straightforward than the alternative to set up, as it is very simple to collect a set of points.

Finally, both collision detection method types were implemented with the RRT algorithm. Due to the inherently random nature of the RRT algorithm the results of this set of tests could not be used to make any further conclusions beyond showing that it is possible to combine the SQ-based algorithm with the RRT algorithm. Instead, one should see the RRT algorithm as a multiplicative factor applied to the inherent properties of the previously examined collision detection methods.

Acknowledgments

To my family, my mother in particular, thank you for everything you have done for me. I dedicate this thesis to you.

To my supervisor Pouya Mahdavipour Vahdati, thank you for your guidance. I wish you good fortune in all your future endeavors.

To everyone at Tritech, thank you for being so welcoming to me and the other thesis students. Rickard Svensson, thank you for being my company supervisor and doing a wonderful job of it.

With my deepest appreciation,

Yared Efrem Afework

Stockholm, Sweden, May 2019

Contents

List of Figures

List of Tables

Acronyms

AABB Axis Aligned Boundings Box

APF Artificial Potential Fields

CAD Computer Aided Design

DoF Degree(s) of Freedom

EE End-Effector

GJK Gilbert-Johnson-Keerthi algorithm

LQ Linear-Quadratic

MAV Micro-Air Vehicle

MIMO Multiple Input, Multiple Output

MPC Model Predictive Control

NMPC Nonlinear Model Predictive Control

OBB Oriented Bounding Box

PRM Probabilistic Roadmap

QP Quadratic Programming

RRT Rapidly-exploring Random Trees

SISO Single Input, Single Output

SQ Superquadrics

Chapter 1

Introduction

The introduction presents the project, detailing the background motivating its conception as well as the manner and scope of its execution.

1.1 Background

Entire industries have emerged on the basis of robotic manipulators autonomously and physically interacting with objects. From production lines employing factory robots working non-stop to machine, weld, paint and assemble parts[Appleton and Williams]; to automated milking systems used by dairy farmers[Rossing et al.] allowing for cows to milk themselves; the uses for responsive and flexible robotic manipulators are plentiful. Furthermore, the integration of machine vision into these systems increases their utility [Snyder and Qi].

Planning the motion of these robotic manipulators is a challenge. The addition of more Degree(s) of Freedom (DoF), even to the point of redundancy, is used to expand the solution space available to solve the manipulation problem. This expansion, however, comes at the cost of increasing the difficulty on the motion planner to produce collision free motion paths.

Sampling-based Path Planning

One commonly used family of algorithms in the field of robotic motion planning is the sampling-based family of algorithms. This group of

algorithms can roughly be divided into two categories: single-query and multi-query algorithms. Multi-query algorithms assume that the environment which the manipulator will navigate itself through is static enough that it is possible to prepare a map of possible paths in advance. The former, however, makes no such assumption of staticity on the environment, resulting in a need to produce a collision-free path online.

Within the category of single-query sampling-based algorithms, an algorithm that has seen significant use is the Rapidly-exploring Random Trees (RRT) algorithm and its variants [Siciliano and Khatib]. RRTs are a popular and general approach for solving single-query high-dimensional motion planning in robotics [Bialkowski et al.]. RRTs work by sampling various angle configurations in configuration space and connecting collision-free points in a tree-like fashion until a path has been found from, start pose to goal pose [LaValle].

The main bottleneck of RRTs and sample-based path planning algorithms in general, however, is the collision detection mechanism[LaValle].

Superquadrics

In mathematics there exists a family of 3D geometric shapes called Superquadrics (SQ) [Jaklič et al.], a generalization on the Quadrics family of 2D surfaces in 3D space. These have been applied in the computer graphic modelling field to describe 3D shapes since the 1980s [Yerry and Shephard]. One of their big advantage is that they can describe a wide variety of solid objects under a mathematical closed form using only eleven parameters [Katsoulas].

An SQ has the implicit equation $F(P, \Lambda)$, where P is a point (x_P, y_P, z_P) and Λ the 11 parameter defining the SQ. The implicit function can be used as a very efficient test of whether P is:

- inside the SQ: $F(P, \Lambda) < 1$;
- outside the SQ: $F(P, \Lambda) > 1$;
- on the surface of the SQ: $F(P, \Lambda) = 1$.

[Fantacci et al.] used SQ to successfully make a humanoid robot grip an unknown, unmarked object with its End-Effector (EE). They pre-modelled the robot hand's grasping volume with SQ using a Computer

Aided Design (CAD) model. For the target object however, they used a 3D sensor to collect a 3D point cloud of its surface and were able to estimate the 11 parameters of a SQ using non-linear optimization.

A number of other researchers have also shown the potential of SQ to be used for shape and pose recovery of unknown objects in robotics using range data ([Duncan et al.],[Silva et al.], [Biegelbauer and Vincze]).

Regarding Collision Detection in General for Robotic Manipulator Planning

Collision detection is often treated as a "black box" in motion planning [LaValle], and in the experience of this thesis author rarely do researchers go into the specifics of how their algorithms handle it. It is often mentioned as a line in their pseudo-code section: *"if collisiondetect(q) then:"*, such as in [Lee et al.] and [Klanke et al.]. The authors in [Hirano et al.] devoted one sentence to it, stating that they made use of the Proximity Query Package to do proximity checks between objects, relying on Oriented Bounding Box (OBB)Trees [Gottschalk et al.] for hierarchical representation of bounding volumes that can be queried for intersections.

Many researchers use model-based development and first deploy their algorithms in robotic simulators and CAD software, which can handle their collision detection for them. In [Lahouar et al.] the authors did their modelling in a Robotics CAD software known as SMAR, which provided them with models of PUMA robotics. The creators of SMAR state in their paper [Zeghloul et al.] that they rely on two parts to do collision checking. The first part consists of checking the intersections of the facets of the obstacles, reminiscent of the famous Gilbert-Johnson-Keerthi algorithm (GJK) distance algorithm ([Gilbert et al.], [Lindemann]). In order to speed up calculation, the CAD software pre-models all objects using bounding primitives like spheres and parallelepipeds. These are much easier to do initial checks with, and combining them using Boolean algebra allows for non-convex sets to be modeled, before resorting to the intersection of facets check.

The researcher in [Park] did something similar, choosing to represent his robot composed of two 6-DoF arms and a torso using spheres instead of in accordance with the exact virtual CAD model. This motivates the use of SQ for collision detection in robotic arm simulation, as SQ

are a mathematical superset that encompass both spheres and some parallelepipeds.

1.2 Purpose

The contribution of this thesis is to investigate the utility of augmenting the Rapidly-exploring Random Trees sampling-based algorithm by relying on SQ to handle collision detection. The RRT is augmented with a method to easily check the occurrence of a collision by evaluating the implicit function $F(.,.)$ in each iteration. If in an iteration, any point P belonging to the EE SQ or, any of the individual manipulator links' SQ, results in $F(P,.) < 1$, then a collision has occurred and the proposed path is discarded.

While SQ, as previously mentioned, have been shown to be used for shape and pose estimation of unknown objects, as far as this thesis author has been able to find, SQ have never been specifically used in conjunction with RRT to handle collision detection.

SQs are useful in that they can be used to pre-model previously known objects (including the individual links of a robotic arm and EE) while also be used to deal with unknown objects, expressing everything in a mathematically advantageous way that allows for potentially convenient collision checking in RRT. The novelty, as well as the wide range of potential use cases in markerless visual servoing, makes it a topic worth researching.

The thesis will attempt to answer the following questions:

RQ1: Comparing the outlined novel method to an alternative motion planning method such as A*, in combination with an alternative obstacle collision-detection method such as GJK Distance algorithm, which is the better combination in regards to:

- **a**: Computational processing time and memory management?
- **b**: Ability to handle arbitrary obstacle shapes?

RQ2: When using the SQ-based method for modeling a 6 DoF arm and performing collision detection, how well can it be fit to the arm shape compared to when GJK is used?

1.3 Scope & Delimitations

While some of the ultimate applications of this method is thought to be found in visual servoing, this project is limited to investigating what benefits and disadvantages an algorithm making use of SQ to perform collision detection in RRT has compared to existing methods. As such, no actual visual servoing will be done.

Furthermore, the following delimitations are set:

- **RQ1a** will be answered by setting up and evaluating simulations for the RRT and collision detection methods. A* will not be implemented, but instead examined qualitatively.
- **RQ1b** will be evaluated qualitatively.
- **RQ2** will be evaluated qualitatively. By fit, the analysis will be considered from the point of risking False Negatives and False Positives in the collision detection due to improper representation, as well as the ability to model arbitrary shapes.
- The analysis of A* will be limited to the vanilla A*.
- Though the proposed method could work with multi-query sampling-based algorithms, only single-query sampling-based algorithms will be looked at - RRT specifically.
- Of all the RRT algorithms, only vanilla RRT will be implemented.
- The six DoF manipulator examined will be a fixed, serial chain arm.

1.4 Methodology

In order to answer the stated research questions an SQ collision detector must be implemented as well as an RRT planner that can utilize it, along with a GJK collision detector. The two collision detector algorithms will be treated as "black boxes" that can be used interchangeably with any planning algorithm, provided that the virtual obstacles and virtual arm have been successfully loaded.

The specifics of the implementation and certain algorithmic design choices will be detailed in Chapter 3, and will have been developed

in accordance with the theoretical background provided in Chapter 2. Once all algorithm combinations have been tested and the results gathered they will be presented in Chapter 4 and further discussed in Chapter 5. See Figure 1.1 for a graphical representation of the workflow.

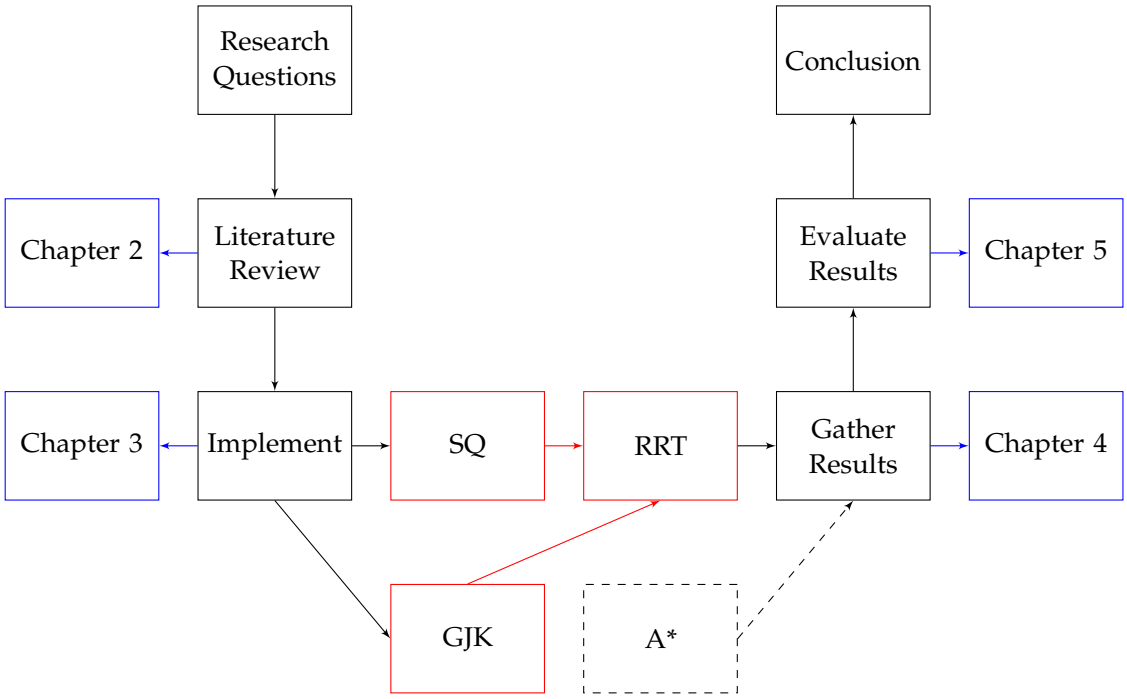


Figure 1.1: Chart illustrating the workflow and temporal dependencies. Blue boxes represent work resulting in report substance. Red boxes represent work resulting in executable code. Dashed represents a qualitative analysis.

1.5 Ethics

By itself, improving a path planning algorithm for markerless visual servoing does not raise ethical issues. But it is part of a general trend of autonomous robotics, which is having and will have a great effect on society. It ties into autonomous:

- driving, which will have significant effects on the employment of millions of lorry and taxi drivers all over the world, with up to 4 million jobs likely to be lost in the near future[Austin et al.];
- operation of military drones, which is raising serious ethical concerns [Wilson];

– manufacturing, which is set to eliminate 800 million jobs by 2030 [Manyika et al.].

In addition to the impact on economic activity and employment, and the concerns associated with the creation of highly autonomous killing machines, there are also the concerns related to software and hardware reliability. If an autonomous vehicle is involved in an accident with fatal consequences as a result of a bug, who is responsible? These concerns need to be considered by anyone looking to use the results of this thesis.

1.6 Disposition

The disposition of this thesis report is as follows:

Chapter 1 Introduction introduces this master thesis, the methodology and scope.

Chapter 2 Theoretical Framework contains a compilation of the theoretical background in the respective fields, which will inform the implementation.

Chapter 3 Implementation details the implementation as based off of the chosen methodology and the theoretical background.

Chapter 4 Results lists the results of a series of test rounds comparing the SQ and GJK algorithms, independently and as part of the RRT algorithm.

In Chapter 5 Discussion the results are discussed in the context of the research questions, as well as the implementation in general. A concluding statement is prepared and suggestions for future work and research are given.

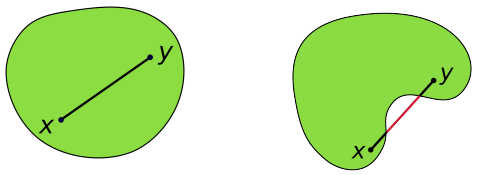
Chapter 2

Theoretical Framework

In order to properly answer the established research questions an extensive review of the literature is needed. This chapter provides the fruits of that review. The knowledge provided here will guide the setup of the simulations, serve to inform the qualitative analyses of the discussion as well as form a reference for the suggestions for future research and work.

2.1 Convexity

The concept of convexity features heavily in the algorithms mentioned in this thesis. Thus, it is relevant to define its definition for the reader's convenience.



(a) A convex set. No matter what points x and y are chosen, the line between them will remain within the set.

(b) A non-convex set. The red section signifies the part of the line not part of the set.

Figure 2.1: Source: Images from Wikicommons released into the public domain.

A set is convex iff any line drawn between any arbitrary chosen pair of its points is fully contained within the set itself (see Figure 2.1a). If this criteria is not met, the set is non-convex (see Figure 2.1b). Similarly, a

function is convex iff the epigraph (the set of points lying on or above its graph) is a convex set [Sasane and Svanberg] .

This can be expressed mathematically using the following inequality,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y), t \in (0,1). \tag{2.1}$$

Convex functions possess some properties that make them desirable in, among other fields, optimization. A problem consisting of finding the optimum of a function can be considered "well-posed" if the function as well as the feasible set (i.e., the set of values the variables are allowed to take) is convex [Sasane and Svanberg].

As an example, x^2 is a convex function possessing one global minimum at $x = 0$. $x^3 + x^2$, on the contrary, is a non-convex function possessing a global minimum at $x = -\infty$, a global maximum at $x = \infty$, a local minimum at $x = 0$ and a local maximum at $-\frac{2}{3}$. The existence of these local optima will cause problems for certain algorithms that rely on differentiating the function.

2.2 Regarding Robotic Arms

Robotic mechanisms are systems of rigid bodies connected by joints, whose positions and orientation are collectively termed the pose. One important topology is the one where the joints are connected in a serial chain, where each joint is connected to two others, except for the first and final link which are only connected to one other member [Siciliano and Khatib].

A multitude of methods and formalisms have been developed to help in abstracting and representing the manipulator, as well as the environment in which it can actuate. One such formalism is that of configuration spaces.

2.2.1 Configuration Space

A useful way to abstract path planning problems is to transform them from the real space of reality into configuration space, or C-space [Siciliano and Khatib]. C-space is the space containing all possible configurations of a manipulator, where each point single point in the C-space corresponds to a specific position and orientation [Lozano-Perez].

The axes spanning the C-space each correspond to a DoF, e.g. the angle of a revolute joint or the translation of a prismatic joint. Thus, while the number of dimensions of the real space our robots work in is limited to 3, the amount of dimensions of C-space is only limited by the number of DoF.

The big benefit of using C-space is that configurations resulting in collisions in real space due to the presence of objects will designate whole regions in it as forbidden. This will have consequences for some of the algorithms proposed here.

2.3 Algorithms That Plan

There is a fundamental need in robotics to have algorithms that work to transform high-level specifications of tasks set by humans (e.g. "grab that object"), into low-level instructions of how to move [LaValle]. The terms motion planning, trajectory planning, and so forth are often used to refer to these kinds of problems.

Motion planning has, since the late 1970s, been an active area of research. Initially the different approaches to the problem focused on complete planners, i.e. planners that return a solution provided that one exist, and return failure if not [Karaman and Frazzoli]. It was however established that even the most simplistic motion planning problem, known as the piano mover's problem, is of complexity type PSPACE-hard (**P**olynomial **SPACE**). Thus, there is strong evidence that *complete* planners are doomed to be computationally intractable, as the algorithm requires time growing exponentially with the number of DoF [Reif].

Algorithms which intend to be tractable thus approach the problem by relaxing the completeness requirement. Examples include [Karaman and Frazzoli]:

1. Resolution Completeness: A solution can be found if one exist provided that the resolution parameter of the algorithm is fine enough. Motion planning methods that are based on gridding or cell decomposition tend to fall into this category.

2. Probabilistic Completeness: The probability of finding a solution, provided that one exists, approaches 1 as the number of samples approaches infinity. Sampling-based algorithms fall under this.

2.3.1 A*

The A* (pronounced A *star*) algorithm was first proposed in its seminal 1968 paper [Hart et al.]. It was developed as a solution to the problem of optimal graph traversal, i.e. that of finding the minimum cost path in a graph. Real life manifestations of that problem included optimal routing of telephone traffic, optimal maze navigation, and layout planning of printed circuit boards. The authors recognized that those problems were often approached in one of two ways: mathematically and heuristically.

The mathematical approach typically dealt with properties of graphs as abstract mathematical objects, whose nodes would be orderly examined using algorithms that were mathematically proven to find the optimal solution. This approach, however, was not necessarily concerned with the *computational feasibility* of the algorithms as much as with the guarantee of the optimal solution being found.

The heuristic approach makes use of domain-specific knowledge of the problem being represented with a graph in order to facilitate the search. For example, an AI strategy board game opponent can make use of expert knowledge to better evaluate the utility of specific states. A public transportation trip planner can make use of Euclidean distance and geographic information to avoid going through branches of the graph that will have the user be transported away from their goal. In contrast with the previously mentioned approach, this approach is typically not concerned with finding a *minimum cost* solution as it is with finding *a* solution, as fast as possible.

A* came about as an attempt by the authors of [Hart et al.] to combine these two approaches.

Regarding Graphs and Traversal

A graph G is composed of a set of $\{n_i\}$ nodes and $\{e_{ij}\}$ arcs between those nodes. If $e_{pq} \in \{e_{ij}\}$ then there is an arc between n_p to n_q, and n_q is in such a case a successor of n_p. Traversing that arc incurs a cost

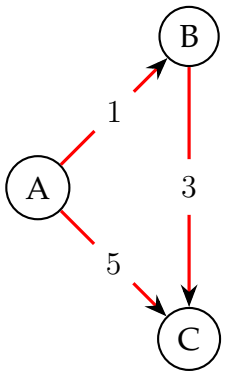


Figure 2.2: Three nodes n_A, n_B and n_C in a graph and the costs of traversing their arcs.

c_{pq}, which is a member of the overall set c_{ij}. Note that in many cases $c_{ij} \neq c_{ji}$ is possible, assuming that both arcs exist.

Pairs of nodes do in general have multiple subsets of arcs connecting them. These are called paths. Figure 2.2 illustrates a case where there are two paths between nodes n_A and n_C, one of which is composed of one arc (e_{AC}) and the second of two arcs (e_{AB}, e_{BC}). Note also that the path composed of two arcs is of a lower total cost than the path composed of one.

An admissible algorithm could work as follows:

1. Start from a start node n_s and expand the node.
2. List all the K nodes $n_1, ..., n_K$ which are connected to n_s by the edges $e_{s1}, ..., e_{sK}$, each of which are associated with a cost c_{sk} for $k \in 1, ..., K$.
3. Sort the nodes along their respective cost.
4. Pick the node whose path has the lowest cost and check if it is the goal node.
   (a) If true, terminate and return path.
   (b) If false, check if a cheaper path to the node exists in the memory bank.
      i. If true, move on to the second cheapest node.
      ii. If false, save to the memory bank as the cheapest path and expand the node.

5. Sort all of the nodes according to costs of their paths. Duplicate nodes, representing new paths to the same nodes, can emerge.
6. Repeat from Step 4.

A memory bank from which to filter out already traversed paths is needed. Otherwise, using the graph in Figure 2.3 as an example, the algorithm could get stuck in a loop: $n_A \rightarrow n_B \rightarrow n_A \rightarrow n_B \rightarrow ....$ This algorithm would eventually return an optimal solution, but it would not necessarily do it fast enough, depending on the application and graph size.

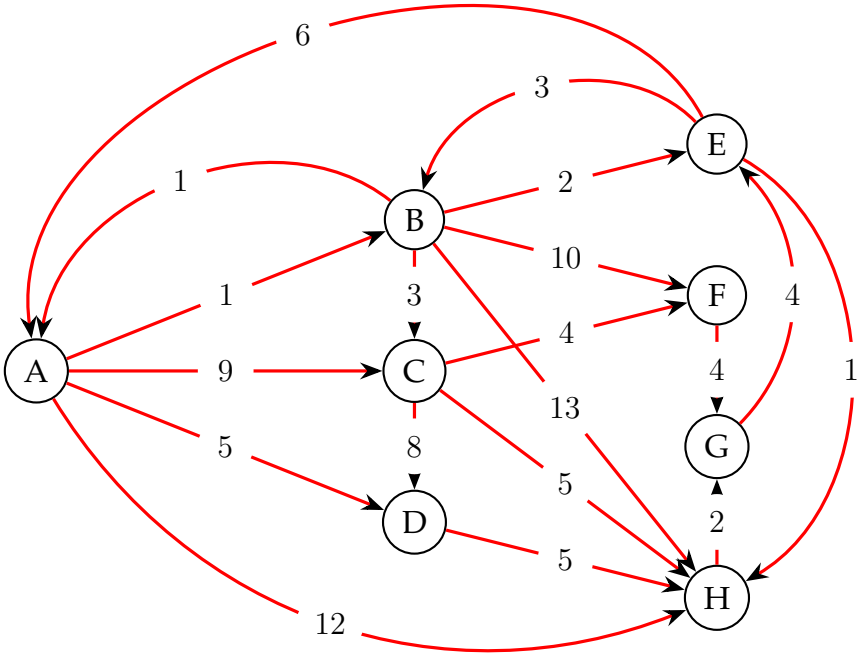


Figure 2.3: A graph with arc costs.

Note the use of the verb expand in the algorithm. A different way of viewing a graph is to begin by only viewing the start node and disregarding the rest of the nodes. Then, the first level of nodes only emerge as a consequence of applying a successor operator Γ to the starting node [Hart et al.]. In Figure 2.3, if n_A is set as the start node it would correspond to nodes n_B, n_C, n_D and n_H.

In essence, the algorithm calculates the cost of the path to the node, which we can call $g(n)$, and attempts to minimize the cost of that all the way from the start node to the goal node.

Heuristics

Instead of being concerned with the cost of traversing an arc to a node, one might instead want to assign a cost to the node in question that is independent of the path taken to reach it. The cost itself would be domain-specific, using a suitable heuristic.

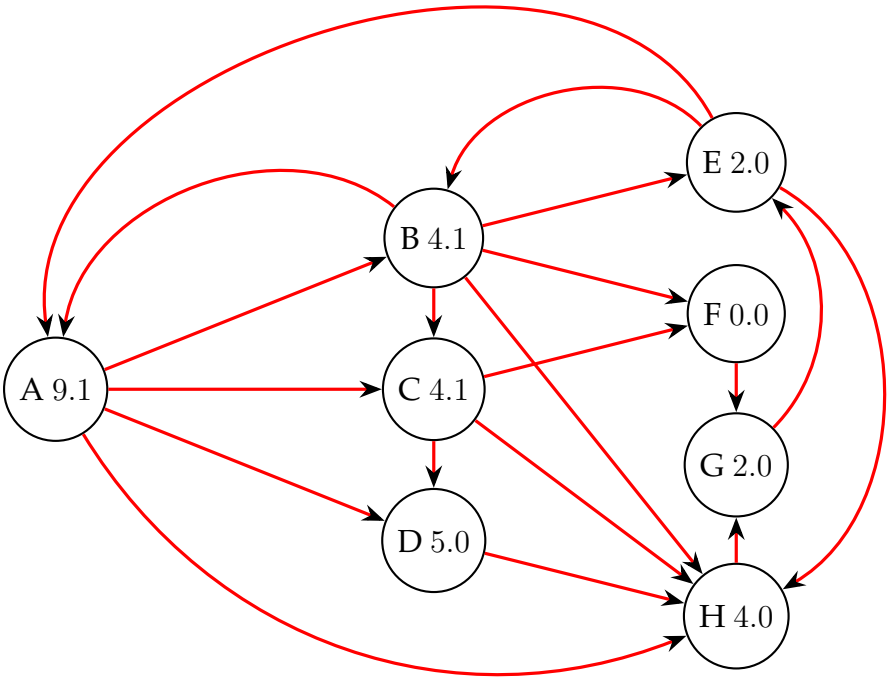


Figure 2.4: A variant on Figure 2.3 with approximate Euclidean distance to goal node n_F as a heuristic cost.

In navigation, the goal in is to minimize the distance between an object and the destination by moving the object to it. Thus, it makes intuitive sense to incorporate the distance as a heuristic cost, by for example calculating the Euclidean distance for each node to the goal node (i.e.

the destination). Figure 2.4 gives an example of such a graph, where n_F is the assigned goal node.

An algorithm making use of the Euclidean distance as the heuristic cost $h(n)$ to minimize in each step, in accordance with Figure 2.4, with n_A as the start node, would view n_H as the cheapest node to expand on, regardless of the cost incurred to traverse to it. It would then expand on node n_G, then node n_E, before finally evaluating n_B and then reach the goal n_F.

As an algorithm working to make locally optimal decisions at every stage in the hope of reaching a globally optimal path, it can be considered a greedy algorithm [Cormen et al.]. In this case however, while it would reach a solution in a reasonable amount of steps, it would not reach the globally optimal solution, which is in actuality $n_A \rightarrow n_B \rightarrow n_C \rightarrow n_F$.

Combining *g(n)* and *h(n)*

A* works by combining the two cost functions, to produce the following

$$f(n) = g(n) + h(n), \tag{2.2}$$

such that the cost evaluated when deciding the cheapest node takes into account both the path and heuristic costs. Thus, the strength of both approaches are combined.

Furthermore, in [Hart et al.], the authors prove that if the heuristics function chosen is *admissible*, the algorithm is guaranteed to return a least-cost path from start to goal. An admissible heuristic is a heuristics function that does not over-estimate the actual cost to get to the goal.

A* in Motion Planning

A* by itself is a path planning algorithm, in that it produces a plan for how to progress through many states before ending at a goal (provided that such a path exists).

It is by combining it with the concept of the C-space mentioned in Section 2.2.1 that motion planning is achieved, as the path produced by the algorithm corresponds to specific joint configurations of the robot.

Feedback control policies can be used to have the joints actuate along the motion plan.

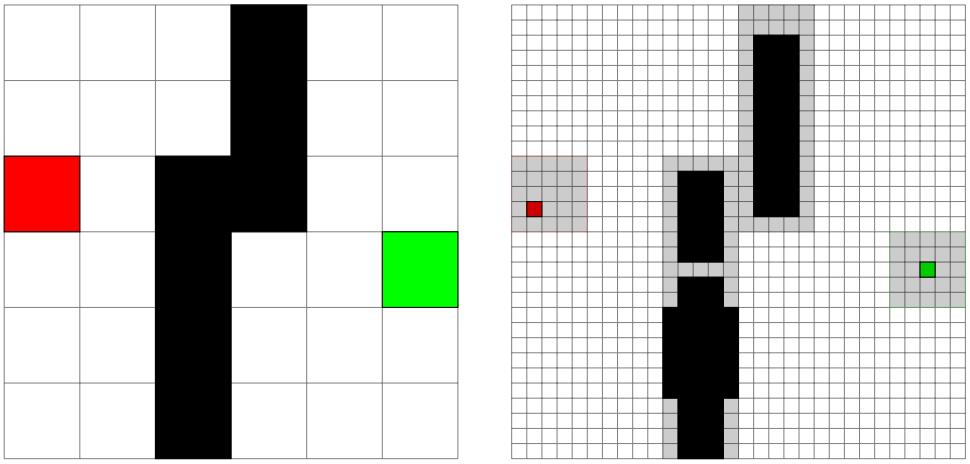


Figure 2.5: The same environment, represented with two grids of different resolutions, 1 cm and 0.2 cm step size respectively. Red represents the start state, green the goal state and black obstacles. The gray outline is used as a visual aid.

A* used in this way falls into the category of algorithms mentioned in Section 2.2.1 as being resolution complete. Figure 2.5 illustrates this concept. Both figures represent a grid representation of an environment in which a path from the red square to the green square must be found. In reality, a path does exist, but the algorithm's ability to find it is dependent on the grid resolution that has been set. Hence, no solution is available in the left case, but several exist in the right case. As the resolution goes to infinity, so does the computational challenge.

In essence, an infinitely fine grid, with infinitely many nodes and arcs between them, would result in a complete but wholly intractable planner, as mentioned in Section 2.3. The exponential nature of the grid means that by increasing the resolution 5x (i.e., reducing the width of the squares by a factor 5), 25x more *square* nodes emerge. In the 3D world, this would be a cubical increase, with 125x more cubic nodes replacing every 1 cube.

Some methods have been developed to mitigate this problem, such as making the grid variable. Occupied grids could be recursively divided into finer grids, while free grids remain coarse. [Kirby et al.] presented an algorithm designed to be used in real-time navigation path planning, where only the space close to the robot (which is of high relevance) is

divided. Meanwhile, areas of the grid farther away are left coarse. Even then, while the computational strain is lessened, the issue of choosing the right resolution remains.

2.3.2 Artificial Potential Fields

A different planning algorithm, which draws inspiration from physics, is the Artificial Potential Fields (APF) algorithm. In the APF algorithm, a potential field is defined for the world. The manipulator moves in this field and experiences forces: repulsive forces are exerted from obstacles, and an attractive force is exerted from the goal point.

The generalized kinetic energy equation for a holonomic articulated mechanism can be defined as follows,

$$T(x, \dot{x}) = \frac{\dot{x}\Lambda(x)\dot{x}}{2} \tag{2.3}$$

where $\Lambda(x)$ designates the symmetric matrix of the quadratic form, i.e. the kinetic energy matrix. Using the Lagrangian formalism, the equations defining the EE motions are given by

$$\frac{d}{dt}\left(\frac{\delta L}{\delta x}\right) - \frac{\delta L}{\delta x} = F, \tag{2.4}$$

where the Lagrangian function is

$$L(x, \dot{x}) = T(x, \dot{x}) - U(x), \tag{2.5}$$

and U(x) is defined as the potential energy of the gravity. F is the operational force vector. [Khatib] further develops and defines these equations in the form

$$\Lambda(x)\ddot{x} + \mu(x, \dot{x}) + p(x) = F, \tag{2.6}$$

with $\mu(x, \dot{x})$ as the centrifugal and Coriolis forces and $p(x)$ the forces of gravity.

For the case of a single obstacle, the following equation can be set up,

$$U_{art} = U_{x_d} + U_O \tag{2.7}$$

where U_O refers to the field generated by the obstacle and U_{x_d} by the goal at position x_d. The $U(x)$ term in Equation 2.5 can thus be redefined as

$$U(x) = U_{art}(x) + U_g(x), \tag{2.8}$$

as in addition to the gravitational field (represented by U_g), there is also the addition of the new artificial potential fields. Finally, we obtain the forces the end effector experiences by taking the negative of the gradient of these field functions:

$$F^*_{art} = F *_{x_d} + F*_O = -\frac{\delta U_{x_d}}{\delta x} - \frac{\delta U_O}{\delta x}. \tag{2.9}$$

The attractive potential field of the goal point is simply

$$U_{x_d}(x) = \frac{k_p(x - x_d)^2}{2}, \tag{2.10}$$

where k_p is a proportional control parameter affecting the strength of the attraction. $U_O(x)$ is a little more complicated to decide on, but it is selected such that the APF $U_{art}(x)$ as a whole is a positive continuous and differentiable function which reaches a zero minimum when $x = x_d$. It should be designed such that it meets the manipulator's stability condition, creates a potential barrier at each point on the obstacle's surface (a barrier that becomes negligible beyond the surface) and is a non-negative continuous and differentiable function. [Khatib] coined the term FIRAS function, stemming from ***F**orce **I**nducing an **A**rtificial **R**epulsion from the **S**urface* (but in French) for the force created by $U_O(x)$.

One example function proposed in [Khatib] is the following function:

$$U_O(x) = \begin{cases} \frac{\eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2}{2} & \textit{if } \rho \leq \rho_0 \\ 0 & \textit{if } \rho > \rho_0 \end{cases} \tag{2.11}$$

where ρ is the closest distance to the obstacle, ρ_0 a distance limit set by the user and η a parameter adjusting the force strength. For distances above the distance limit the function is cut off.

Figure 2.6 shows how a discontinuous potential field might look like. For further in-depth information regarding the control theory, potential artificial potential functions and more, see [Khatib].

It is worth pointing out that SQ have been used to model obstacles for APF. *Superquadric Potentials* are discussed in [Volpe et al.], as their

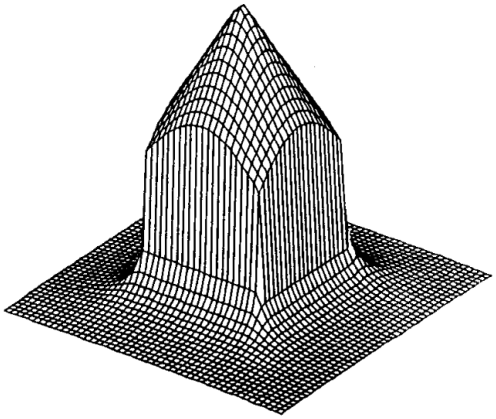


Figure 2.6: A potential field function. **Source**: [Warren].

simple mathematical expression allow for creating *deformable* (see Section 2.4.3) SQ-based potential functions that can vary their shape as a function of distance.

[Khatib] used the APF in real space, but others such as [Warren] and [Volpe et al.] have used it to navigate in the C-space, providing an alternative to graph-searching algorithms like A*. By limiting to real space and avoiding the conversion to C-space, along with the fact that real space has only three dimensions, the algorithm solution time is reduced [Warren]. But the computational cost increases quickly as a function of the manipulator's DoF, at the very least quadratically [Volpe et al.], making them more suitable for offline planing than real-time obstacle avoidance with dynamic obstacles. Regardless, the APF algorithm suffers from a fundamental problem of dealing with "local minima".

The Local Minima Problem

The fields in Figure 2.7 could represent the forces guiding the EE in space, with the goal point creating an attractor field from the point-of-view of the EE and each obstacle creating a form of repulsive field. An unfortunate constellation of obstacle placements could result in certain points having prohibitively low values (depending on factors of thresholding, discretization, etc) following a summation of vectors. In other words, these points represent local minima which a EE, mobile robot, etc could become trapped in ([Lahouar et al.], [Lindemann and LaValle], [Tahir et al.]).

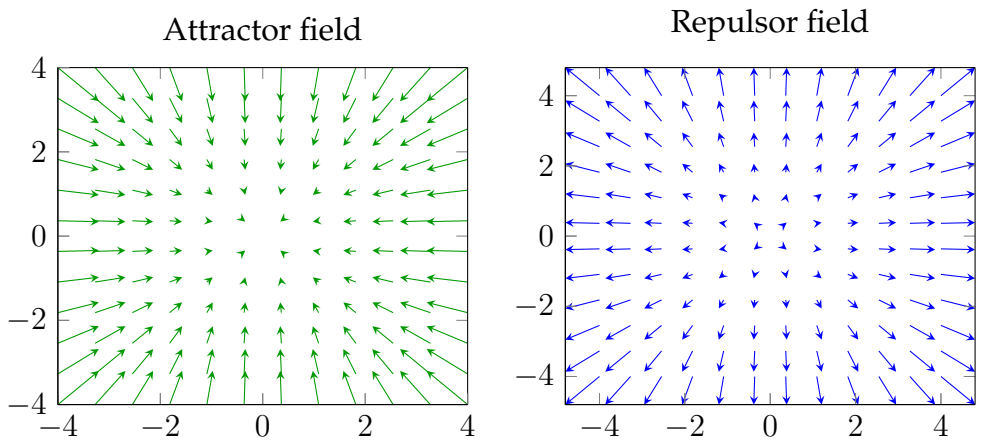


Figure 2.7: Two vector fields.

Note that functions describing these fields are continuous, but in reality discontinuous functions can be used (e.g., Equation 2.11). The ρ_0 variable means that the obstacle will only affect the overall U_{art} within ρ_0 length units, which could mitigate the problem of local minimas. On the other hand, that distance must be properly set in relation to the maximum actuation speed. If ρ_0 is set to low the robot might not be able to decelerate itself in time to avoid impact.

2.3.3 Model Predictive Control

Model Predictive Control (MPC) is slightly different from the other algorithms presented in that it belongs to the field of control, specifically optimal control.

Using an understanding of the internal dynamics of the plant model, a recorded history of the past control inputs u and past measured outputs y, it iteratively explores state trajectories emanating from the current state at time t (continuous) or k (discrete). Then, it produces a cost-minimizing control strategy $u(\tau)$ (for $\tau \in [t, t+T]$) using a cost function J. Afterwards, it discards all the calculated future control moves but the next control move u^+ and implements it, before sampling the plant state again and repeating the process, sliding the time horizon forward ([Hoy et al.], [Wang]).

J is designed to steer the control algorithm. One potential cost function, looking at a simple Linear-Quadratic (LQ) control example in the discrete domain (though MPC is not limited to this problem statement) [Rawlings and Mayne], could be the following,

$$J(x,u) = \frac{1}{2}\sum_{k=0}^{N-1}\left[x(k)'Qx(k) + u(k)'Ru(k)\right] + \frac{1}{2}x(N)'P_f x(N), \quad (2.12)$$

subject to,

$$x^+ = Ax + Bu, \quad (2.13)$$

$$y = Cx + Du. \quad (2.14)$$

Due to the principle of receding horizon control where only current information of the plant is required for prediction and control, it is implicitly assumed that the input $u(k)$ cannot affect the output $y(k)$ at the same time. Thus, in MPC literature ([Hoy et al.], [Wang]) matrix D is typically set as $D = 0$.

The cost functon in Equation 2.12 measures the square deviation of the trajectory of $x(k)$ and $u(k)$ from zero, such that the algorithm will work to minimize them. Matrices Q and R have specific uses as tuning parameters [Rawlings and Mayne].

Large values of Q in comparison to R means that the algorithm will penalize large deviations in state values, resulting in a more aggressive control. The reverse, large values of R relative to Q, represents a wish to keep the magnitude of the control signals small, at the cost of slower convergence. Perhaps, due to concerns of energy consumption or wear and tear on the actuator, this is preferable. P_f is there to specifically target the final state value (of the time horizon) for the sake of generality.

In order to ensure the existence of a unique solution to the optimal control problem, matrices Q, P_f and R are *real* and *symmetric*; Q and P_f are *positive semidefinite* ($\succeq 0$); and R is *positive definite* ($\succ 0$) ([Rawlings and Mayne], [Richter et al.]).

It is worth mentioning that under certain circumstances in linear MPC, such as when the prediction horizon is large, the optimal control trajectory u converges to that of an underlying optimal control trajectory given by a discrete-time LQ-regulator with the same weight matrix Q and R [Wang].

Constraints

The manipulated control inputs to various physical systems tend to be bounded, whether they be valve positions, voltages, torques and so forth [Rawlings and Mayne]. It is possible to encode these hard limits using MPC, in the following manner,

$$Eu(k) \leq e, \tag{2.15}$$

in which,

$$E = \begin{bmatrix} I \\ -I \end{bmatrix}, \qquad e = \begin{bmatrix} u^{max} \\ -u^{min} \end{bmatrix}, \tag{2.16}$$

are used to express simple bounds like,

$$u^{min} \leq u(k) \leq u^{max}, \tag{2.17}$$

where u^{min} and u^{max} represent the lower and upper bounds respectively, and k is a non-negative integer. It is also possible to impose constraints on states or outputs. These can be stated analogously as,

$$Fx(k) \leq f. \tag{2.18}$$

Limits on the control inputs u are typically hard limits imposed by the physical system, whereas the constraints on the state output are typically imposed for reasons of safety, operability, product quality and such [Rawlings and Mayne]. In other words, in the former case, if the controller does not respect the constraints then the physical system will naturally reinforce them. It is also possible to incorporate hard limits on the rate of change, by expanding the state variable with the previous control input value,

$$\tilde{x}(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}, \tag{2.19}$$

creating an augmented system model,

$$\tilde{x}^{+} = \tilde{A}\tilde{x} + \tilde{B}u, \tag{2.20}$$

$$y = \tilde{C}\tilde{x} + Du, \tag{2.21}$$

where,

$$\tilde{A} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}, \qquad \tilde{B} = \begin{bmatrix} B \\ I \end{bmatrix}, \qquad \tilde{C} = \begin{bmatrix} C & 0 \end{bmatrix}. \tag{2.22}$$

A constraint on the rate of change,

$$\Delta^{min} \leq u(k) - u(k-1) \leq \Delta^{max}, \tag{2.23}$$

is then formulated as,

$$F\tilde{x}(k) + Eu(k) \leq e, \tag{2.24}$$

where,

$$F = \begin{bmatrix} 0 & -I \\ 0 & I \end{bmatrix}, \qquad E = \begin{bmatrix} I \\ -I \end{bmatrix}, \qquad e = \begin{bmatrix} \Delta^{max} \\ -\Delta^{min} \end{bmatrix}. \tag{2.25}$$

Linear inequalities like these are useful when looking at linear systems, but the controller of the analysis is not significantly simplified by using them when looking at nonlinear systems [Rawlings and Mayne]. Thus, we might as well generalize the constraints to memberships of sets:

$$x(k) \in \mathbb{X}, \qquad u(k) \in \mathbb{U}, \qquad k \in \mathbb{I}_{\geq 0}, \tag{2.26}$$

where $\mathbb{X}$ is the set of allowable states, $\mathbb{U}$ the set of allowable control inputs, and $\mathbb{I}_{\geq 0}$ the set of non-negative integers.

Without constraints the model predictive control scheme devolves into a state feedback control system whose gain is chosen from minimizing a finite prediction horizon cost [Wang]. Analytical and numerical results can be used to demonstrate the equivalence between the continuous-time MPC and the classical LQ-regulator for this case.

This flexibility in being able to set the constraints is one of the benefits of using MPC. If constraints are not set, it is of vital importance to come up with another way of handling control signal saturation, as it may severely deteriorate the performance. While feasible with Single Input, Single Output (SISO) systems, for Multiple Input, Multiple Output (MIMO) systems the limits of system operation appear in many forms (e.g, limits on each control signal, each control's systems respective $\Delta u(k)$, various state variables, etc). Working out the individual saturation limits in a co-ordinated manner becomes a very complex task, which motivates the use of MPC and the constrained control framework that it brings.

MPC for Guidance

MPC has received interest in vehicle navigation and steering. A strength of MPC in active steering control is that it relies on a model predictive framework, which works to predict the outcome of the system. Due to this ability to project into the future, or "look-ahead", MPC can be utilized to generate obstacle-avoiding trajectories [Park et al.].

The authors of [Marzat et al.] used MPC to guide an autonomous Micro-Air Vehicle (MAV) to avoid obstacles in an cluttered indoor environment. The so-called *Reactive* MPC algorithm consists of having the MAV follow a given reference trajectory defined in terms of positions $\xi(k)^r$ and velocities $V(k)^r$. This can be used regardless if a whole trajectory has been specified previously, or if only the goal point has been set. In the former case, interpolation is done in order to fit the discretization with the MPC time step, and in the latter case the sequence is built with a constant nominal velocity v_{nom} and the MPC time step (with velocity ramps at the beginning and at the end).

At each time step, when producing the control scheme and emanating state trajectory, the algorithm checks if it MAV will collide with an obstacle (or enter within a safety margin to it) and will add a corrective component to the control input that will cause it to deviate from the plan. The authors defined the following equation,

$$u(k) = u(k)^n + \mathbf{1}_k^{obs} u(k)^a \tag{2.27}$$

where $u(k)^n$ is the nominal control signal and $u(k)^a$ a component that will nudge the MAV from its path. Note that this component is dependent on $\mathbf{1}_k^{obs}$, a conditional variable dependent of the collision risk (equal to 0 if no risk of collision exists).

The author of [Fahimi] used Nonlinear Model Predictive Control (NMPC) to do formation control of multiple autonomous surface vessel, integrating local obstacle avoidance. APF, the subject of Section 2.3.2, was used to create an artificial repulsive potential between the marine vessels and obstacles in the cost function of the NMPC. The author showed that the relative distances and orientations of the vessels could be stabilized, even to the point where the surface vessels were able to avoid local obstacles and regain their formation after passing them.

The ability to "look ahead" reduces the possibility of falling into local minima, which has earlier been stated (see Section 2.3.2) to be one of the main disadvantages of using APF [Park et al.].

In [Yoon et al.] obstacle avoidance for wheeled robots in unknown environments was done using NMPC, specifically for car-like unmanned ground vehicles. They also added APF to their cost function, to help it penalize state progressions that bring it closer to obstacles and farther away from the goal. They also experimented with two different methods of designing potential functions, one purely based on distance (between the center of mass and the obstacle) and one based on the modified parallax taking into account the dimension and heading of the vehicle. The modified parallax method potential function was superior, especially in a complex environment. In fact, one could take as a conclusion from [Yoon et al.] that simply relying on the distance can be harmful without any consideration for the nonholonomic constraints (e.g., a speeding car's inability to accelerate sideways, see Section 2.3.4). [Park et al.] also used a parallax-based obstacle avoidance method.

MPC Complexity

MPC falls into the open-loop optimal control classification and compensates for the lack of closed-loop feedback by generating a new solution at every sampling instant at which new state information is available.

Being able to generate new solutions fast, even up to the kHz range, is of vital importance in a number of applications [Richter et al.], including power electronics [Karamanakos et al.] and mechatronic systems [Broeck et al.], and thus it follows that worst case computational complexity needs to be guaranteed.

Though virtually all processes are inherently nonlinear in nature, most MPC applications are based on linearized or uncertain linear dynamic models [Razi and Haeri]. One of the main reasons for control engineers and researchers to make this design choice is because of the high online computational complexity arising from the direct use of nonlinear or non-convex programming techniques. For some problems the nonlinear effects are very significant and need to be taken into account at the control design stage.

As MPC problems posed as minimizing quadratic cost functions with both x and u as variables, subject to linear models of the systems (as in Equation 2.14 with $u \in \mathbb{U}$ defined as a polytopic set, they could also take advantage of the development of fast solution methods for Quadratic Programming (QP) which emerged over the previous decade [Richter and Morari]. This, in combination with the increase of computational power, has led to MPC moving from being limited to slow sampling applications (in the minutes range) to fast sampling ranges (ms to μs range).

In [Richter et al.], the authors further discuss various optimization methods used specifically for constrained LQ MPC problems, and the issue of complexity certification for those. Most assume that the set of $\mathbb{U}$ is a polytopic set, in which case solving for J becomes a parametric, convex QP problem. The vast numbers of methods for these problems can be classified in offline and online solution methods.

For the offline case, methods exist that allow for solutions for each state to be pre-computed. A prevalent solution method is multi-parameter programming [Richter and Morari], which solves the MPC problem offline for every system state in a bounded set. Thus, it only requires a look up operation at runtime. Unfortunately, the complexity of the system increases dramatically with the system size, to the point where approximate solutions will need to be used if the complexity does not become intractable altogether [Richter and Morari], motivating the switch to online methods. Online methods have two main proponents, Interior Point and Active Set methods. Alternative optimization-based approaches do exist, which typically either change the problem formulation with the introduction of conservatism or simplifying assumptions, or that look for an approximate solution only.

The Active Set methods come in different variants, and tailored implementations exploiting the problem structure have been developed and shown to improve the computational speed. The convergence can be ensured in a finite number of iterations, but no practical lower bound on the iteration count exist as the converge rate is unknown. These methods do however perform well in practice and can benefit from warm-starting (i.e. making use of previously computed values in the next MPC iteration) ([Shahzad and Kerrigan], [Richter et al.]).

Interior Point methods also come in different variants and posses the flexibility to tailor them to the specifics of the problem, allowing for significant reductions in complexity. In contrast with the Active Sets methods, there are well-established results for the convergence rate which allow for certifying the computational complexity. However, Interior Point methods are not able to exploit warm-starting [Shahzad and Kerrigan].

The research group in [Karamanakos et al.] discuss linear systems with specifically integer inputs, where some (or all) of the decision variables of the MPC problem are integer-valued. The optimization problem underlying MPC is then NP-hard. Explicit methods can be computed offline, but these typically require significant memory resources to store the explicit control law, as the required memory increases dramatically with the problem size and complexity. For long prediction horizons, or for problems with many integer variables, due to a combinatorial explosion in the number of possible solutions solving the integer optimization problem online might lead to computational intractability. This is problematic as long horizons are often required to ensure stability and good performance. Short sampling intervals only serve to aggravate this issue.

The methods mentioned in Section 2.3.3 for guidance and obstacle avoidance rely on having a large enough time horizon that can project far enough into the future that any potential collisions can be adequately compensated for at the current time without resorting to extreme measures (e.g., an extremely large compensatory actuation signal). A trade-off must thus be made between horizon length and safety.

MPC for Robot Arms

Making a direct comparison between MPC and the other algorithms would not be a fair comparison, as the other comparison do not inherently consider the control aspects of motion planning. Rather, they produce collision-free trajectories which can serve as references for other control algorithms.

The team behind [Guechi et al.] developed both MPC control and LQ optimal control algorithms for a two-link robot arm. They set up the equations for the kinetic and potential energy respectively for the robot arm and combined them in the Lagrange formalism, similar to what

was done in Section 2.3.2, though instead of force it is torque they are concerned with. Despite stating that it is possible to design a NMPC, they nonetheless chose to linearize the system. They showed that their method worked better than only using LQ control.

The authors of [Terry et al.] compared three different methods for linearizing robot manipulator dynamics, and implemented each in an MPC controller in simulation. They also confirmed the sentiment that for complex models, such as manipulators with high DoF, NMPC requires particular adaptions or is often simply too slow for real-time applications. Their three models include: the Taylor Series Linearization method, the Fixed State method, and the Coupling-Torque method; the latter of which is their main contribution. The three linearized MPC models were compared with a baseline NMPC in terms of accuracy, to examine to what extent the simplified models will deviate from reality as the controller projected further into the future.

Similar to [Guechi et al.] and [Terry et al.], [Rovný and Belda] also used Lagrange equations to define their dynamic model of their 5 DoF robot manipulator (mounted on top of their 3 DoF robotic platform). Their proposed MPC motion controller followed a reference motion trajectory made up of time parametrized line and arc segments, a parametrization in alignment with the sampling time of the MPC model. Their model also undergoes modifications enabling the obtainment of a standard linear state space model similar to Equation 2.14.

2.3.4 Rapidly-exploring Random Trees

RRT is an incremental sampling and searching approach originally presented in [LaValle] and specifically designed to handle nonholonomic constraints and high DoF. For single-query planning of high DoF spaces, RRT is the *de facto* standard algorithm [Akgun and Stilman].

Sampling-based Methods

RRT relies on sampling configurations in C-space. It samples them, checks for collisions and then connects them in a tree like structure. It will keep sampling and connecting, producing denser and denser tree structures as shown in Figure 2.8.

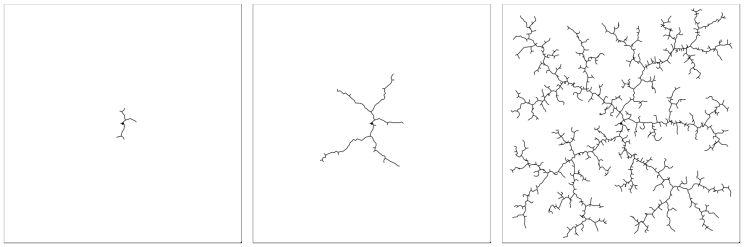


Figure 2.8: The RRT algorithm shown expanding in a uniformly random manner. This implementation lacks a goal state. **Source**: [Cheng et al.].

One of the advantages of sampling based algorithms over many other methods, including APF and A* (which relies on decomposing space into cells) is that it does *not* require an explicit representation of obstacles in C-space [Karaman and Frazzoli]. This is advantageous because such a requirement may lead to excessive computational burden in high DoF systems or scenarios with complex environments featuring many obstacles.

Sampling-based methods avoid explicit representation by relying on a collision checking module. This relaxes the completeness criteria, making the algorithms probabilistic complete, as mentioned in Section 2.3. In other words, the likelihood of an algorithm returning a path provided that one exist goes to 1 as the number of samples go to ∞.

Another randomized sampling-based method that has seen much use is the Probabilistic Roadmap (PRM). PRM and RRT are fundamentally similar, except PRM is a multi-query planner and RRT a single-query planner.

PRM first builds up a large number of configuration points in C-space by (mostly) randomly sampling them, without any idea of what the start and goal state could be. A user will then query the PRM planner, providing a start and goal state, triggering the planner to start working on creating a path of concatenated straight edges from one connecting node to the other. The first part is called the pre-processing phase, and the second part the query phase [Kavraki et al.].

Multi-query planners are appropriate when the environment can be assumed to be static, i.e. the environment does not change in terms of the robot or the obstacles. Obviously, any change in the environment would necessitate the reprocessing of the stored points, either by redo-

ing the collision check or discarding them altogether and starting from scratch.

Single-query planners like RRT are hence more appropriate in dynamic situations, when the query comes and the points are sampled and a path produced as a response to it online.

The Algorithm

Assuming that there is a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges making up graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; as well as a maximum distance d, the algorithm is as follows [Karaman and Frazzoli]:

1. Add the initial point $\mathbf{p}_{initial}$ to $\mathcal{V}$. $\mathcal{E} = \emptyset$.
2. Sample $\mathbf{p}_{random}$.
3. Find the $\mathbf{p}_{nearest}$ nearest point in $\mathcal{V}$.
4. Produce a $\mathbf{p}_{new}$ such that it is a d distance from $\mathbf{p}_{nearest}$ in the direction of $\mathbf{p}_{random}$.
5. **If** the line segment between $\mathbf{p}_{nearest}$ and $\mathbf{p}_{new}$ is collision free: update $\mathcal{V}$ with $\mathbf{p}_{new}$, update $\mathcal{E}$ with the line segment.
   (a) **If** $\mathbf{p}_{new}$ is the goal state (or in a goal region): **return** $\mathcal{G}$.
6. Return to Step 2.

[LaValle] suggested that if the $\mathbf{p}_{nearest}$ produced in Step 3 lies on an edge between two vertices, the edge should be split up and a new vertex be created on that spot. Furthermore, if $\mathbf{p}_{new}$ lies in an obstacle, or if part of the line segment does, the algorithm should travel along the edge to where the line segment enters the obstacle and set a vertex on that boundary ($\mathbf{p}_s$ in Figure 2.9).

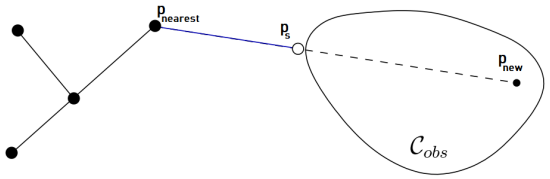


Figure 2.9: If the $\mathbf{p}_{new}$ is in an obstacle region ($\mathcal{C}_{obs}$) it could propose a $\mathbf{p}_s$. **Source**: [LaValle] with modifications.

In Step 5 a collision detection query is made. Considering that it is one of the major focuses of this thesis, it is obviously a very non-trivial step and it will be expounded upon later (e.g., Section 2.4).

How $\mathbf{p}_{random}$ should be sampled is not completely straightforward either. Section 2.3.4 details how one might bias the sampling.

Nonholonomic Constraints and Kinodynamics

A constraint on the k coordinates $\mathbf{r}_1, ..., \mathbf{r}_k$ can be called holonomic if it is a constraint of the form of,

$$g_i(\mathbf{r}_1, ..., \mathbf{r}_k) = 0, i = 1, ..., l. \tag{2.28}$$

If it is not an equality constraint of this kind it is nonholonomic [Spong and Vidyasagar]. An example of a holonomic system would be two particles connected by a rigid beam without mass. Normally, a two particle system would have six DoF (two particles capable of three-dimensional translation), but a DoF is lost due to the fact that the length between the particles is constant [Chung]. If a a constraint cannot be expressed in the form of Equation 2.28 it is nonholonomic, such as constraints given by inequalities are nonholonomic (e.g., a molecule inside of a sphere with radius ρ is subject to $x^2 + y^2 + z^2 < \rho^2$). Mechanical constraints expressed in the following way,

$$g(r, \dot{r}, \ddot{r}, t) = 0, \tag{2.29}$$

are nonholonomic if they cannot be reduced to the form of 2.28, i.e. if they are nonintegrable [Chung].

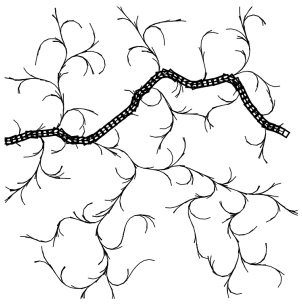


Figure 2.10: A 5D RRT for a kinodynamic car, that has been projected down to two dimensions. **Source**: [LaValle].

In kinodynamic planning, constraints set on the velocity, acceleration, force and torque must be satisfied along with any kinematic goals

(e.g., avoiding an obstacle) [Donald et al.]. Figure 2.10 shows how an RRT tree can be generated that takes into account the kinodynamic properties of a car. A car cannot simply suddenly accelerate to the side, but must roll forward and turn in accordance with its steering system. We can take differential constraints into account by substituting the C-space with that of a $T(C)$-space, which represents the tangent bundle of the C-space [LaValle]. An impossible acceleration would thus lie in a forbidden region, which would be returned by the "obstacle detection" module as illegal.

Sampling

Figure 2.11 shows how the RRT algorithm is naturally biased towards exploring the large Voronoi regions (regions of low node density), effectively breaking up the space it is exploring in. This serves as a visual aide in showing how RRT is probabilistically complete.

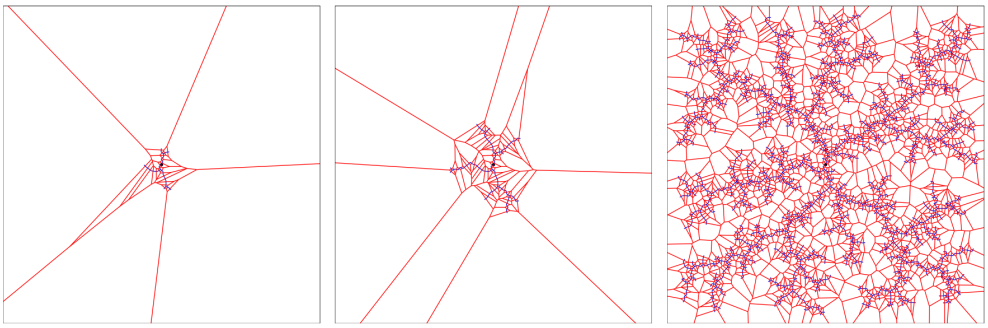


Figure 2.11: The Voronoi regions associated with each uniformly sampled point from Figure 2.8. **Source**: [Cheng et al.].

The vanilla RRT as it has been mentioned so far is by itself an algorithm made to efficiently explore C-space, but not necessarily to solve planning queries. In fact, RRT can have supplementary uses in other algorithms, such as helping to escape local minima [LaValle] in APF, a problem of mentioned in Section 2.3.2.

To make it more suitable, one could grow a tree from an initial configuration $\mathbf{p}_{initial}$ and then periodically try to see if it is possible to connect the RRT to a $\mathbf{p}_{goal}$. One way to achieve this is to do the sample with a 0.01 probability that $\mathbf{p}_{random} = \mathbf{p}_{goal}$. and 0.99 probability that it will be random. $\frac{1}{100}$ has experimentally been shown to prove well [LaValle] [Urmson and Simmons]. Research has gone into ways of biasing the

sampling mechanism without compromising the probabilistic completeness of the algorithm. In general, provided that the mechanism will provide a dense sequence of samples with a probability of 1, meaning that the produced graph gets arbitrarily close to any point in space, any nonuniform sampling method is acceptable [LaValle].

In the field of autonomous driving, [Link et al.] created a variant of RRT called Close-Loop-RRT. They integrated the RRT algorithm in a control loop controlling an autonomous car. They argued that in structured environments (e.g., an urban environment) sampling purely randomly could result in an big number of wasted samples. Instead, they drew their sample from the controller's input space.

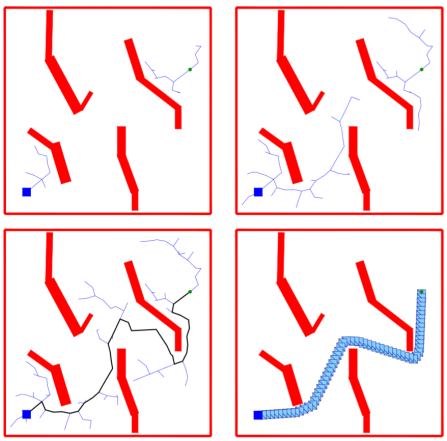


Figure 2.12: The RRT-connect algorithm. Trees expand from the start and goal node, impeded by the red obstacle regions. **Source**: [Kuffner and LaValle].

Variants aiming to help the algorithm converge faster have also been introduced. For example, RRT-Connect (shown in Figure 2.12) works by growing two trees simultaneously, one from the start position and one from the goal [Kuffner and LaValle]. The trees work to both explore the areas around them, as well as to grow towards each other using a greedy heuristic.

Optimality

RRT in its original form is probabilistically complete, but it does not guarantee that the paths produced are in any way optimal. In fact, it has been shown that the solutions the algorithm converges to are almost surely non-optimal (in terms of path length, time cost, etc) [Karaman and Frazzoli]. Thus, variants of the vanilla RRT concerned

with finding optimal solutions have been presented to help address this problem.

RRT* was introduced in [Karaman and Frazzoli] as a way to produce optimal paths from RRT, given a cost function. It adds vertices to $\mathcal{V}$ in the same way as RRT, as well as consider the nearest points in order to return $\mathbf{p}_{nearest}$. The difference is that not all feasible connections will result in new edges being added to $\mathcal{E}$. RRT* maintains a tree structure, and will add the new edge (to $\mathbf{p}_{new}$) in a path that will result in the lowest path cost. Also, new edges from the nearest vertices to $\mathbf{p}_{new}$ are examined, and if lower cost paths can be found through $\mathbf{p}_{new}$ then the algorithm will create new edges and prune away old ones.

One limitation of RRT* is that it is only applicable to systems with simple dynamics, due to relying on the ability to connect any pair of states with an optimal trajectory. [Webb and van den Berg] introduced Kinodynamic RRT* to deal with this, by using a fixed-final-state-free-final-time optimal control formulation.

Transition-based RRT, or T-RRTis another improvement introduced in [Jaillet et al.]. As the name suggest, it makes use of transition tests to judge whether to accept or reject a new potential state, a concept inspired from stochastic optimization (e.g., Monte Carlo optimization) that was developed in order to find global optima in very complex function spaces. T-RRT also relies on a notion of a minimal work path to evaluate path costs.

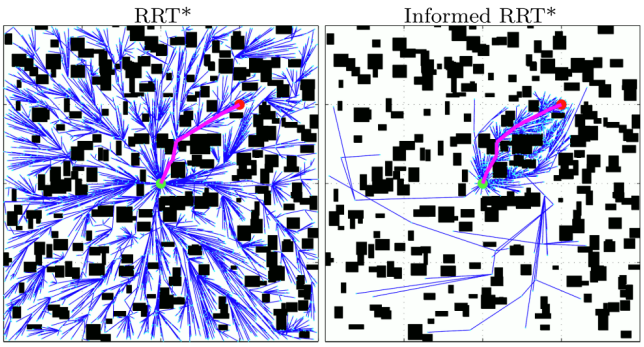


Figure 2.13: A comparison between RRT* and Informed RRT*. **Source**: [Gammell et al.].

Informed RRT* was presented in [Gammell et al.] as an improvement on RRT*. Informed RRT* incorporates search heuristics in order to avoid having to find the optimal path from *every* state, as RRT* does. Figure

2.13 shows RRT* and Informed RRT* finding equivalent (optimal) cost solutions to a goal. In that specific experiment, Informed RRT* required $\frac{1}{8}$ of the time to reach the same path. Informed RRT* narrows down the search on an ellipsoidal informed subset of the state space once an initial solution is found.

2.4 Collision Detection

This section is dedicated to listing various methods for collision detection.

2.4.1 Bounding Box Methods

Bounding Box methods refers to methods that work on the principle of enclosing objects in boxes and then operating on or with those boxes.

The boxes are meant to be minimally encompassing, in the sense that they cover as little beyond their inner objects as possible, along the axes they are defined on. For the 3D case, the box can be formulated as:

$$\mathcal{B} = \{x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}\}. \tag{2.30}$$

$\mathcal{B}$ can be set by sorting through the set of points comprising the object it is to enclose and setting the parameters accordingly. This is the basis for Axis Aligned Boundings Box (AABB).

Axis-Aligned Bounding Boxes

AABB is a simple bounding box method that relies on the Cartesian coordinate axes of the overall world frame. Checking if a point $\mathbf{P}$ is within bounding box $\mathcal{B}$ is as simple as checking that all of the following inequalities are true:

$$\begin{Bmatrix} \mathcal{B}_{x_{min}} \\ \mathcal{B}_{y_{min}} \\ \mathcal{B}_{z_{min}} \end{Bmatrix} \leq \begin{pmatrix} \mathbf{P}_x \\ \mathbf{P}_y \\ \mathbf{P}_z \end{pmatrix} \leq \begin{Bmatrix} \mathcal{B}_{x_{max}} \\ \mathcal{B}_{y_{max}} \\ \mathcal{B}_{z_{max}} \end{Bmatrix}. \tag{2.31}$$

It is simple to extend this to checking if two bounding boxes, cuboids $\mathcal{B}$ and $\mathcal{C}$, are intersecting. In such a case, the following inequalities hold true:

$$\begin{Bmatrix} \mathcal{B}_{x_{min}} \\ \mathcal{B}_{y_{min}} \\ \mathcal{B}_{z_{min}} \end{Bmatrix} \leq \begin{Bmatrix} \mathcal{C}_{x_{max}} \\ \mathcal{C}_{y_{max}} \\ \mathcal{C}_{z_{max}} \end{Bmatrix} \wedge \begin{Bmatrix} \mathcal{B}_{x_{max}} \\ \mathcal{B}_{y_{max}} \\ \mathcal{B}_{z_{max}} \end{Bmatrix} \geq \begin{Bmatrix} \mathcal{C}_{x_{min}} \\ \mathcal{C}_{y_{min}} \\ \mathcal{C}_{z_{min}} \end{Bmatrix} \tag{2.32}$$

Oriented Bounding Boxes

OBB is similar to AABB, except instead of using the world frame it makes use of a local coordinate system associated with the local object itself, or any arbitrarily chosen orientation [Van Den Bergen].

In cases where the object just so happens to not expand in the directions of the Cartesian coordinate axes x, y or z, an AABB will be encapsulating an unnecessary amount of empty space, returning false positives to collision queries. For example, a 1^3 volume units cube rotated $\frac{\pi}{4}$ radians will need a $\sqrt{2}^3 \approx 2.83$ volume units cube to encompass it. The rotated cube would only be occupying $\frac{\sqrt{2}^3 - 1^3}{\sqrt{2}^3} \approx 0.65\%$ of the space. If the axes drawing the bounding box could rotate along with it, 100% of the space would still be used.

This freedom comes at the cost of *storage* space. An OBB is represented using 15 scalars: a 3×3 orientation matrix, 3×1 vector for position and 3 variables for extent [Van Den Bergen]. Meanwhile, an AABB requires only 6, as shown earlier.

The *Separating Axis Theorem* was introduced in [Gottschalk et al.], and is a method used to efficiently check overlap between two OBBs. It is based on the fact that two disjoint convex polytopes in 3D space can always be separated by a plane parallel to either an edge from each polytope, or a face of either polytope. Thus, two convex polytopes are disjoint only if there exists such a separating axis orthogonal to an edge of either polytope, or orthogonal to a face form each polytope. As such, 15 candidate separating axes need to be tested, stemming from the six faces of the OBB and the nine pairwise combination of edges.

If the OBB do not overlap, a separating axis will exist and it will be one of those 15 axes listed. If the OBB are not disjoint, no separating axis will exist.

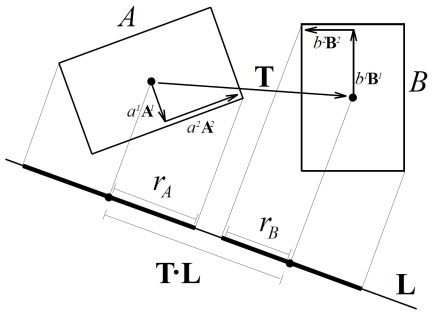


Figure 2.14: Two OBB A and B, along with a separating axis **L**. **Source**: [Gottschalk et al.].

Figure 2.14 illustrates this concept. In that case **L** constitutes a separating axis as the projected versions of A and B do not overlap along it. The projections $\mathbf{r}_A$ and $\mathbf{r}_B$ are the sum of their respective radii (a^1, a^2, b^1, b^2) in their respective axes unit vectors $(\mathbf{A}^1, \mathbf{A}^2, \mathbf{B}^1, \mathbf{B}^2)$ projected onto the separating axis **L**.

Bounding Volume Hierarchy

For more complex environments, one method of speeding up collision detection queries is to structure the bounding volumes into a tree structure [Haverkort]. Objects are recursively wrapped into bounding volumes that form leaf nodes on the tree, as modes are grouped in small sets and enclosed into larger bounding volumes represented as their parent in the tree. When a collision detection query is made, sibling nodes are pairwise checked for collision from top to bottom until a collision is detected.

Take as an example a game containing a world inhabited by two characters, A and B, as shown in Figure 2.15. The naive approach would be to check collisions between all level 7 bounding boxes, of which there are $2^7 = 128$. Checking every bounding box on level n, all with each other, would require $O(2^{n^2})$ checks. A better approach would be to move from the top-most node down, and only doing checks on child nodes of those whose parents are intersecting.

2.4.2 Gilbert-Jonhson-Keerthi Distance Algorithm

The GJK algorithm is an algorithm used for calculating minimum distances between a pair of convex sets [Gilbert and Johnson]. The algorithm was designed to be most efficient for 3D space, where the convex

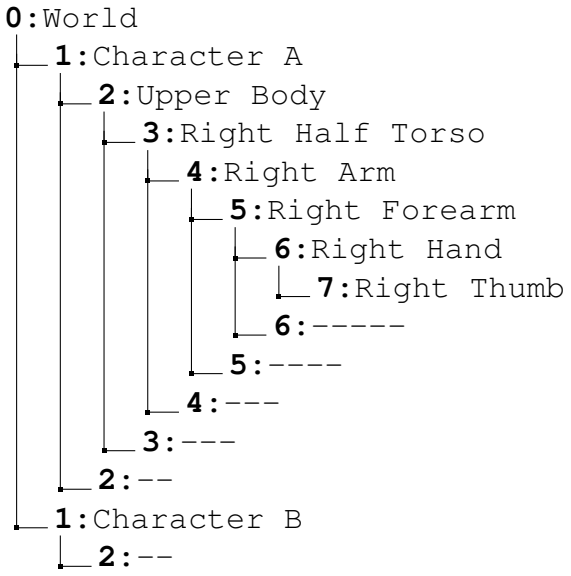


Figure 2.15: A bounding volume hierarchy organizing objects into a tree structure. Two characters inhabit the world, with an anatomy defined in the tree nodes. Dashes are used for brevity, to avoid having to list all child nodes at each level.

sets are polytopes defined by their vertices. The algorithm works iteratively to find the distance, but was shown to terminate in a finite number of steps in such a scenario, growing linearly in the total number of vertices associated with the two polytopes.

An extended version for non-polytopal convex objects (e.g., cones or cylinders) was presented in [Gilbert and Foo]. That version, while fast does not converge in finite time. But it was shown that by setting an effective stopping condition, numerical solutions with guaranteed accuracy could be computed. Extensive numerical experiments supported this claimed efficiency.

The distance between objects A and B, represented by a set of points $\mathcal{K}_A$ and $\mathcal{K}_B$, is given by the following,

$$d(\mathcal{K}_A, \mathcal{K}_B) = \min\{|\mathbf{p}_A - \mathbf{p}_B|^2 : \mathbf{p}_A \in \mathcal{K}_A, \mathbf{p}_B \in \mathcal{K}_B, \mathcal{K}_A, \mathcal{K}_B \subset \mathbb{R}^m\}. \tag{2.33}$$

Provided that $\mathcal{K}_A$ and $\mathcal{K}_B$ are compact sets, i.e. containing all their limits points and having all points $\mathbf{p}_*$ lie in a fixed distance of each other, such a minimum distance will exist [Gilbert and Johnson].

Support Mapping Functions

GJK relies on support mapping functions to describe the convex sets, reducing the problem to computing the distance between one object to the origin [Lindemann]. A support mapping function $s_A(\mathbf{v})$ of a convex set $\mathcal{K}_A$ maps a vector $\mathbf{v}$ to a specific point $\mathbf{p}_A$ known as the *support point*. The support point, $\mathbf{p}_A$, is the point that is located on the most extreme edge point along the boundary of $\mathcal{K}_A$ in the direction of the vector $\mathbf{v}$ [Lindemann], fulfilling the following equation,

$$\mathbf{v} \cdot s_A(\mathbf{v}) = \max\{\mathbf{v} \cdot \mathbf{p}_A : \mathbf{p}_A \in \mathcal{K}_A\}. \tag{2.34}$$

For polytopes, this can be calculated in linear time in regards to the number of vertices, and if frame coherence is used it could potentially be reduced to almost constant time. This could be done by maintaining an adjacency graph of the vertices for the polytope, and is also known as *hill climbing* [Van Den Bergen]. Specific convex sets can have support functions that are particularly well suited for them. For example, the following support function for a circle or sphere returns the support point automatically [Lindemann]:

$$s_A(\mathbf{v}) = c_A + r_A \cdot \frac{\mathbf{v}}{||\mathbf{v}||}, \tag{2.35}$$

where c_A is the center of the circle or sphere and r_A the radius. Refer to [Van Den Bergen] for examples of more support functions.

Support functions can thus be calculated easily, making GJK reliable and fast. Complicated objects can be disassembled into primitives and then have the calculations performed using the sub-parts.

Minkowski Difference

Another point of theory necessary to understand the GJK algorithm is that of the Minkowski sum operation [Gilbert and Johnson]. It is defined for general m-dimensional cases as the following,

$$\mathcal{K}_C = \mathcal{K}_A \pm \mathcal{K}_B = \{\mathbf{p}_A \pm \mathbf{p}_B : \mathbf{p}_A \in \mathcal{K}_A, \mathbf{p}_B \in \mathcal{K}_B, \mathcal{K}_A, \mathcal{K}_B \subset \mathbb{R}^m\}. \tag{2.36}$$

In other words, every point in $\mathcal{K}_B$ is added (or subtracted) to every point in $\mathcal{K}_A$, resulting in a new set of points $\mathcal{K}_C$. Note that this operation

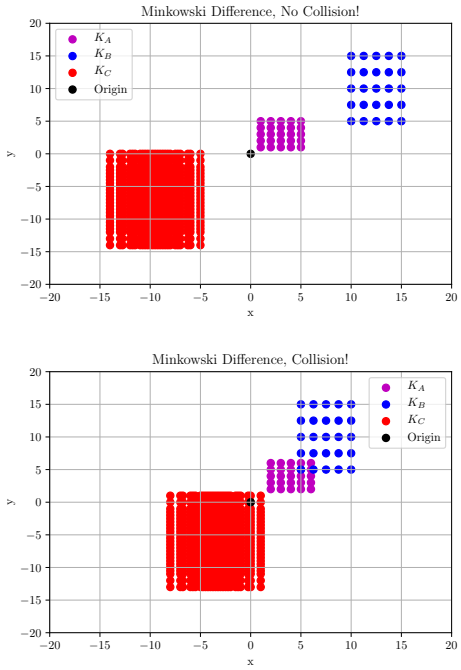


Figure 2.16: Minkowski difference being taken on sets $\mathcal{K}_A$ and $\mathcal{K}_B$ resulting in $\mathcal{K}_C$.

can result in duplicates $\mathbf{p}_C$, so the cardinality of $\mathcal{K}_C$ is always equal to or less than that of $\mathcal{K}_A$ and $\mathcal{K}_B$.

The Minkowski sum, or rather the Minkowski *difference* specifically, is the key to GJK, as it can be shown that the shortest distance difference between $\mathcal{K}_A$ and $\mathcal{K}_B$ is equal to the closest distance between $\mathcal{K}_C$ and the origin [Lindemann]. The support mapping functions can similarly be calculated as,

$$s_C(\mathbf{v}) = s_{A\pm B}(\mathbf{v}) = s_A(\mathbf{v}) \pm s_B(\pm\mathbf{v}). \tag{2.37}$$

Thus, if two objects are intersecting (i.e., colliding), the $\mathbf{p}_C$ closest to the origin will be the origin itself. In other words, if the origin is a part of the Minkowski difference $\mathcal{K}_C$, $\mathcal{K}_A$ and $\mathcal{K}_B$ are in collision. This is shown in Figure 2.16, where in the upper scenario the origin is not a point in $\mathcal{K}_C$ but in the lower scenario it is, due to the intersection around $(5, 5)$.

The Iterative Algorithm

For the purposes of this algorithm, for a given set $\mathcal{A}$ and given set of vertices $\mathcal{W}$, $\mathbf{v}(\mathcal{A})$ is meant to return the point closest to the origin in set

$\mathcal{A}$, and $CH(\mathcal{W})$ refers to the convex hull spanned by the set of vertices $\mathcal{W}$.

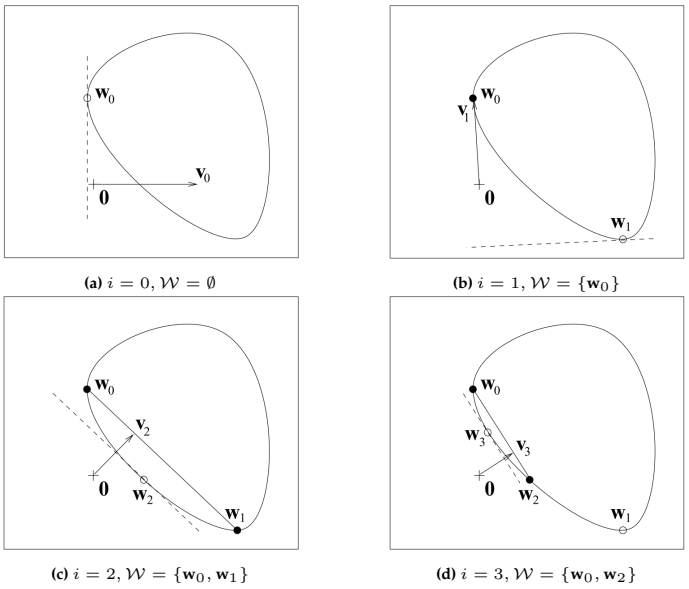


Figure 2.17: Four iterations of the GJK algorithm. Note that the origin is not part of the set. **Source**: [Van Den Bergen].

As mentioned, the GJK works iteratively. It produces a simplex in K_C that is closer and closer to the origin in each step. The simplex is composed of a set of vertices, stored in $\mathcal{W}_i$, where i represents the ith iteration of the algorithm. The simplex computed in each iteration has a convex hull, on which lies $\mathbf{v}_i$ defined as the point closest to the origin ($\mathbf{v}_i = \mathbf{v}(CH(W_i))$) in iteration i. The algorithm is then as follows [Lindemann]:

1. Initialization step:
   (a) Let i = 0.
   (b) Define the simplex set $\mathcal{W}_0 = \emptyset$.
   (c) Let $\mathbf{v}_0$ be an arbitrary point within K_C.
2. Compute the support point $\mathbf{w}_i = s_C(-\mathbf{v}_i)$ in direction $-\mathbf{v}_i$.
3. **If** $\mathbf{w}_i$ is no more extreme than $\mathbf{v}_i$ in direction $-\mathbf{v}_i$: **return** $||\mathbf{v}_i||$.
4. **Else**: add $\mathbf{w}_i$ to the current simplex set $\mathcal{W}_i$.
5. Compute $\mathbf{v}_{i+1} = \mathbf{v}(CH(W_i \cup \{\mathbf{w}_i\}))$.

6. Make $\mathcal{W}_{i+1}$ the smallest convex subset of $\mathcal{W}_i \cup \{\mathbf{w}_i\}$ still containing $\mathbf{v}_{i+1}$.
7. i_{++}, go to Step 2.

Figure 2.17 presents four iterations of the algorithm for the reader's convenience.

In Step 3 the function will produce the distance to the origin, for which a value of 0 (or $||\mathbf{v}_i|| < \epsilon$, for a small ϵ) would imply collision.

Step 5 computes $\mathbf{v}_{i+1}$, but this is not exactly a trivial task. Originally *Johnson's Distance Subalgorithm* [Gilbert and Johnson] was used, it searches all simplex subsets and solves systems of linear equations for each subset recursively [Ericson]. Advances over time have however made it obsolete [Lindemann].

One alternative algorithm was provided in [Cameron], which also makes use of support mapping functions and the aforementioned hill climbing method. It buffers the support point from the last iteration, and then checks its neighboring vertices first in the current iteration. For larger convex hulls it brought dramatic improvements.

Another alternative method was presented in [Ericson] is mathematically equivalent to the *Johnson's Distance Subalgorithm*, but relies instead on a geometrical approach that the author claims to be more intuitive. It relies on dividing the space into Voronoi regions defined by the points and the edges, as seen in Figure 2.18.

1. If the origin lies in a vertex's region, that vertex is the closest point.
2. If the origin lies in an edge's region, the closest point is somewhere on that edge. The closest point can be calculated using the orthogonal projection on that edge (which is vector between two points).
3. If the origin is in neither a vertex region or edge region, it must be in the face region.

It applies to the 2D scenario of finding intersecting polytopes, but can be generalized to be used in 3D. In 3D, for example, a point could exist above or below the triangle *F*, in addition to being perfectly in its plane.

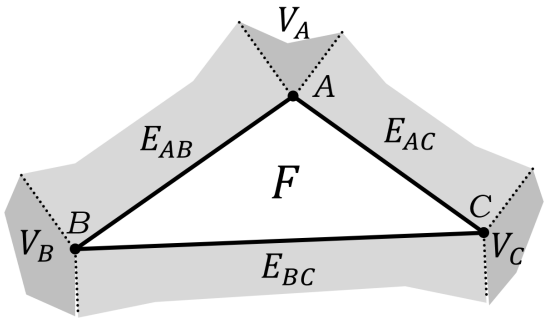


Figure 2.18: An illustration of convex hull with face F, the vertices $\mathbf{A} = \mathbf{q}_1$, $\mathbf{B} = \mathbf{q}_2$ and $\mathbf{C} = \mathbf{q}_3$, as well as the Voronoi regions (**E** for edge, **V** for vertex) $\mathbf{E}_{AB}, \mathbf{E}_{AC}, \mathbf{E}_{BC}, \mathbf{V}_A, \mathbf{V}_B$ and $\mathbf{V}_C$. **Source**: [Ericson].

2.4.3 Superquadrics

SQ are a family of 3D geometric shapes. These shapes are actually surfaces obtained by taking the spherical product of two 2D curves, specifically that of the superellipse presented in its parametric form:

$$\mathbf{s}(\theta) = \begin{bmatrix} a\, cos^{\epsilon}\, \eta \\ b\, sin^{\epsilon}\, \eta \end{bmatrix}, \quad -\pi \leq \theta \leq \pi, \tag{2.38}$$

or, in its implicit form,

$$\left(\frac{x}{a}\right)^{\frac{2}{\epsilon}} + \left(\frac{y}{b}\right)^{\frac{2}{\epsilon}} = 1. \tag{2.39}$$

Note that in the parametric form, taking the exponentiation of the trigonometric functions with ϵ is defined as a signed power function, such that $cos^{\epsilon}\theta = sign(cos\theta)|cos\theta|^{\epsilon}$ and similarly for the sine equivalent.

By taking the spherical product [H.Barr] of a pair of superellipses, a *superellipsoid* can be obtained:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = s_1(\eta) \otimes s_2(\omega) = \begin{bmatrix} cos^{\epsilon_1}\eta \\ a_3\ sin^{\epsilon_1}\eta \end{bmatrix} \otimes \begin{bmatrix} a_1\, cos^{\epsilon_2}\omega \\ a_2\, sin^{\epsilon_2}\omega \end{bmatrix} = \begin{bmatrix} a_1\, cos^{\epsilon_1}\eta\, cos^{\epsilon_2}\omega \\ a_2\, cos^{\epsilon_1}\eta\, sin^{\epsilon_2}\omega \\ a_3\, sin^{\epsilon_1}\eta \end{bmatrix}, \quad \begin{matrix} -\pi/2 \leq \eta \leq \pi/2 \\ -\pi \leq \omega \leq \pi \end{matrix}. \tag{2.40}$$

Geometrically, $s_1(\eta)$ is a horizontal curve that is vertically modulated by $s_2(\omega)$. Angle η can be seen as a north-south parameter (not unlike

latitude), and ω corresponds to an east-west parameter (like longitude) [H.Barr]. The dimension parameters a_1, a_2, a_3 scale the spherical product in the respective axes. The shape parameters ϵ_1 and ϵ_2 are also known as the squareness parameters, associated with the north-south and east-west directions respectively. They can be used to pinch, round and square off the shape.

Cuboids are produced when $\epsilon_1 < 1$ and $\epsilon_2 < 1$, cylindroids when $\epsilon_2 \sim 1$ and $\epsilon_1 < 1$ and pillow-shapes when $\epsilon_1 \sim 1$ and $\epsilon_2 < 1$. When either ϵ_1 or ϵ_2 are greater than 2 pinched shapes are produced, and flat-beveled shapes are produced when either $\epsilon_1 = 2$ or $\epsilon_2 = 2$.

The implicit function for this *superellipsoid* can be written as follows,

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} = F_E(\mathbf{P}, \mathbf{\Lambda}). \tag{2.41}$$

where $\mathbf{P}$ refers to the point $(x, y, z)^T$ and $\mathbf{\Lambda}_5$ the parameters $a_1, a_2, a_3, \epsilon_1, \epsilon_2$. Refer to [Jaklič et al.] for the details of the conversions between the parametric forms and implicit forms of the respective shapes.

All the points P that satisfy $F = 1$ lie on the surface. If a point results in $F < 1$ or $F > 1$ then the point lies inside or outside of the surface respectively. Because the implicit function divides 3D space into three distinct regions (inside, outside, surface boundary), another word for it is *inside-outside* function.

A range of superellipsoidal SQ have been plotted using Equation 2.40 in Figure 2.19.

Non-Superellipsoidal Superquadrics

As mentioned, the specific F function F_E of Equation 2.41 applies only to superellipsoids, but the SQ term encompasses more than just the superellepsoid shape. Other shapes include supertoroids, superhyperboloids of one sheet and superhyperboloids of two sheets [Jaklič et al.]. It is common in literature to use SQ as a synomym for superellipsoids only unless otherwise specified and it also the case in this thesis.

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{\epsilon_1}} - \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} = F_{H1}(P, \mathbf{\Lambda}_5) \tag{2.42}$$

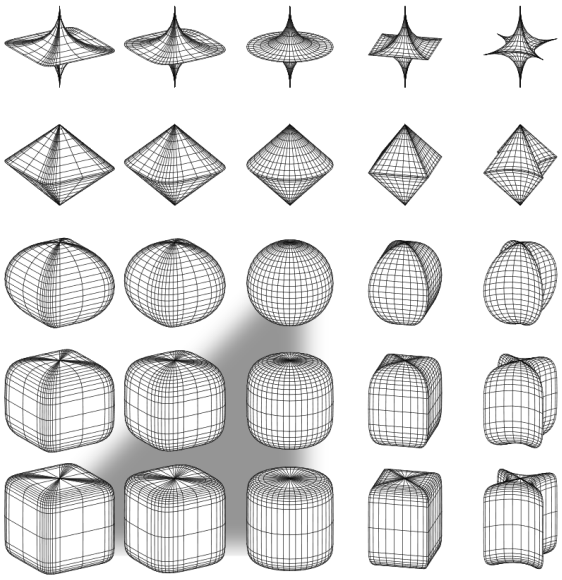


Figure 2.19: SQ (superellipsoidals) drawn, for constant a_1, a_2, a_3 but varying ϵ_1 (increasing in the figure's y-axis) and ϵ_2 (increasing in the figure's x-axis). $\epsilon_1, \epsilon_2 = \{\frac{1}{4}, \frac{1}{2}, 1, 2, 4\}$. **Source**: [Kindlmann].

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} - \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{\epsilon_1}} - \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} = F_{H2}(P, \mathbf{\Lambda}_5) \tag{2.43}$$

$$\left(\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} - \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{2}} - a_4\right)^{\frac{2}{\epsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} = F_T(P, \mathbf{\Lambda}_5) \tag{2.44}$$

Equation 2.42 defines the inside-outside function for a superhyperboloid of one sheet, Equation 2.43 defines the inside-outside function for a superhyperboloid of two sheet, and Equation 2.44 defines the inside-outside function of a supertoroid [H.Barr].

Superhyperboloids of one or two sheets are for the purposes of this thesis not considered useful as they do not encapsulate any useful internal volumes like superellipsoids and supertoroids do. The explicit function for supertoroids is as following,

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} (a_1 + cos^{\epsilon_1}\eta)cos^{\epsilon_2}\omega \\ (a_2 + cos^{\epsilon_1}\eta)sin^{\epsilon_2}\omega \\ a_3\, sin^{\epsilon_1}\eta \end{bmatrix}, \begin{matrix} -\pi \leq \eta \leq \pi \\ -\pi \leq \omega \leq \pi. \end{matrix} \tag{2.45}$$

To provide an idea of how a supertoroid looks like, a supertoroid with shape parameters $\epsilon_1 = \epsilon_2 = 1$ is donut-shaped. They could be used to model a robot's EE. As mentioned, unless explicitly stated otherwise, the main SQ focus will be on superellipsoid and future equations (e.g., the transformation matrix in Section 2.4.3 and its inverse) will be applicable to its implicit and explicit functions.

Translation and Rotation

The SQ equations mentioned thus far all described the shapes in standard positions and orientations, but it is possible to translate and rotate them.

Previously only five parameters $(a_1, a_2, a_3, \epsilon_1, \epsilon_2)$ were necessary, six more parameters are needed to express the translation and rotation of the SQ in the global coordinate system. [Jaklič et al.].

$$\begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} = \mathbf{T} \begin{pmatrix} x_s \\ y_s \\ z_s \\ 1 \end{pmatrix}. \tag{2.46}$$

Using a homogenous transformation, Equation 2.46 shows how it is possible to transform a point coordinate from the SQ centered coordinate system $(x_s, y_s, z_s)^T$ to the world coordinate system (x_w, y_w, z_w) using $\mathbf{T}$,

$$\mathbf{T} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{P}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & q_x & p_x \\ n_y & o_y & q_y & p_y \\ n_z & o_z & q_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.47}$$

$\mathbf{T}$ thus translates (p_*) and rotates (n_*, o_*, q_*) the point. As it is sometimes needed for world points to be expressed in SQ centered coordinates for the inside-outside function, the inverse transformation $\mathbf{T}^{-1}$ is used,

$$\mathbf{T}^{-1} = \begin{bmatrix} n_x & n_y & n_z & -(p_x\, n_x + p_y\, n_y + p_z\, n_z) \\ o_x & o_y & o_y & -(p_x\, o_x + p_y\, o_y + p_z\, o_z) \\ q_x & q_y & q_z & -(p_x\, q_x + p_y\, q_y + p_z\, q_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.48}$$

By substituting the SQ centered coordinates, defined with the inverse transformation from Equation 2.48, into Equation 2.41 a new inside-

outside function can be defined:

$$F(x_w, y_w, z_w) = \left[\left(\frac{n_x x_w + n_y y_w + n_z z_w - p_x n_x - p_y n_y - p_z n_z}{a_1} \right)^{\frac{2}{\epsilon_2}} + \left(\frac{o_x x_w + o_y y_w + o_z z_w - p_x o_x - p_y o_y - p_z o_z}{a_2} \right)^{\frac{2}{\epsilon_2}} \right]^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{q_x x_w + q_y y_w + q_z z_w - p_x q_x - p_y q_y - p_z q_z}{a_3} \right)^{\frac{2}{\epsilon_1}}. \quad (2.49)$$

The next step is to replace n_*, o_* and q_* with elements of a rotation matrix $\mathbf{R}$. This can be done using Euler angles [Siciliano and Khatib]. The idea is to take a rotation ψ around the x-axis, a rotation θ around the y-axis and a rotation ϕ around the z-axis.

$$\mathbf{R}(\phi, \theta, \psi) = \mathbf{R}_{Z_\psi} \mathbf{R}_{Y_\theta} \mathbf{R}_{X_\phi} \quad (2.50)$$

where,

$$\mathbf{R}_{X_\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\phi & -sin\,\phi \\ 0 & sin\,\phi & cos\,\phi \end{bmatrix}, \quad (2.51)$$

$$\mathbf{R}_{Y_\theta} = \begin{bmatrix} cos\,\theta & 0 & sin\,\theta \\ 0 & 1 & 0 \\ -sin\,\theta & 0 & cos\,\theta \end{bmatrix}, \quad (2.52)$$

and

$$\mathbf{R}_{Z_\psi} = \begin{bmatrix} cos\,\psi & -sin\,\psi & 0 \\ sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.53)$$

Substituting $\mathbf{R}(\phi, \theta, \psi)$ into $\mathbf{T}$ results in the following transformation matrix,

$$\mathbf{T} = \begin{bmatrix} cos\phi\, cos\theta & cos\phi\, sin\theta\, sin\psi - sin\phi\, cos\psi & cos\phi\, sin\theta\, cos\psi + sin\phi\, sin\psi & p_x \\ sin\phi\, cos\theta & sin\phi\, sin\theta\, sin\psi + cos\phi\, cos\psi & sin\phi\, sin\theta\, cos\psi - cos\phi\, sin\psi & p_y \\ -sin\phi & cos\theta\, sin\psi & cos\theta\, cos\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.54)$$

Thus, we have increased the number of parameters of the inside-outside function from five to eleven. See Figure 2.20 for an illustration of

a translated and rotated SQ. We can refer to these parameters, i.e. $a_1, a_2, a_3, \epsilon_1, \epsilon_2, \phi, \theta, \psi, p_x, p_y, p_z$, as the set $\{\lambda_1, \lambda_2, ..., \lambda_{11}\} = \mathbf{\Lambda}_{11} \equiv \mathbf{\Lambda}$.

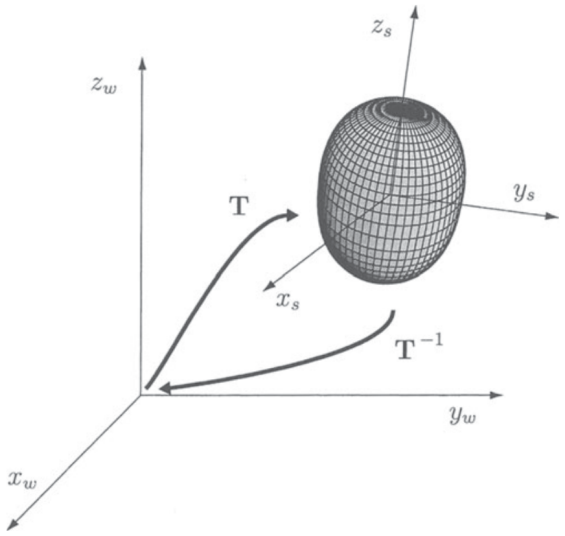


Figure 2.20: A SQ that has been translated and rotated by a transformation **T**. **Source**: [Jaklič et al.].

Deformable Superquadrics

Beyond simply translating and rotating a SQ, it is also possible to deform them. It was shown in in [Jaklič et al.] that SQ can be *tapered*, *bent* and *twisted*.

Tapering involves the gradually thininng or expansion of an object along a chosen dimension. It is defined as a deformation as a function of z,

$$x_D = f_x(z)\, x_U \tag{2.55}$$
$$y_D = f_y(z)\, y_U \tag{2.56}$$
$$z_D = z_U, \tag{2.57}$$

where x_U, y_U, z_U are components of an original undeformed surface vector $\mathbf{x}_U$ and x_D, y_D, z_D the components of a surface vector $\mathbf{x}_D$ belonging to the deformed SQ. The tapering functions f_x, f_y work in the x_S-axis and y_S-axis directions of the SQ centered coordinate system. In other words, the functions (linearly) offset points in the x_S and y_S directions respectively along the surface as a function of z (along z_S).

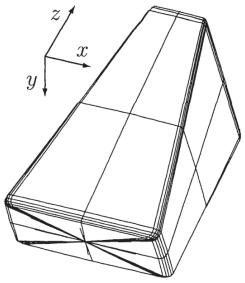


Figure 2.21: Tapered SQ along two axes. **Source**: [Jaklič et al.].

For linear tapering, the two tapering functions are defined as,

$$f_x(z) = \frac{K_x}{a_3}z + 1, \quad -1 \leq K_x \tag{2.58}$$

$$f_y(z) = \frac{K_y}{a_3}z + 1, \quad K_y \leq 1. \tag{2.59}$$

Thus, the two parameters K_x and K_y are necessary to define the (linear) tapering deformation process, corresponding to the tapering degree in each axial direction. Figure 2.21 shows an SQ that has been tapered by a $f_x(z)$ and a $f_y(z)$.

Bending involves transforming a straight line into a circular section, such as that of a z_S-axis. Note that the line segment keeps its length, which does not necessarily correspond to shape deformations when actual physical objects are bent. The bending plane is rotated around the z_S-axis to an angular position defined by angle α (in the $x_S - y_S$-plane). The x_U and y_U coordinates are projected to the bending plane ($x_U, y_U \rightarrow r$), deformed (bent) using bending angle γ in the $x_S - z_S$ or $y_S - z_S$ planes ($r \rightarrow R$), and then projected back ($R \rightarrow (x_D, y_D)$). The bending angle γ is a function of the curvature parameter k and where on the z_S-axis it is being evaluated.

Figure 2.22 shows how one SQ can look after being bent in three different planes. The transformation from bent (deformed) to unbent is given by,

$$x_U = x_D - (R - r)\, cos\alpha, \tag{2.60}$$

$$y_U = y_D - (R - r)\, sin\alpha, \tag{2.61}$$

$$z_U = \frac{1}{k}\gamma, \tag{2.62}$$

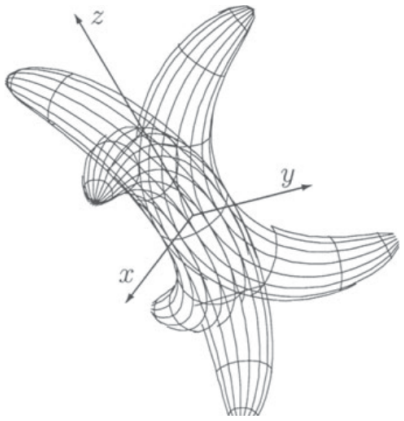


Figure 2.22: A SQ that has been bent in three different bending planes, each of which have been rotated around the z-axis. **Source**: [Jaklič et al.].

where,

$$\gamma = tan^{-1}\frac{z_D}{\frac{1}{k} - R}, \tag{2.63}$$

$$r = \frac{1}{k} - \sqrt{z_D^2 + (\frac{1}{K} - R)^2}, \tag{2.64}$$

$$R = \sqrt{x_D^2 + y_D^2} cos\left(\alpha - tan^{-1}\frac{y_D}{x_D}\right). \tag{2.65}$$

Thus, the two parameters used for doing bending are α and k. It can be interpreted that α is the angle around z_S that decides the curving *direction*, while k decides *how much* the SQ will actually bend. Refer to [Jaklič et al.] for details regarding the derivations of these equations.

Twisting is another global deformation that can be applied on a SQ. It can be likened to twisting a deck of cards, in that the twisting happens along one axis but does not have an effect in the other axes. If the z_S-axis is once again used as an example, the global twist around it can be produced using the following equations [Barr],

$$x_D = x_U\, cos\xi - y_U\, sin\xi, \tag{2.66}$$

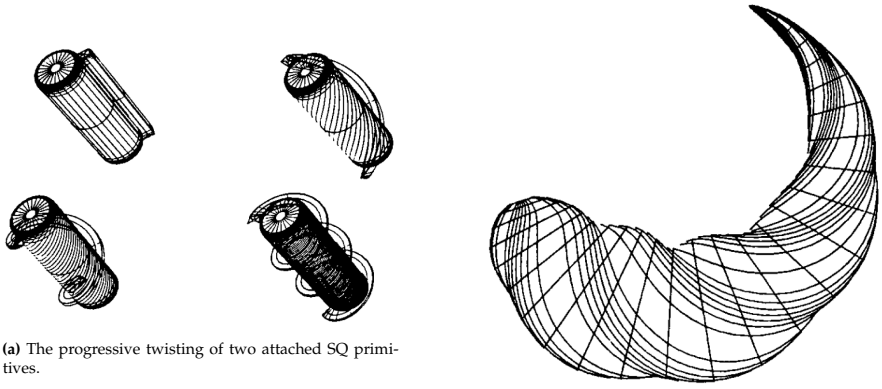
$$y_D = x_U\, sin\xi + y_U\, cos\xi, \tag{2.67}$$

$$z_D = z_U, \tag{2.68}$$

where crucially,

$$\xi = f(z_U) = f(z_D). \tag{2.69}$$

Figure 2.23a clarifies the twisting deformation.



(a) The progressive twisting of two attached SQ primitives.

(b) A bent, twisted and tapered SQ.

Figure 2.23: Source:[Barr]

Any global deformations should be performed before performing translation and rotation. Also, note that tapering, bending and twisting (shown together in Figure 2.23) are not commutative operations and as such the order matters. In addition to these global deformations, it is also possible to do local deformations [Jaklič et al.].

Fitting Superquadric Parameters to Range Data

While it is perfectly possible to manually tune the $\mathbf{\Lambda}$ parameters in order to define the specific surface shape wanted, it is also possible to fit a SQ to a cloud of points lying on the surface of an object. This can be re-framed as an optimization problem consisting of finding the SQ that best represents an object's surface, minimizing the closest distance to each point in the point cloud.

The optimization function is as follows,

$$\min_{\lambda} \sum_{i=1}^{N} \left(\sqrt{\lambda_1 \, \lambda_2 \, \lambda_3} \, (F(x_i, y_i, z_i; \lambda_1, ..., \lambda_{11}) - 1) \right)^2 \tag{2.70}$$

for N number of points.

$(F(\mathbf{P}, \mathbf{\Lambda}) - 1)^2$ is at its minimum point when the point $\mathbf{P}$ lies on the surface of the SQ defined by $\mathbf{\Lambda}$. It features in the loss function, though the reverse is being done: finding the $\mathbf{\Lambda}$ that best fits the set of all $\mathbf{P}$ points, i.e. which keeps the points as close as possible to the surface.

Due to the object self-occluding itself, it is obviously not possible to view all sides of it at the same time. The exact shape of the object

can naturally not be recovered from such "degenerate" view points. Even when a "general" viewpoint is assumed, objects with surfaces where at least one principal curvature equals zero (e.g. cylinders or parallelepipeds) cannot provide sufficient constraints in order to limit the shape recovery to a unique result using only the inside-outside function by itself. The next best thing that can be done is to simply aim to reconstruct the SQ that is the smallest. This is done by including the square root of the product of the axes scaling parameters a_1, a_2, a_3 (or $\lambda_1, \lambda_2, \lambda_3$), in the cost function [Jaklič et al.].

Finally, it has been suggested by [Jaklič et al.] that replacing F with F^{ϵ_1} (or $F^{\frac{\epsilon_1}{2}}$) would make the fitting function more suited for rapid convergence as it would make the loss function independent from the shape of the SQ controlled by ϵ_1. Compare it with Equation 2.41. This formulation has also been present in other works ([Tzovaras and Strintzis], [Pascoal et al.]). [Jaklič et al.] argue that since $F = 1$ then $F^{\epsilon_1} = 1^{\epsilon_1}$ and the shape that is optimal will not be different. If the intention is to use the inside-outside implicit function for purposes other than as a simple "inside $F < 1$, outside $F > 1$, on surface $F = 1$" check, it will have an effect. In some works however, such as in [Vezzani et al.] and its related work [Fantacci et al.], the authors did not seem to use this trick.

As this is a nonlinear least squares minimization problem, the Levenberg-Marquardt method can used to solve it. Setting good initial values, if possible, is helpful to getting good performance. Also, SQ with $\epsilon_1, \epsilon_2 > 2$ have concavities which make the form non-convex and can result in singularities [Jaklič et al.]. Thus, inequalities limiting this should be set. Of course, it is not necessary to limit the SQ in this way if the parameters are manually set by a human.

For the recovery of deformed SQ, the procedure is largely the same, though the initial values should be corresponding to non-deformed SQ (e.g., for tapering set $K_x, K_y = 0$). For bending, make sure to set k to something very small but not equal to 0, as that would result in a singularity.

Chapter 3

Implementation

Using the theory provided in the previous chapter the methods are implemented and certain design choices explained. As mentioned in Chapter 1, software for doing collision detection with SQ and GJK as well as the RRT motion planning algorithm are made. A* is not implemented.

3.1 Superquadrics

Using SQ to do collision checking involves using the implicit inside-outside function to check if a specific point lies inside the SQ primitive or not. From a memory point of view, one non-deformed SQ that has been translated and rotated (collectively referred to *transformed*)in space requires eleven parameters to be represented. The transformation operation itself is represented by a 4×4 matrix (Equation 2.54). Note that SQ deformations are not implemented.

Since each SQ has one $\mathbf{T}$ matrix, it is used as the SQ state holder in the sense that it contains all of the information necessary to transform the SQ from the origin to wherever in space. For plotting purposes, producing the explicit representation using Equation 2.40 and then doing matrix multiplication with that in accordance with 2.46 will produce points transformed to their designated space.

Successive transformations can be calculated by $\mathbf{T}_{New} = \mathbf{T}_n, \ldots, \mathbf{T}_2\,\mathbf{T}_1$ and can from there be assigned as the new state of the SQ in question.

This is is useful for creating arms composed of SQ linked in succession. The new inside-outside function is given by Equation 2.49, relying on the notation defined in Equation 2.47 for the elements of **T**.

Linking several SQ together to form an arm with a base and links (as in Figure 3.1) is done in the following way:

1. Initialize all the SQ links and their base in the origin.
2. For L_i the *i*:th link in the arm (following the base):

   2.1 $L_i.\mathbf{T} \leftarrow L_{i-1}.\mathbf{T}$

   2.2 $L_i.\mathbf{T} \leftarrow L_i.Rotate([\phi, \theta, \psi]) \cdot L_i.\mathbf{T}$

   2.3 $L_i.\mathbf{T} \leftarrow L_i.Translate([p_x, p_y, p_z]) \cdot L_i.\mathbf{T}$

Each link has its transformation matrix as an attribute accessed by ".**T**". Note that the $\cdot$ operation is used here to denote matrix multiplication. Mayavi [van der Walt et al.] is used to do the visualizations in this thesis, as seen in Figure 3.1.

Functions $Translate([p_x, p_y, p_z])$ and $Rotate([\phi, \theta, \psi])$ refer to creating a transformation matrix **T** that purely translates and purely rotates respectively. In the case of this algorithm specifically, $[\phi, \theta, \psi]$ can be seen as angle commands for the specific joint, and $[p_x, p_y, p_z]$ is defined as the vector spanning from the top of the previous link to the bottom of the current one (obtained in practice by subtracting the center coordinates of the top and bottom surfaces).

Expressed in words, this algorithm will assign the previous link's transformation matrix to the current one, shifting all the points to where the previous link is located. Then, it will rotate in accordance to outside commands before shifting the current link to connect with the top of the previous link.

3.1.1 Intersection Checking

Doing intersection checks with SQ requires storing one of the objects implicitly, and the other explicitly. Then, the explicitly stored object has its points looped through the implicit object's inside-outside function, to see if one of them is inside the other object.

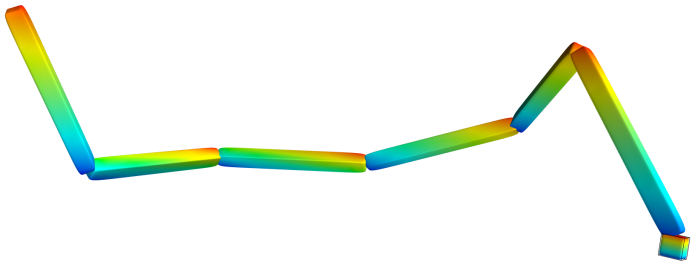


Figure 3.1: A seven link arm.

An engineer making use of this method must choose whether the obstacles should be defined explicitly and the robotic manipulator's links implicitly, or if the obstacles should be defined implicitly and the robotic manipulator's links implicitly. The choice will depend on several factors, including how many (relevant) obstacles there are in comparison to the links, which one needs to be perfectly mathematically modeled (i.e. implicitly, to capture a completely accurate) versus which one is okay sample points explicitly, and so on.

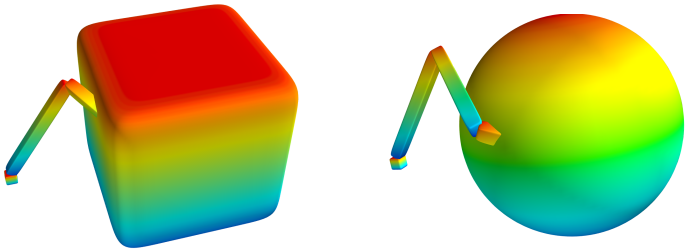
If one class of objects is very large, in order to ensure that the same relative resolution between the obstacles and the links is preserved, a larger number of points will need to be sampled, creating a case for that specific class being implicitly defined. In other words, maintaining 1 centimeter resolution on a 1×1 meter grid requires 100^2 points, and expanding that to a 10×10 meter grid will result in 1000^2 points being required to maintain the same resolution. Such an object is better off being represented implicitly, as that only requires 12 SQ parameters and the transformation matrix.

3.1.2 The Problem of Uniform Angle Sampling

Explicitly representing the SQ by naively producing points through uniform sampling points of the angles η and ω might result in some unwanted behavior.

Producing the coordinates in accordance with Equation 2.40 by uniformly sampling the angles (from $\frac{-\pi}{2}$ to $\frac{\pi}{2}$ and from $-\pi$ to π respectively) will not necessarily result in uniformly spread points in the real world, especially for ϵ-values below 1. This issue was discovered while testing

the collision detection method for a four link arm, as shown in Figure 3.2.



(a) Box-shaped ($\epsilon_1 = \epsilon_2 = 0.2$) SQ obstacle. The algorithm failed to register the collision.

(b) Spherical ($\epsilon_1 = \epsilon_2 = 1$) SQ obstacle. The algorithm correctly registered collision.

Figure 3.2: Collision detection testing with a three link arm.

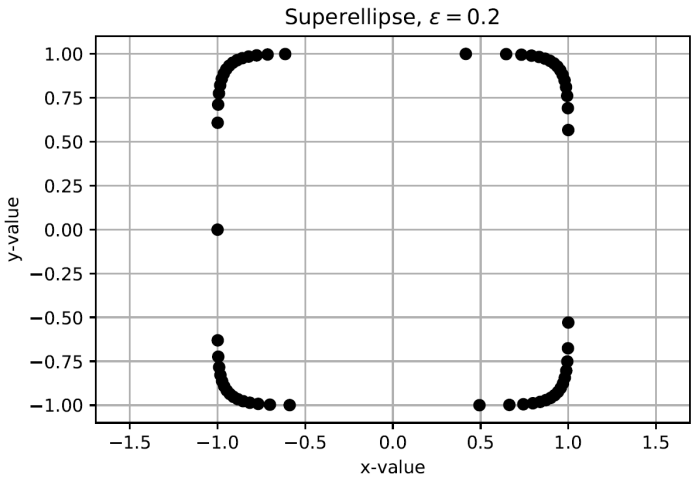
In the case of the box-shaped obstacle (shown in Figure 3.2a), due to the non-uniform spread of the points, the algorithm did not detect the collision. This problem can be clearly understood by going down to two dimensions and plotting superellipses (refer to Equation 2.38), as seen in Figure 3.3.

For larger ϵ values the points are spread quite uniformly, but for values below 1 the points are concentrated along the corners. Thus, for this method to effectively work with box-shaped obstacles specifically, the points cannot be uniformly sampled over the angle values. It can either be improved with selective sampling, or simply replaced with a grids of points oriented to resemble a box in accordance with the a_1, a_2 and a_3 values defining the lengths of the sides. Naturally, this is not an issue when implicitly representing box-shaped objects..

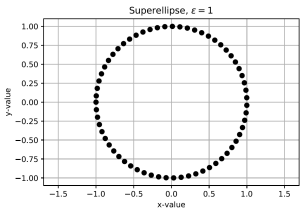
Note that while it would be possible to rotate a rhomb-shaped superellipse (Figure 3.3c) to get a square shape, simply setting $\epsilon_1 = \epsilon_2 = 2$ results in a double-pyramid shaped SQ and not a rotated cube.

Also, note that the reverse of this problem is encountered for very large ϵ values. Such superellipses resemble crosses (or increasingly pinched versions of Figure 3.3d), whose points become more and more concentrated in the center.

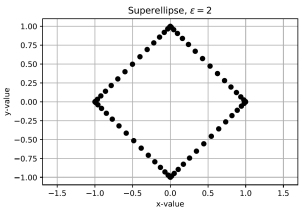
(a) Square-shaped superellipse. Note the conspicuous lack of points along the sides.



(b) Circle-shaped superellipse.



(c) Rhomb-shaped superellipse.



(d) Concave-shaped superellipse.

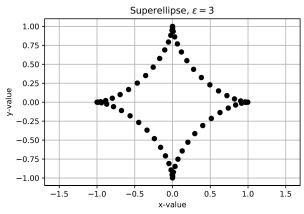


Figure 3.3: Four different superellipses sampled from $-\pi$ to π with the same increment value.

3.2 GJK

As first mentioned in Section 2.4.2, the implementation of GJK used in this thesis relies on a geometric approach that makes use of vector algebra. The n-simplex $\mathcal{W}$ only goes between five different stages:

- 0-simplex: The empty set, before the algorithm starts.
- 1-simplex: The set is a point in K_C.
- 2-simplex: The set is a line between two points.
- 3-simplex: The set is a triangle, as seen earlier in Figure 2.18.
- 4-simplex: The set is a pyramid, as can be seen in Figure 3.5. The algorithm tries to encapsulate the origin.

The algorithm does not go beyond that. If K_C surrounds the origin (i.e., K_A and K_B are in collision) a 4-simplex encapsulating it will eventually be able to be set up.

A 4-simplex is composed of four sides of 3-simplices. By keeping track of which order the points were added to the simplex, as well as using vector algebra (dot and cross products specifically), simple operations such as checking which point, edge, or face is facing towards the origin can be performed.

As an example, consider the 2-simplex case. Using the same notation as in Figure 2.18, the simplex is composed of $\mathcal{W} = \{B, A\}$, where B was added previously and A in the current iteration. The algorithm is trying to answer the following: is the origin closest to A, B, or the line AB?

AB Check this case first. If the origin is in the AB Voronoi region defined as all the points closest to the line AB (i.e. $AB \cdot (-A) > 0$), both A and B should be kept and the next search direction will be in the direction perpendicular to AB in the direction of the origin. This new direction can be calculated using a double cross-product: $AB \times (-A) \times AB$.

A Check this case after having checked the previous one. If the origin is closest to A, B can be discarded and a new point in the extreme direction of $-A$ can be added to the simplex. A new point will be produced and checked to see whether it is on A's side of the origin, or if it is actually beyond the origin. In the latter case the new point will be added to the simplex as the new A while the old A becomes the new B.

B This case is not checked because it is assumed to be impossible because of the way the algorithm will have produced A. A was produced in the direction of the origin, such that either a point on the line AB will be closer to the origin, or that point is A itself.

This geometric approach can be extended to 3-simplices, where $\mathcal{W} = \{C, B, A\}$. The same kind of optimization could also be applied, where there is no point in checking C, B or the edge BC because they should have been already checked as part of getting to the current A. Depending on whether the origin is closest to AC, AB or the new A region points are expelled and directions set.

For 4-simplices, each face, with the exception of the now previously visited DCB triangle, is checked individually to see if the point is either "below" the plane defined by the triangle (i.e. towards the inside of the pyramid 4-simplex) or anywhere else. As this is a binary response, eight different (2^3) scenarios are considered and measures are taken depending on each of them.

3.2.1 Support Mapping Function

A general support function is used, not one particularly optimized to a specific shape. As a function, it is applied once on K_A and once on K_B with the search direction reversed in the latter case, with the result for K_C obtained by subtracting them.

The following Python code takes a list of Numpy array *points* and a Numpy array *direction*, as well as a *reverse* boolean to reverse the direction when using this function for K_B.

It will loop through all the points in *points* and finds the one that is most extreme in the *direction* by taking their dot product and comparing the result with previous dot products. The most extreme will produce the biggest dot product. Numpy is a scientific computing package for Python ([Oliphant], [van der Walt et al.]).

```
def support_mapping_function(points, direction,
    reverse=False):
  if reverse==True:
    direction = -1*direction
  bestIndex = 0
  bestDotP = numpy.dot((points[0][bestIndex],
     points[1][bestIndex], points[2][bestIndex]),
     direction)
  for i in range(len(points)):
    dotProd = numpy.dot([points[0][i], points[1][i],
       points[2][i]], direction)
    if dotProd>bestDotP:
    bestIndex = i
    bestDotP = d
  furthestPoint = numpy.array([points[0][bestIndex],
     points[1][bestIndex], points[2][bestIndex]])
```

```
return furthestPoint
```

3.2.2 Termination Conditions

The algorithm will return either *True* (collision) or *False* (no collision), but there are four overall different ways for it to terminate into these. Note that unlike the general algorithm in 2.4.2, the one implemented for this thesis does not care about the specific distance between the object, just whether or not they intersect.

The following cases will result in the algorithm terminating:

- Termination Condition 1: If when generating a new point $\mathbf{w}_{i+1}$ in some direction **-v** the new point is not on the other side of the origin, it means that all points of K_C exist on the **+v** side of the origin and thus the origin is not surrounded by K_C. Thus, we can conclude that there is no collision. See Figure 3.4 for an example.
- Termination Condition 2: If the new support point $\mathbf{w}_{i+1}$ is arbitrarily close to the origin we can assume that a collision took place. In practice, the algorithm checks $abs(\mathbf{w}_{i+1}) < \mathbf{e}_{tol}$, where $\mathbf{e}_{tol}$ is set to something arbitrarily small like $\mathbf{e}_{tol} = (0.1, 0.1, 0.1)^T$.
- Termination Condition 3: If the algorithm can construct a 4-simplex (pyramid) that encapsulates the origin, the origin is obviously part of K_C and a collision happened.
- Termination Condition 4: If the algorithm is stuck in a loop of creating and dismissing various constellations of simplices, it can be assumed that a collision occurred. The variable i_{max} is used to define an upper limit on the allowed algorithm iteration loops.

The algorithm is sensitive to two things. The first is the explicit representation of the object: how many points where used to define K_A and K_B (i.e., the link and the obstacle)? More points is more accurate, but worse in terms of memory and the computation time for the support mapping function. The second factor it is sensitive to is numerical precision.

Termination Condition 4 might at first glance look like a very weak condition by itself, but it is acceptable because Termination Condition 1 is so strong that if the algorithm does get stuck in a loop it can be

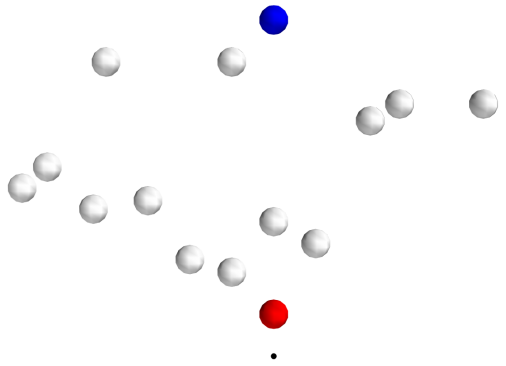


Figure 3.4: The set of points (in the x-y-plane) are all to one side of the origin (Black). Blue $(0, 0, 10)^T$ was arbitrarily chosen in the initial steps and Red $(0, 0, 1)^T$ is the point most extreme in direction $(0, 0, -10)^T$. The algorithm terminates according to Termination Condition 1. Note that the points are inflated for visual purposes only.

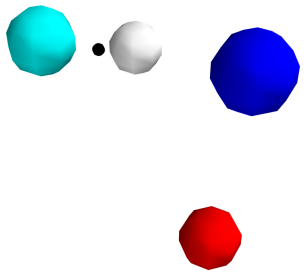


Figure 3.5: A 4 simplex (White-Blue-Cyan-Red) with the origin (Black). Note that the points have been inflated for visual purposes only. Also, note that the origin was purposefully rendered to be much smaller than the other four points, which are different in size due to the perspective.

assumed that it is due to the aforementioned factors. For example, in Figure 3.5 the origin is actually (if barely) inside the 4-simplex. But due to numerical imprecision, the algorithm misses this and does not register the Blue-Cyan-White triangle 3-simplex as being above it. Thus, the algorithm iterates i_{max} times until it is forcefully terminated.

It is worth pointing out that in the example that produced Figure 3.5, K_A and K_B were two spheres with radius 1 length units represented by 20^2 points each. K_B had been translated 2.01 length units in the x-axis. When translated 2.02 length units, the algorithm terminated in accordance with the Termination Condition 1.

3.3 RRT

The RRT algorithm is implemented as listed in Section 2.3.4. To generate distance, it uses the Euclidean distance between the angles in C-space.

When generating $\mathbf{p}_{random}$, at a specific count (e.g., every 100th sampled point) it will return the goal configuration instead of a randomly sampled point in order to help it converge faster and not have it blindly explore the whole C-space.

Though a goal configuration exists and is used for the biasing, in practice the aim is to have the tip of the robotic manipulator (typically, the EE) within a goal region. Thus, in this implementation, the algorithm terminates once the tip of the final link has entered a specific goal region. Whether or not the whole robotic manipulator is in the exact same angle configuration or pose as the goal configuration is not as relevant.

The two collision detection methods can be used interchangeably. The functions receive a specific arm configuration, check if it collides with the obstacles of the world and returns a boolean. When checking a line segment, if a collision is found the algorithm discards the point completely.

3.4 Simulation Design and Measurements

In accordance with **RQ1a** and what was stated in the Methodology section of Chapter 1, the SQ and GJK algorithms will be tested with each other. As they are used as black box methods in the RRT algorithm, it makes sense to test them separately with each other as the RRT should simply act as a multiplicative factor that is the same for both algorithm. Nevertheless, the full RRT+SQ and RRT+GJK algorithms will be tested for the sake of clarity.

Note that both algorithms will use objects generated by the SQ class implemented. The SQ class can generate convex shapes ($\epsilon_1 \leq 2$ and $\epsilon_2 \leq 2$) and store them explicitly, so it is useful for generating the links and obstacles that the GJK can take. For each SQ object, the points will be generated over η and ω, from $-\pi/2$ to $\pi/2$ and from $-\pi$ to π respectively. The amount of points generated in these intervals is determined by the parameter n, which sets the increment size. Thus, due to the spherical product, each SQ has n^2 number of points.

The following factors are deemed to have an effect on the collision detection computation time of the algorithms:

1. Number of links l.
2. Number of obstacles o.
3. SQ increment parameter n.
4. GJK's max iteration threshold i_{max}.
5. Convexity: GJK only works for convex shapes.

When testing how the algorithms perform relative each other, it is of interest to set up experiments that examine how the algorithms compare with each other when the aforementioned parameters (sans the convexity clause) are varied.

Chapter 4

Results

This chapter contains the results, using the implementations explained in the previous chapter. All of the experiments were performed on a computer with an i7-4900MQ Processor (2.8 GHz Quadcore). Note however that the code was not optimized to take advantage of the multiple cores, to make use of multithreading, or to use GPU hardware acceleration.

As there are, given an environment with l links and o obstacles, two ways of running the SQ collision detection method, both are used in the testing. In the following these are called OSQ and RSQ, which stand for Obverse-SQ and Reverse-SQ respectively. In OSQ the links are represented implicitly and the obstacles explicitly, whereas in RSQ obstacles are represented implicitly and the links explicitly.

4.1 Comparing Collision Detection Methods

In this section, the three collision detection methods are compared with each other in four rounds of tests. Each test round aims to challenge the algorithms in a distinct way.

4.1.1 Collision Detection: True Negatives

In the first round of tests, the aim was to see how fast the algorithms terminate when there are no collisions. The obstacle shapes were spherical ($a_1 = a_2 = a_3 = 5$, $\epsilon_1 = \epsilon_2 = 1$) and the links ($a_1 = a_2 = 1$, $a_3 = 10$,

$\epsilon_1 = 0.7$, $\epsilon_2 = 1$) prolate spheroids. Note that the links were all placed on a box-shaped SQ base, which added to the number of links and, the DoF, by one. Table 4.1 contains the results of this test round.

Table 4.1: Table showing the results of the first round of tests. The bold and underlined values are the quickest times.

Test:	n^2	l	o	GJK [s]	OSQ [s]	RSQ [s]
1.I	80^2	2	1	0.1420000	0.1599998	**0.1399999**
1.II	80^2	2	4	0.5679998	0.6619999	**0.5500000**
1.III	80^2	2	6	**0.7830000**	0.9519999	0.9399998
1.IV	80^2	2	12	1.3030000	1.4630001	**1.2670000**
1.V	80^2	4	1	0.5859999	0.5869999	**0.5300000**
1.VI	80^2	4	4	**1.1200001**	1.1340000	1.5039999
1.VII	80^2	4	6	**1.6449999**	2.0280001	1.8030000
1.X	80^2	4	12	2.6849999	2.955000	**2.648000**
1.IX	80^2	6	1	0.8770001	0.8309999	**0.8090000**
1.X	80^2	6	4	**1.8080001**	1.8940001	1.8220000
1.XI	80^2	6	6	**2.6629999**	2.5239999	2.9459999
1.XI	80^2	6	12	3.9990001	4.3389999	**3.9619999**

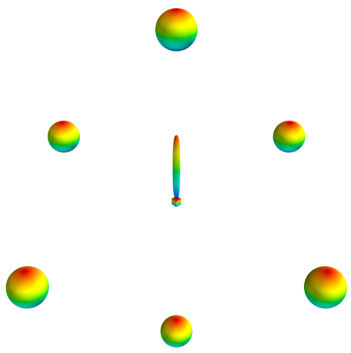


Figure 4.1: Visualization of Test 1.III.

i_{max} is presented here though it had no bearing on the terminations of the GJK algorithm instances; they all terminated in accordance with Termination Condition 1. Figure 4.1 visualizes Test 1.III (two links, six obstacles). Obstacles have been placed in pairs along the x, y and z axes around the arm, which was kept in that upright position. For the later tests with more links, the top spherical obstacle was translated sideways in order to avoid collision.

4.1.2 Collision Detection: True Positives

In the second round of tests, the algorithms were tested on how fast they detect a collision. Using 6 DoF arm (assuming that all the joints are revolute joints), various constellations of collision scenarios were tested. The links and obstacle were the same shapes as the test round in Section 4.1.1. One obstacle was used, which intersected with each of the links individually. Table 4.2 contains the result of this test round.

Table 4.2: Table showing the results of the second round of tests. The bold and underlined values are the quickest times.

Test:	n^2	Collision	i_{max}	GJK [s]	OSQ [s]	RSQ [s]
2.I	80^2	Link 0	50	0.1139998	0.3239999	**0.0007000**
2.II	80^2	Link 1	50	0.3740001	0.3050001	**0.0719998**
2.III	80^2	Link 2	50	0.4330001	0.2920001	**0.1240001**
2.IV	80^2	Link 3	50	0.3170002	0.4480000	**0.1790001**
2.V	80^2	Link 4	50	0.5440002	0.3099999	**0.2300000**
2.VI	80^2	Link 5	50	0.6230001	**0.2909999**	0.3079999

The GJK algorithm had an average time of 0.4008334 seconds, compared to the 0.3283333 seconds and the 0.1522833 seconds average times of the OSQ and RSQ algorithms respectively.

Figure 4.2 illustrates how the obstacle was moved along the z-axis. The GJK algorithm all terminated due to the algorithm being able to encapsulate the origin in a 4-simplex, i.e. Termination Condition 3. It was

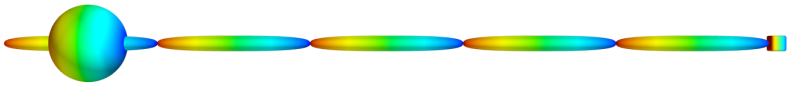


Figure 4.2: Visualization of Test 2.VI. The z-axis is horizontally positive to the left.

however discovered while testing that if the obstacle is translated by 25 units in the positive z-axis, as shown in Figure 4.3, the GJK algorithm will have problems and terminate in accordance with Termination Condition 4. As the maximum iteration limit i_{max} is set to 50, this resulted in a computation time of 1.275000 seconds.

4.1.3 Varying Number of Object Points

In the previous test rounds of Sections 4.1.1 and 4.1.2, n was kept at 80. This means that the points, of total number 80^2, were sampled with

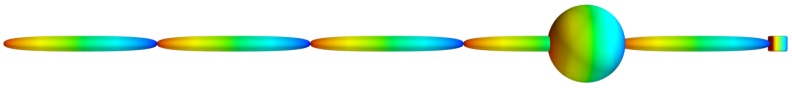


Figure 4.3: Visualization of a particularly troublesome case for the GJK algorithm. The obstacle has been translated 25 units along the positive z-axis.

increments of $\frac{pi}{80}$ radians (2.25 degrees) and $\frac{pi}{40}$ radians (4.5 degrees) for η and ω respectively. The aim of this test round was to see how the algorithms fare when the total number of points change. Or, in other words, when the point density changes.

In all the tests, there was one spherical obstacle as well as an arm with the base (link 0) and one more link. They did not intersect. The results are shown in Table 4.3.

Table 4.3: Table showing the results of the third round of tests. The bold and underlined values are the quickest times.

Test:	n^2	GJK [s]	OSQ [s]	RSQ [s]
3.I	10^2	0.0060000	**0.0030000**	0.0040000
3.II	20^2	0.0190001	0.0100000	**0.0079999**
3.III	50^2	0.0639999	0.0520000	**0.0480001**
3.IV	80^2	0.1580000	0.1170001	**0.1069999**
3.V	100^2	**0.2010000**	0.2180002	0.2230000
3.VI	150^2	0.4600000	0.5349998	**0.4470000**
3.VII	200^2	0.9840000	0.8140001	**0.7579999**
3.VIII	300^2	1.9730000	1.7790000	**1.7070000**
3.IX	500^2	**4.5090000**	4.8710001	4.6440001
3.X	1000^2	**19.0970001**	19.1299999	20.3789999

4.1.4 Large Number of Objects

The purpose of this test round was to examine how the algorithms were affected by truly large number of objects, both in terms of many links and in terms of many obstacles. $n^2 = 6400$ for all the values listed in Table 4.4.

4.2 RRT 6 Degrees of Freedom

The collision detection algorithms have been implemented as black box methods that can be used interchangeably within the RRT algorithm. A

Table 4.4: Table showing the results of the fourth round of tests. The bold values are the quickest times.

Test:	l	o	GJK [s]	OSQ [s]	RSQ [s]
4.I	1001	1	**58.779000**	64.878999	62.555999
4.II	2501	1	159.03400	165.02100	**158.38899**
4.III	5001	1	352.58500	339.92200	**309.05500**
4.IV	2	500	56.187000	57.055000	**55.895000**
4.V	2	1000	**124.67700**	131.08800	125.52399
4.VI	2	2500	296.88299	293.21700	**291.88700**

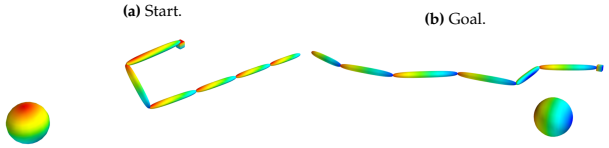
robotic arm with 6 DoF is used, with one obstacle preventing the arm from reaching the goal configuration.

The arm links are considered to be revolute joints that can rotate around the y-axis (θ Euler angle) only. Thus, in this case, the arm moves in the x-z-plane. The base (link 0) is fixed, such that a five link arm represents 4 DoF. $n^2 = 20^2$, the obstacle shape was spherical ($a_1 = a_2 = a_3 = 5$, $\epsilon_1 = \epsilon_2 = 1$), shifted 50 length units in the positive x-axis; and the links ($a_1 = a_2 = 1$, $a_3 = 10$, $\epsilon_1 = 0.7$, $\epsilon_2 = 1$) were prolate spheroids.

Table 4.5 contains the experiment data. Six tests were done for each algorithm.

Each test was started with the starting configuration in Figure 4.4a, with the provided goal configuration as seen in Figure 4.4b, though the algorithm itself would terminate if the EE (tip of the final link) was within a 5^3 cubic units box of the goal configuration's EE position. The configurations are also shown, from the point-of-view of the positive y-axis, in the example motion plan shown in Figure 4.5, along with intermediary steps.

Figure 4.4: Start and goal configurations for every test. Note that the point of view was rotated between the snapshots.



Note that the algorithm returned *a* motion plan in Figure 4.5, but not necessarily an optimal one by any measure.

Table 4.5: Result of the RRT testing using the three algorithms, with each having been run six times. The shortest test of each respective algorithm has been made bold and underlined.

RSQ						
Tests:	Test I	Test II	Test III	Test IV	Test V	Test VI
Total Time [s]:	4150	5715	6506	**780**	5588	8109
Total Loop Iterations:	6193	20909	12284	3165	21128	31426
Collisions:	1915	6874	4122	1062	5174	10670
Path Node Length:	8	7	6	7	6	9
GJK						
Total Time [s]:	21812	6520	708	2648	**148**	2486
Total Loop Iterations:	14954	21525	2611	9623	521	8811
Collisions:	3538	5244	771	3308	146	2073
Path Node Length:	7	6	6	7	5	6
OSQ						
Total Time [s]:	8583	8308	1928	10954	3702	**844**
Total Loop Iterations:	28449	25346	6326	38059	15222	3909
Collisions:	6245	6124	1672	10991	3758	1118
Path Node Length:	7	7	6	6	7	6

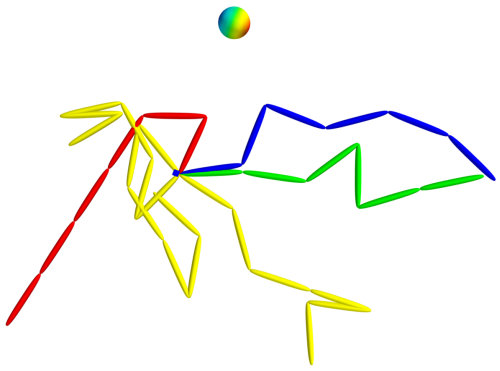


Figure 4.5: Example of a motion plan, with various configurations over time. Red is the start configuration, blue is the provided goal configuration and green is the accepted returned final configuration. The yellow configurations are intermediary steps.

Chapter 5

Discussion

This chapter concludes the thesis. Initially the research questions are addressed, before the results are evaluated in the context of the implementation. Finally, a section providing guidance for any future work is included.

5.1 Research Questions

The Research Questions initially introduced in Chapter 1 are discussed in the following subsections.

5.1.1 Handling Arbitrary Shapes

The biggest difference between the SQ and GJK when it comes to the ability to deal with arbitrary shapes of obstacles is related to that of convexity.

The GJK relies on the points being arranged in a convex shape. Minkowski summing convex shapes will result in another convex shape, and conversely any concavity will also be transferred onward into the resulting shape, within which the algorithm tries to find the origin.

Consider, in 2D, a case where the Minkowski difference returns a **C**-shaped figure with the origin located in between the tips but outside of the figure. In such a case it would be trivial to pick three points in the C and construct a triangle encapsulating the origin, but it would

not provide any information regarding whether the origin is actually on the C or not. Contrast this to a case with the **D**-shaped figure, with the origin in the center.

For SQ however, checking whether a point is in a non-convex is as simple as checking whether a point is in a convex SQ. Thus, there are whole families of shapes that SQ can represent better than what could be done with GJK.

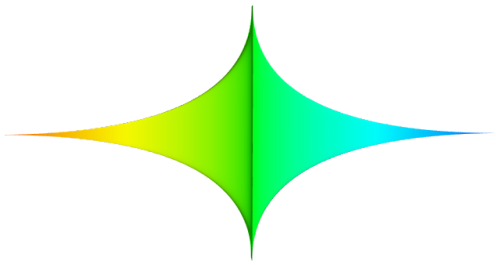


Figure 5.1: Non-convex: $a_1 = 0.5, a_2 = 5, a_3 = 10, \epsilon_1 = 4, \epsilon_2 = 0.2$.

Figure 5.1 shows one non-convex link that the GJK algorithm would not be able to properly handle as is. One would have to artificially inflate the shape and make the pinched sides straight to have it work with GJK.

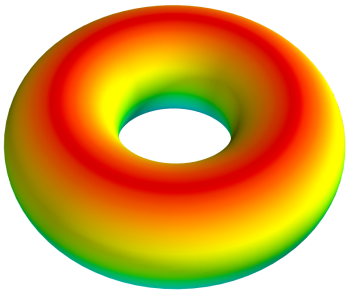


Figure 5.2: Supertoroid: $a_1 = 2, a_2 = 2, a_3 = 1, \epsilon_1 = 1, \epsilon_2 = 1$.

Figure 5.2 shows a supertoroid, a non-convex shape that the GJK algorithm would require be broken up into several convex primitives (e.g., overlapping spheres) to then do the check separately for each. A trade off would have to be made between amount of computations and accuracy. For fewer computations, bigger and fewer pieces would need to be used, sacrificing accuracy. More, smaller pieces could better follow the true shape of the supertoroid but would obviously require more computation.

While it is outside the scope of this work, it is worth restating the fact that SQ can be further manipulated and deformed using the taper, bend and twist operations mentioned in Section 2.4.3.

Finally, in defense of the GJK algorithm, the creation of (convexly clustered) points for it to use will always be much more straightforward. To create a SQ it has to either be manually created by tuning the dimension and shape parameters, or by solving an optimization problem to fit it to the points.

5.1.2 Modeling a 6 DoF Arm

The ability to be able to model arbitrary shapes and being able to conveniently do collision detection on them is useful. The discourse in Section 5.1.1 regarding the arbitrary shapes of obstacles applies here as well. The supertoroid, for example, is very useful for modeling a ring-shaped EE part of a robotic manipulator. The bending operation, though not part of this work, allows for creating links that are inherently curved with a non-convexity.

In practice, to increase fitting accuracy multiple objects (SQ or general polytopes) have to be used, combining primitives into a a more complex whole. SQ make for very customizable primitives, which in return means even better fitting modeling. GJK and polytopes will always be simpler to set up, though they might require more sets of polytope clouds to compensate for the same non-convex overall structure.

5.1.3 Qualitative Analysis of A*

The qualitative analysis of the A* algorithm is presented here.

Memory Issues

A*, when used in motion planning, means finding a path in configuration space. Assuming that the space is divided into hypercube units with orthogonal sides, a 2D hypercube (a square) has four sides into which the algorithm can expand; and a 3D hypercube (a cube) has six faces. Extrapolating, any n-dimensional hypercube will have $2n$ corresponding hypercubes the algorithm can expand into.

Thus, as the number of DoF increase, the memory space required to hold the graph nodes also increases dramatically. This can result in significant problems, if not a total collapse of the algorithm and the computing unit it is running on. The RRT algorithm does not suffer from this problem with higher DoF.

One way to remedy this would be to reduce the resolution parameter such that each hypercube occupies a bigger volume, reducing the total number of hypercubes. The advantage of a robotic manipulator having many DoF is precisely that the redundancy of the joints increases the probability that a feasible path will be found. Thus, it is conceivable that A* could be successfully used for motion planning high DoF manipulators, with the resolution parameter set low to compensate. Of course, then setting the resolution parameter (and parameter tuning in general) becomes another issue for the engineer to deal with. Regarding this specific topic, more research is required to fully be able to answer.

Parameter Tuning

A* is resolution complete and relies on having properly tuned parameters to even be able to find a solution (if one exists). This can be difficult to know beforehand. Even then, as previously mentioned, an increased resolution will increase the memory requirements and will also lead to an increased computation time.

RRT is probabilistically complete, and will return a solution (provided that it exists) given enough time. However, it is not bereft of the need to manually tune some parameters. The difference is that it will not affect its ability to return a solution, just its ability to do it in a reasonable time frame. The parameter in question is the maximum distance d, which limits how far away from the tree a new node in configuration space can be located. A very small d will result in many nodes having to be generated to explore the same amount of space. But a very large d will result in a very long line segment that needs to be checked for collisions.

In the RRT implementation of this thesis, it is possible to set a maximum distance (defined as Euclidean distance in configuration space between radian angle values) as well as a second parameter: interval number. The default is ten, such that it will collision check the old node, the new node and the eight angle configurations between them.

If the combination of these two parameters in the implementation is not thorough enough, the algorithm could be at risk of approving a line segment between two angle configurations that in reality has a collision.

A* does come with a small risk of producing false negative collision checks. The biggest danger lies in specific critical regions, like the boundary between free space and the forbidden region. One simple way to mitigate this risk is to insert a safety margin around the forbidden region of at least one hypercube, as false positives are slightly more preferable over false negatives, even at the risk of not being able to return valid paths.

Forbidden Regions

One of the stated benefits of using RRT is that it allows for the forbidden configuration space to be calculated *ad hoc*, when it is necessary, and not *a priori* for the whole space.

For A*, the algorithm itself is generally agnostic to this, provided that a heuristic function is not used that requires the a priori explicit computation of the forbidden regions. For example a heuristics measure of "distance to closest forbidden region" would necessitate this kind of calculation, though it is not clear if such a heuristic would be considered admissible for use in A* in the first place.

5.1.4 Evaluation of the Computational Memory Management

From a memory standpoint, the GJK algorithm needs explicit representations of all objects, both links and objects. The SQ algorithms on the other hand only requires an explicit representation of one of them; for the one that is implicitly represented, only the five a_* and ϵ_*, as well as six translation and rotation parameters need to be stored. These are a negligible number compared to n^2. As such, GJK keeps $O(n^2 \times (l + o))$ points in memory, and SQ keeps either $O(n^2 \times o)$ or $O(n^2 \times l)$ in memory.

5.1.5 Evaluation of the Computational Processing Time

Regarding the Implementation

Since the collision detection methods were implemented as black box methods, the RRT acted as a multiplicative factor on each algorithm's respective computation time. Considering that creating motion plans for a six DoF arm required hours of processing time, one of the first conclusions to make is that the implementation was slow. This can most likely be attributed to the fact that it was written in Python, a dynamically typed, interpreted high-level language. Typically, if this had been used in a production environment like in a simulation engine or on a robot the algorithms would have been written in a compiled language like C or C++. However, as these issues are shared by both algorithm implementations they do not affect the difference in relative performance between them.

Comparison

When there were no collisions (i.e. the tests from Sections 4.1.1, 4.1.3, 4.1.4), there was surprisingly not much of a difference in performance between the algorithms, even when dramatically varying the number of links, obstacles or even points n^2. The Reverse-SQ generally outdid the Obverse-SQ, perhaps due to small fixed costs associated with jumping between loops in the implementation.

This lack of difference in performance can most likely be attributed to the support mapping function of the GJK, which is also composed of a for loop looping through the points to find the most extreme point. It runs this function twice (for K_A and K_B, in lieu of taking the Minkowski sum and then performing it on K_C). If there is no collision, the algorithm will in the second iteration fail to find a point on the other side of the origin, terminating in accordance with Termination Condition 1 and returning a false. Thus, it has had to sweep through $2 \times n^2$ points twice to conclude a non-collision. It is also possible that the algorithm managed to set up a 2-simplex (line), with the edges on each side of the origin, but then be unable to find a third point on the other side of the origin, resulting in the algorithm having swept through $2 \times n^2$ three times. Finally, the worst case is if it draws up a

3-simplex (triangle) but terminates when it fails to find a fourth point. Since the initial search vector is randomly initialized, the algorithm can produce different execution times for the same collision detection scenario.

The SQ algorithms, however, sweep through n^2 points of one object and calculate the $F(.)$ inside-outside implicit function of the other object for each point. This calculation is only done once for each point, and as the n^2 points are only looped through once for each check, it is advantageous in comparison to the support mapping function even if the individual calculations might take a longer time themselves. Furthermore, when there is a collision, unless the GJK algorithm specifically returns the origin itself (Termination Condition 2), the GJK algorithm is doomed to either do four iterations of its algorithm (Termination Condition 3), or to get stuck in a non-stopping loop, hit i_{max} and trigger Termination Condition 4. All of these will require more point sweeps, which increase the computational work.

This would explain the results of the test rounds in Section 4.1.2, where the GJK algorithm was outclassed by the SQ algorithms, going by the average times listed in Section 4.1.2.

It is also evident that the GJK is a more sensitive algorithm. In the case where it terminated in accordance with Termination Condition 4 in Section 4.1.2, it took 1.275 seconds, roughly five times as long as it should have taken. This is quite an extraordinary long time and can be attributed to the i_{max} being too high and most likely needing to be lowered. This was not experimented with, as it was considered too specific to the GJK algorithm and only came into effect in rare cases.

In some specific cases where collision has taken place, the SQ algorithm(s) have the ability to immediately return a collision. For example, in Test 2.I the RSQ returns a collision answer incredibly fast at 0.0007 seconds. This is because the algorithms loops through the arm links and check each with the obstacle, beginning with the base (Link 0). As the RSQ checks whether each point is within the obstacle or not, and the points are very tightly concentrated, the algorithm almost immediately returns a collision. The OSQ algorithm must however loop through the points of the sphere until it reaches the "latitude" at which the sphere intersects with the base.

Note that there is a potential to miss a collision here, if one SQ object is much smaller than the other SQ and if it has fully entered the other, the algorithm which explicitly represents the bigger object and implicitly the smaller object will miss a collision as only the surface is explicitly represented. The mirror SQ collision detection algorithm will of course be able to detect this collision. In practice, this is very unlikely to happen as the RRT algorithm checks collisions between angle configurations by incrementing the links forward. It is similar to the case when n is too small and point density is lost, as mentioned in Section 3.1.1.

Regarding the RRT, the inherent random nature of the vanilla RRT algorithm makes it impossible to judge if a specific combination of RRT and collision detection algorithms is superior. The GJK generally had lower execution times than both RSQ and OSQ, producing the absolute fastest execution time at 148 seconds in RRT Test V. This does however not mean much, as it cannot be attributed to some conceived superior quality of GJK, as opposed to random chance. The fact that, for the same environment, the test results could vary so wildly lends credence to this.

As was mentioned previously, the GJK algorithm achieves parity with the SQ-based algorithm(s) when there are no collisions (i.e., when returning a true negative). When there actually were collisions, the algorithm was outshone. Hence, in more obstacle-rich environments there might be a starker difference in the RRT algorithm's performance.

When the RRT algorithm evaluates a newly produced node (one in the direction of the randomly sampled node from the closest already saved node), it checks the line segment between them. It checks it in increments of tenths, including the existing node which has already been checked and the potential new node that is being evaluated. As the collision can have happened over those nine increments it is thus also not possible to say that there is direct relationship between the number of collisions and the total time.

Thus, it is better to examine the results of Sections 4.1.1 to 4.1.4 and view the RRT as an overall multiplicative factor on top.

5.2 Future Work

Potential avenues for future work and research are provided in the following subsections.

5.2.1 Regarding the Implementation

Both the GJK and SQ algorithms could have been improved by refactoring the code and making use of concurrency, as opposed to running everything on one thread. The algorithms check each link with each obstacle. It would be perfectly possible to do these checks simultaneously concurrently, and possibly in parallel over multiple cores.

In some cases a lot of unnecessary checks were done. For example, there is no reason to check collisions with obstacles farther away from the base than the maximum extent of the sum of the links. One could also surround each object in a sphere that is big enough to maximally surround each object, and then check collisions between each spherical representation as a pre-check before doing the actual check. One could expand on it further by drawing inspiration from Bounding Box Methods (mentioned in Section 2.4.1), and organize objects and their spherical representations in tree structures.

Implementing these changes could increase the raw performance of the algorithms by orders of magnitude. This has the potential to bring the calculations for the RRT algorithm from thousands of seconds (hours) to tens of seconds or even individual seconds, making it appropriate for online use.

5.2.2 Use in Visual Servoing

As was mentioned in Chapter 1, SQ have seen use in the field of visual servoing by solving the SQ parameters for point clouds captured with 3D sensors. Further research could be done looking into the potential to integrate the RRT+SQ algorithm with real similar cases, where motion plans can be drawn up on-line for high DoF manipulators.

In such a future work, it would behoove the user to also look into variants of the RRT (such as those listed in Section 2.3.4) that are able to faster find better paths in terms of some measure of utility. It can also be useful to do some post-processing to produce smoother paths.

While researching MPC, it was found that one method of doing obstacle avoidance with MPC is to insert an APF into the cost function. Furthermore, it was also found that a method for creating SQ-based APFs was developed back in the 1980s. Combining these two could provide a new method of MPC-based motion planning, making use of the aforementioned 3D sensors to collect point clouds and construct potential functions from them.

5.3 Conclusion

In this thesis, a collision detection algorithm based on the concept of SQ used with the RRT algorithm to perform motion planning was presented. It was compared with the GJK collision detection method which is sometimes used with the RRT algorithm and was shown to perform at a similar level or better when there were no collisions. But when there were collisions the SQ-based algorithms were able to outclass the GJK algorithm in terms of speed of execution.

Its ability to handle non-convex obstacle shapes makes it stand out from GJK, which is limited to only being able to handle convex objects. This superiority extends to its ability to most accurately fit 6 DoF arm shapes, though as the GJK relies on convex sets of points in space it will be easier to set up.

Furthermore, from a memory standpoint, GJK was inferior as it requires points of both obstacles and robot manipulator links to be explicitly stored, whereas the SQ-based algorithms only require one to be explicitly saved with the other object implicitly modeled using the SQ parameters.

However, care needs to be taken when using SQ-based collision detection. There needs to be an certainty that the explicit representation of objects are of high enough precision for the purposes at hand. If the point density is not high enough, or if the points are not spread uniformly enough, there is a risk of missing true collisions.

Finally, it was shown that SQ-based algorithms can be used with the RRT algorithm. The RRT algorithm acted as a multiplicative factor that only amplified the inherent benefits or deficiencies of the internal collision detection method.

Bibliography

Baris Akgun and Mike Stilman. Sampling Heuristics for Optimal Motion Planning in High Dimensions. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011. URL http://www.leonardjaillet.com/Publications{_}files/Iros08{_}Jaillet{_}TransitRRT.pdf.

E. Appleton and D. J. Williams. *Industrial Robot Applications*. Springer, Dordrecht, 1987. ISBN 978-94-009-3125-1. doi: 10.1007/978-94-009-3125-1.

Algernon Austin, Cherrie Bucknor, Kevin Cashman, and Maya Rockeymoore. Stick Shift: Autonomous Vehicles, Driving Jobs, and the Future of Work. Technical report, Center for Global Policy Solutions, 2017.

Alan H. Barr. Global and local deformations of solid primitives. *ACM SIGGRAPH Computer Graphics*, 18(3):21–30, 1984. ISSN 00978930. doi: 10.1145/964965.808573.

Joshua Bialkowski, Sertac Karaman, Michael Otte, and Emilio Frazzoli. Efficient collision checking in sampling-based motion planning. *Springer Tracts in Advanced Robotics*, 86:365–380, 2013. ISSN 1610742X. doi: 10.1007/978-3-642-36279-8_22.

Georg Biegelbauer and Markus Vincze. Efficient 3D object detection by fitting superquadrics to range image data for robot's object manipulation. *Proceedings - IEEE International Conference on Robotics and Automation*, (April):1086–1091, 2007. ISSN 10504729. doi: 10.1109/ROBOT.2007.363129.

Lieboud Van Den Broeck, Moritz Diehl, and Jan Swevers. Time optimal MPC for mechatronic systems. *IEEE Conference on Decision and Control (CDC)*, page 3000, 2009.

Stephen Cameron. Enhancing GJK: computing minimum and penetration distances between convex polyhedra. (April):3112–3117, 2002. doi: 10.1109/robot.1997.606761.

Peng Cheng, Zuojun Shen, and Steven M. LaValle. RRT-based Trajectory Design for Autonomous Automibles and Spacecraft. *Archives of Control Sciences*, 2001.

Woojin Chung. Introduction. In *Nonholonomic Manipulators*, chapter 1, pages 1–15. Springer, Berlin, Heidelberg, 2004. ISBN 978-3-540-22108-1. doi: https://doi-org.focus.lib.kth.se/10.1007/978-3-540-44425-1_1.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. RIvest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3 edition, 2009. ISBN 9780262270830.

Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1994. ISSN 00045411. doi: 10.1145/174147.174150.

Kester Duncan, Sudeep Sarkar, Redwan Alqasemi, and Rajiv Dubey. Multi-scale superquadric fitting for efficient shape and pose recovery of unknown objects. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4238–4243, 2013. ISSN 10504729. doi: 10.1109/ICRA.2013.6631176.

Christer Ericson. The Gilbert-Johnson-Keerthi (GJK) Algorithm, 2004.

F. Fahimi. Non-linear model predictive formation control for groups of autonomous surface vessels. *International Journal of Control*, 80(8): 1248–1259, 2007. ISSN 00207179. doi: 10.1080/00207170701280911.

Claudio Fantacci, Giulia Vezzani, Ugo Pattacini, Vadim Tikhanoff, and Lorenzo Natale. Markerless visual servoing on unknown objects for humanoid robot platforms. 2017. doi: 10.1109/ICRA.2018.8462914. URL http://arxiv.org/abs/1710.04465.

Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via

direct sampling of an admissible ellipsoidal heuristic. *IEEE International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014. ISSN 21530866. doi: 10.1109/IROS.2014.6942976.

E Gilbert and D Johnson. A fast procedure for computing the distance between complex objects in three space. *Robotics and Automation.*, 4(2), 1987. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1087825.

Elmer G. Gilbert and Chek-Peng Foo. Computing the Distance Between General Convex Objects in Three-Dimensional Space. *IEEE Transactions on Robotics and Automation*, 6(1):53–61, 1990.

Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space, 1988.

S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. 1996. ISSN 00084166. doi: http://doi.acm.org/10.1145/237170.237244.

El-Hadi Guechi, Samir Bouzoualegh, Youcef Zennir, and Saso Blazi. MPC Control and LQ Optimal Control of A Two-Link Robot Arm : A Comparative Study †. pages 1–14, 2018. doi: 10.3390/machines6030037.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths, 1968.

Herman Johannes Haverkort. *Results on geometric networks and data structures*. PhD thesis, Universiteit Utrecht, 2004. URL http://dspace.library.uu.nl/handle/1874/892.

Alan H.Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

Y Hirano, K Kitahama, and S Yoshizawa. Image-Based Object Recognition and Dextrous Hand/Arm Motion Planning Using {RRT}s for Grasping in Cluttered Scene. *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2–7, 2005.

Michael Hoy, Alexey S. Matveev, Matt Garratt, and Andrey V. Savkin. Collision-free navigation of an autonomous unmanned helicopter in unknown urban environments: Sliding mode and MPC ap-

proaches. *Robotica*, 30(4):537–550, 2012. ISSN 02635747. doi: 10.1017/S0263574711000816.

Léonard Jaillet, Juan Cortés, and Thierry Siméon. Transition-based RRT for Path Planning in Continuous Cost Spaces. Technical Report Section II. URL http://www.leonardjaillet.com/Publications{_}files/Iros08{_}Jaillet{_}TransitRRT.pdf.

Aleš Jaklič, Aleš Leonardis, and Franc Solina. *Segmentation and Recovery of Superquadrics*, volume 20. Springer Science+Business Media Dordrecht, 2000. ISBN 978-90-481-5574-3. doi: 10.1007/978-94-015-9456-1. URL http://link.springer.com/10.1007/978-94-015-9456-1.

Sertac Karaman and Emilio Frazzoli. Robotics: Science and Systems Incremental Sampling-based Algorithms for Optimal Motion Planning. In *Robotics: Science and Systems*, 2010. URL http://ares.lids.mit.edu/software.

Sertac Karaman and Emilio Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *The International Journal of Robotics Research*, pages 846–849, 2011. doi: 10.1177/0278364911406761.

Petros Karamanakos, Tobias Geyer, Senior Member, Ralph Kennel, and Senior Member. A Computationally Efficient Model Predictive Control Strategy for Linear Systems With Integer Inputs. (1), 2016.

Katsoulas. Reliable recovery of piled box-like objects via parabolically deformable superquadrics, 2003. URL http://ieeexplore.ieee.org/document/1238448/.

Lydia E Kavraki, Mihail N. Kolountzakis, and Jean-Claude Latombe. Analysis Of Probabilistic Roadmaps For Path Planning - Robotics and Automation. *IEEE Transactions on Robotics and automation*, 14(1): 9–12, 1998.

Oussama Khatib. *Commande dynamique dans l'espace opérationnel des robots manipulateurs en présence d'obstacles.* PhD thesis, Toulouse, France, 1980.

Oussama Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The Intenrational Journal of Robotics Research*, 5(1), 1986.

Gordon Kindlmann. Superquadric Tensor Glyphs. *Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 1–8, 2004. doi: 10.2312/VisSym/VisSym04/147-154.

Rachel Kirby, Reid Simmons, and Jodi Forlizzi. Variable sized grid cells for rapid replanning in dynamic environments. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 4913–4918, 2009. doi: 10.1109/IROS.2009.5354352.

Stefan Klanke, Dmitry Lebedev, Robert Haschke, Jochen Steil, and Helge Ritter. Dynamic path planning for a 7-DOF robot arm. *IEEE International Conference on Intelligent Robots and Systems*, (November): 3879–3884, 2006. ISSN 0014-4819. doi: 10.1109/IROS.2006.281798.

J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. *Proceedings - IEEE International Conference on Robotics and Automation*, (Icra):995–1001, 2002. ISSN 1050-4729. doi: 10.1109/robot.2000.844730.

Samir Lahouar, Zeghloul Said, Lotfi Romdhane, Collision Free, and Path Planning. *Collision Free Path Planning for Multi-DoF Manipulators*. Number December. 2006. ISBN 3866112858. doi: 10.1177/0146167204271659.

Steven M. LaValle. *Planning algorithms*, volume 9780521862. 2006. ISBN 9780511546877. doi: 10.1017/CBO9780511546877. URL http://ebooks.cambridge.org/ref/id/CBO9780511546877.

Steven M. (Iowa State University) LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998.

Seung-jae Lee, Seung-hwan Baek, and Jong-hwan Kim. Arm Trajectory Generation based on RRT * for Humanoid Robot. *Advances in Intelligent Systems and Computing book series (*, 45, 2015.

Patrick Lindemann. The Gilbert-Johnson-Keerthi Distance Algorithm. *Algorithms in Media Informatics*, 2009. URL http://www.medien.ifi.lmu.de/lehre/ss10/ps/Ausarbeitung{_}Beispiel.pdf.

Stephen R. Lindemann and Steven M. LaValle. Current Issues in Sampling-Based Motion Planning. pages 36–54, 2005. ISSN 16107438. doi: 10.1007/11008941_5. URL http://link.springer.com/10.1007/11008941{_}5.

Citable Link, Yoshiaki Kuwata, Justin Teo, Student Member, Gaston Fiore, Student Member, Jonathan P How, and Senior Member. Real-Time Motion Planning With Applications to Autonomous Urban Driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

Tomas Lozano-Perez. Spatial Planning : A Configuration Space Approach. *IEEE Transactions on Computers*, c-32(2), 1983.

James Manyika, Susan Lund, Michael Chui, Jacques Bughin, Jonathan Woetzel, Parul Batra, Ryan Ko, and Saurabh Sanghvi. Jobs lost, jobs gained: What the future of work will mean for jobs, skills, and wages. Technical report, McKinsey Global Institute, 2017.

Julien Marzat, Sylvain Bertrand, Alexandre Eudes, Martial Sanfourche, and Julien Moras. Reactive MPC for Autonomous MAV Navigation in Indoor Cluttered Environments: Flight Experiments. *IFAC-PapersOnLine*, 50(1):15996–16002, 2017. ISSN 24058963. doi: 10.1016/j.ifacol.2017.08.1910.

Travis E. Oliphant. *A guide to NumPy*. Trelgol Publishing, 2006.

Chanhun Park. Self-Collision Detection & Avoidance Algorithm for a Robot Manipulator. *Internation Journal of Engineering and Innovation Technology*, 5(4):139–142, 2015. ISSN 2226809X.

J. M. Park, D. W. Kim, Y. S. Yoon, H. J. Kim, and K. S. Yi. Obstacle avoidance of autonomous vehicles based on model predictive control. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 223(12):1499–1516, 2009. ISSN 09544070. doi: 10.1243/09544070JAUTO1149.

Ricardo Pascoal, Vitor Santos, Cristiano Premebida, and Urbano Nunes. Simultaneous segmentation and superquadrics fitting in laser-range data. *IEEE Transactions on Vehicular Technology*, 64(2):441–452, 2015. ISSN 00189545. doi: 10.1109/TVT.2014.2321899.

J. B. Rawlings and D. Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, Madison, Wisconsin, 2 edition, 2012. ISBN 978-0-9759377-3-0.

M Razi and M Haeri. Design of a robust model predictive controller with reduced computational complexity. *ISA Transactions*, 53(6):1754–

1759, 2014. ISSN 0019-0578. doi: 10.1016/j.isatra.2014.09.008. URL http://dx.doi.org/10.1016/j.isatra.2014.09.008.

John H. Reif. Complexity of the Generalized Mover's Problem. Technical report, Harvard University, Cambridige, MA, 1985.

Stefan Richter and Manfred Morari. High-Speed Online MPC Based on a Fast Gradient Method Applied to Power Converter Control. *American Control Conference*, pages 4737–4743, 2010.

Stefan Richter, Colin Neil Jones, and Manfred Morari. Computational Complexity Certification for Real-Time MPC With Input Constraints Based on the Fast Gradient Method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, 2012. doi: 10.1109/TAC.2011.2176389.

W Rossing, P. H. Hogewerf, A.H. IPEMA, C.C. Ketelaar-De Lauwere, and C.J.A.M. De Koning. Robotic Milking in Diary Farming. *Netherlands JNournal of Agricutlral Science*, 45:15–31, 1997.

Oliver Rovný and Kvetoslav Belda. Predictive Control of 5 DOF Robot Arm of Autonomous Mobile Robotic System. pages 339–344, 2017. doi: 10.1109/PC.2017.7976237.

Amol Sasane and Krister Svanberg. *Optimization*. Department of Mathematics, Royal Institute of Technology, 2016.

Amir Shahzad and Eric C Kerrigan. A Warm-start Interior-point Method for Predictive Control. 2008.

Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*, volume 46. Springer, 2nd edition, 2016. ISBN 978-3-319-32550-7. doi: 10.1007/978-3-319-32552-1.

Eliana Costa E. Silva, M. Fernanda Costa, Wolfram Erlhagen, and Estela Bicho. Superquadrics objects representation for robot manipulation. *AIP Conference Proceedings*, 1738(1), 2016. ISSN 15517616. doi: 10.1063/1.4952096.

Wesley E. Snyder and Hairong Qi. *Machine Vision*. Cambridge University Press, 2010. ISBN 9780511988578.

Mark W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. Wiley India Pvt. Limited, 2nd edition, 2004. ISBN 9788126517800.

Zaid Tahir, Ahmed H. Qureshi, Yasar Ayaz, and Raheel Nawaz. Potentially guided bidirectionalized RRT* for fast optimal path planning in cluttered environments. *Robotics and Autonomous Systems*, 108:13–27, 2018. ISSN 09218890. doi: 10.1016/j.robot.2018.06.013.

Jonathan S Terry, Levi Rupert, and Marc D Killpack. Comparison of Linearized Dynamic Robot Manipulator Models for Model Predictive Control. 2017.

Dimitrios Tzovaras and Michael Gerassimos Strintzis. SQ-map: Efficient layered collision detection and haptic rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):80–93, 2007. ISSN 10772626. doi: 10.1109/TVCG.2007.20.

C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (October):1178–1183, 2004. doi: 10.1109/iros.2003.1248805.

Gino Van Den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. pages 1–14, 1998.

Gino Van Den Bergen. A Fast and Robust GJK Implementation for Collision Detection of Convex Objects. *Journal of Graphics Tools*, 4(2): 7–25, 1999. ISSN 1086-7651. doi: 10.1080/10867651.1999.10487502.

Stéfan van der Walt, Chris S. Colbert, and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(22):22–30, 2011.

Giulia Vezzani, Ugo Pattacini, Giulia Pasquale, and Lorenzo Natale. Improving Superquadric Modeling and Grasping with Prior on Object Shapes. pages 6875–6882, 2018. ISSN 1549-7747. doi: 10.1109/ICRA.2017.7989187.

R Volpe, P Khosla IEEE Transactions on Systems, Undefined Man, Undefined And, and Undefined 1990. Manipulator control with superquadric artificial potential functions: Theory and experiments. *Ieeexplore.Ieee.Org*, 1990. URL https://ieeexplore.ieee.org/abstract/document/61211/.

Liuping Wang. *Model Predictive Control System Design and Implementation Using MATLAB*. Springer, 2009. ISBN 978-1-84882-330-3 978-1-84882-331-0. doi: 10.1007/978-1-84882-331-0.

Charles W. Warren. Global Path Planning Using Artificial Potential FIelds, 1989.

Dustin J. Webb and Jur van den Berg. Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints. Technical report, 2012. URL http://arxiv.org/abs/1205.5088.

Richard L. Wilson. Ethical issues with use of Drone aircraft. Technical report, 2014.

Mark A. Yerry and Mark S. Shephard. Automatic mesh generation for three-dimensional solids. *Computers and Structures*, 20(1-3):31–39, 1985. ISSN 00457949. doi: 10.1016/0045-7949(85)90050-1.

Yongsoon Yoon, Tokson Choe, Yongwoon Park, and H. Jin Kim. *Obstacle avoidance for wheeled robots in unknown environments using model predictive control*, volume 17. IFAC, 2008. ISBN 9783902661005. doi: 10.3182/20080706-5-KR-1001.1873. URL http://dx.doi.org/10.3182/20080706-5-KR-1001.01151.

S Zeghloul, B Blanchard, and M Ayrault. SMAR : A Robot Modeling and Simulation System. *Robotica*, 15(1997):63–73, 1997.

TRITA TRITA-ITM-EX 2019:394