Institut für Automatisierungstechnik

Artificial Intelligence

Master Thesis

Robot trajectory optimization for collision-free fetch-and-place applications using sequential quadratic programming

Mahesh Kumar Karikalan (Matr.Nr: 3020323)

May 24, 2018

Tutor:
Dipl.-Ing. Georg Bartels

1st Examiner:
Prof. Dr.-Ing. Kai Michels

2nd Examiner:
Prof. Michael Beetz, PhD

Diese Erklärungen sind in jedes Exemplar der Abschlussarbeit mit einzubinden.
This declaration must be included into the Master's Thesis.

Name: ______________________ Matrikel-Nr: ________________

Urheberrechtliche Erklärung

Erklärung gem. § 10 (10) Allgemeiner Teil der MPO vom 27.10.2010

Hiermit versichere ich, dass ich meine Masterarbeit ohne fremde Hilfe angefertigt habe, und dass ich keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, habe ich unter Angabe der Quellen als solche kenntlich gemacht.

Die Masterarbeit darf nach Abgabe nicht mehr verändert werden.

Datum: ______________ Unterschrift: ________________________

Erklärung zur Veröffentlichung von Abschlussarbeiten

Bitte auswählen und ankreuzen:

- ☐ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
- ☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
- ☐ Ich bin *nicht* damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Datum: ______________ Unterschrift: ________________________

Abstract

The current technological advancements made robots to be intelligent and cost-effective to use them in unstructured environments like supermarkets. In supermarkets, lack of human labour availability is a major problem, where the robots can be of help to monitor and replenish the sold products on the supermarket shelves.

Supermarkets are challenging environments, because of the constant change of products, Although products undergo constant change, the supermarket shelf remains constant making the fetch and place of these products on the shelf a monotonous task. Hence, an idea would be to reuse robots' previously gained knowledge from past experiences to plan a trajectory for fetching and placing the supermarket products.

In order to reuse previous knowledge, an appropriate software algorithm has to be chosen. Sequential Quadratic Programming (SQP) is one such mathematical optimization technique that can be initialized with an initial guess to obtain an optimized result. With SQP, the trajectory planning was proposed as a mathematical optimization problem by Schulman et al. in [1]. In this thesis, the proposed algorithm is re-implemented to adapt the obstacle collision constraints to convert the constrained problem into an unconstrained problem. Also, the algorithm is adapted by formulating time continuous collision cost for the robot's self-collisions instead of discrete collision costs.

The newly adapted algorithm is evaluated for reliability, run-time performance, consistency, sensitivity for different solver parameters. The adapted algorithm is also compared with the re-implemented original algorithm to evaluate the run-time performance and the number of problems solved for the randomly generated problems and obstacle constraints.

The evaluated results showed that the adapted trajectory planning algorithm can solve a good ratio of previously unsolvable problems by an original re-implemented algorithm with an extra computational cost.

Table of Contents

List of Figures

List of Tables

Acronyms

CES Convex Elastic Smoothing. 18

CSO Configuration Space Obstacle. 33, 34, 36, 39

CVXOPT A Python package for convex optimization. 48

CVXPY Python-embedded modeling language for convex optimization. 46, 48

DOF Degrees of freedom. iv, v, 3, 8, 9, 23, 42, 50–57, 59–63, 67

DP Dynamic Programming. 18

ECOS Embedded Conic Solver. 48

ECOSBB Embedded Conic Solver with a Branch-and-Bound procedure. 48

EPA Expanding Polytope Algorithm. iii, iv, 36, 40

GJK Gilbert–Johnson–Keerthi. ii, iv, 33–35, 37–39

GUI Graphical User Interface. iii, iv, 4, 49, 69

IAI Institute for Artificial Intelligence. 3

KDL Kinematics and Dynamics Library. 46

KKT Karush–Kuhn–Tucker. 15

LGP Logic Geometric Program. 19

LQG Linear Quadratic Guassian. 19

MD Minkowski Difference. 33

MS Minkowski Sum. 33

OOS out-of-stock. 1

PD Penetration Depth. 39, 40

PSO Particle Swarm Optimization. 18

QP Quadratic Programming. v, 13, 14, 59, 64, 65

REFILLS Robotics Enabling Fully-Integrated Logistics Lines for Supermarkets. 1

RIA Robot Institute of America. 5

SCS Splitting Conic Solver. 48

SD Signed Distance. 40, 42

SOC Stochastic Optimal Control. 19

SQl_1P Sequential l_1 Quadratic Programming. ii, 31, 32

SQP Sequential Quadratic Programming. i, ii, v, 3, 4, 19–21, 23–32, 42–46, 48–50, 56, 59, 64–69

SRDF Semantic Robot Description Format. 9, 46, 48, 49

URDF Universal Robotic Description Format. 8, 9, 23, 46

1 Introduction

1.1 Motivation

In the past, robot technology allowed robots to be used in structured environments such as industries for the manufacturing proccesses. With the help of less sophisticated programming, robots could perform same movements over and over again for about thousand times a day. Unlike industries, the unstructured environments like homes, hospitals and supermarkets offers more challenges to the robots. In such a situtions, robotics technology was not ready to handle a different variety of tasks with an infinite number of combinations or even to collaborate with humans [2].

However, recent scientific and technical advancements have made robots to be used in wide applications rather than industrial process [3]. Also, these advancements made robot to be flexible and autonomous to interact with human or even with the other robot to accomplish the assigned task [4]. One such application of robots are in service industry, where the robots have to operate in an constrained environment to perform some useful task for humans in a full or semi-autonomous manner. Hence, it attracts importance for more research and explorations in service sector.

For example, a cleaning robot in a hospital has to autonomously react to the movement of doctors and patients and also has to navigate through the obstacles [5]. In [3], a robot called DAVID has been developed to collect and deliver mails, print-outs and stationery items within office environment. A vacuum cleaning robot Roomba has been developed as commercial product by iRobot [6], which can autonomously move around the house to vacuum and charge itself once the battery level reaches below the certain level [7] .

In case of retail industry, out-of-stock (OOS) is a serious problem [8], as it affects their slim profit margin because of the 4% decrease in overall sale [9]. Around 70% to 90% of OOS is due to improper maintenance to replenish missing products on the shelf [10]. In order to improvise the shelf replenishing of products, retailers can deploy human labours to periodically check the stocks on the shelf and accordingly can take necessary actions [10]. Due to the lack of human labour and increase in demand for more workers [2], robots can be deployed to perform tasks like fetching and placing the products, move them to different places as shown in the Figure 1.1.

As part of Robotics Enabling Fully-Integrated Logistics Lines for Supermarkets (RE-

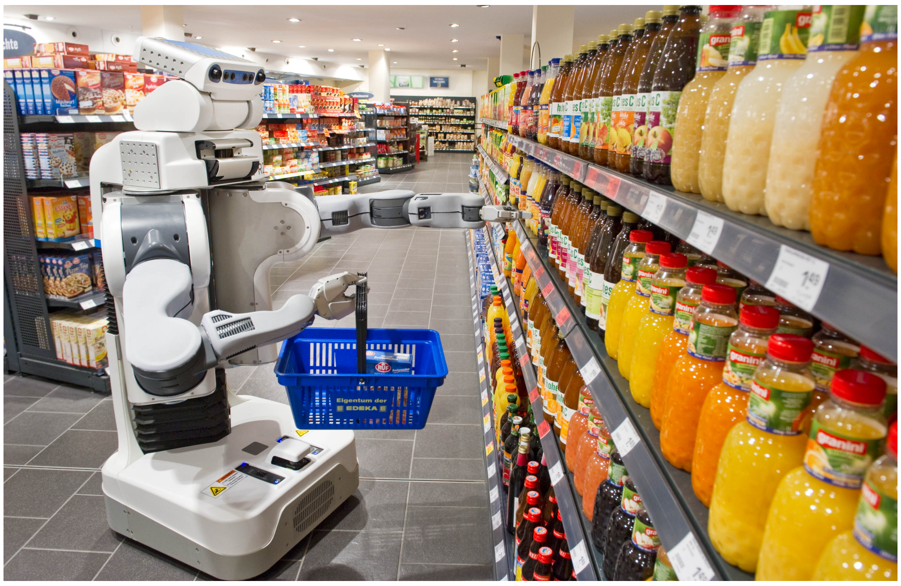


Figure 1.1: Robots deployed in super market to pick and place grocery items [11]

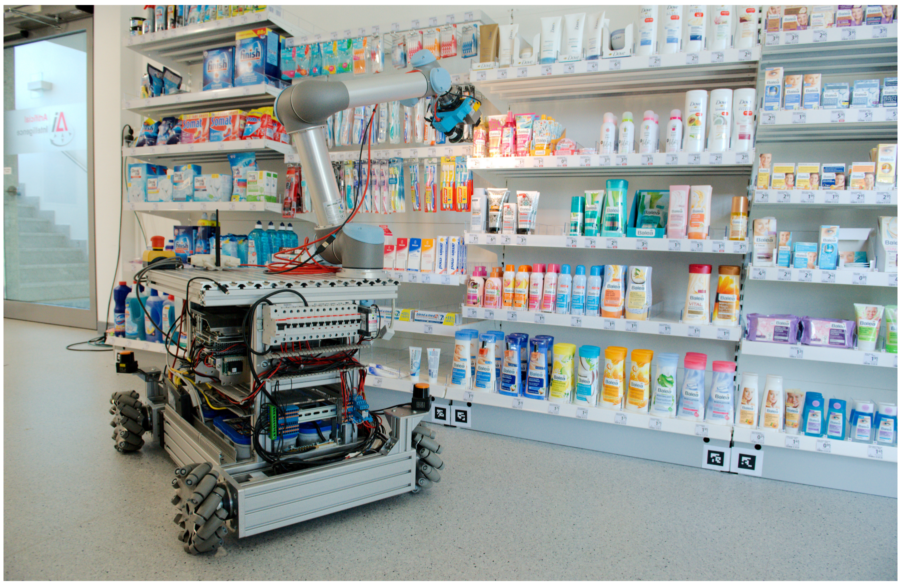


Figure 1.2: Donbot robot in action at the supermarket setup [11]

FILLS) project [12], at Institute for Artificial Intelligence (IAI), a robot called Donbot as shown in the Figure 1.2 has been developed. The main aim of this project is to improve or replace the intra-logistic wearing and monotonous operations performed by the clerks in supermarkets. Specifically, this robot expected to monitor the supermarket shelves and replenish the missing products, store delivery, pre-sorting and refilling of products in the store inventory.

There are several challenges in refilling the supermarket shelves as the retail environment is difficult and dynamic, hence deploying robot in such a situations demands robots to be intelligent enough to handle and manipulate the products. In supermarkets, one complex task is to plan a collision-free trajectory to fetch and place an item. For effective handling of the products, the planned trajectory has to be optimized, so that the robot action is smooth, jerk free and looks natural. Also, optimizing the trajectory could not only smoothens the trajectory but also could conserves energy there by staying active for more time and also could reduce the damages of the actuators caused due to the jerks in motion [13].

There are several approaches to plan a collision-free trajectory and the Sequential Quadratic Programming (SQP) is a most recent technique. This thesis implements this SQP technique for planning collision-free path within the context of shelf replenishment.

1.2 Problem Formulation

A trajectory planning in a supermarket environment is an interesting problem, as it is dynamic with the constant change in products on the shelves. Hence, each time a new trajectory has to be planned with the changed collision obstacles. This trajectory planning problem gets even more difficult, if the whole body of the robot has to be moved with more number of Degrees of freedom (DOF).

Fetch and place items on shelf is very stereotypical, where the humans plan the task once and adapt quickly with the change and doesn't plan from scratch. Hence, it is undesirable if the robot takes more time to plan trajectory for this monotonous task.

With SQP, trajectory planning is conceived as a mathematical optimization problem, where an old solution can be re-used to plan new trajectory as the planner can adapt with each input initial guess and accordingly the planning time can be improved.

While optimizing the trajectory, following challenges has to be addressed,

- The optimizer algorithm should spend most of the time on optimizing the objective cost and hence modelling of the problem has to be really fast but still accurate
- Trajectory has to optimized adhering to the robot's limitations and collision

constraints. Hence, the problem should be formulated so that it always stays in the feasible region and appropriate solution could be found

- The output of the trajectory is a set of discrete sample points. These discrete sample points could lead the robot into collision as the robot moves. Hence the collision constraints should be formulated such that the optimized trajectory is time continuous collision-free even in case of robot's self-collision

1.3 Contribution

In thesis, I have re-implemented and adapted the trajectory optimization algorithm, from Schulman et al. [1], by converting and adding collision constraints to the objective function and also by formulating time continuous collision costs for the robot's self-collision case. This algorithm has been developed in python and can solve more complex problem in less time. Additionally, a Graphical User Interface (GUI) application has also been developed to tune the SQP solver parameters and to interact with the planner. Finally, the developed algorithm has been evaluated in different scenarios. This re-implemented trajectory optimization planner can be found at [14].

This thesis report is organized as follows: In chapter 2 basic terminologies relating to robot, trajectory and optimization techniques, background works are discussed. The methodology followed and software used to implement the trajectory optimization planner are discussed in chapter 3. The evaluation results and conclusion are given in chapter 4 and chapter 5 respectively.

2 Background and Related works

2.1 Basic Terminologies

Robot

The Robot Institute of America (RIA) defines a robot as, *"A reprogrammable, multi-functional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks"* [15, pg. 5].

Robot Manipulator

A robot manipulator is a set of rigid links that are connected together by a set of joints, as shown in Figure 2.1. At the end of the manipulator, a tool such as a gripper attached to manipulate objects in the environment. The joints can one of the following six types: prismatic, revolute, cylindrical, spherical, helical and planar joints. These joints contains motor, which gets actuated to move to the links in a controlled manner to perform a given task [16, pg. 81].

The joints are numbered as i which connects the links $i-1$ and i. In a manipulator, link 0 is denoted as reference frame and link n is connected to the end effector. Such a sequence of robotic link connected from base of the robot to the end-effector link is called as kinematic chain [17, pg. 21].

In a kinematic chain, each joint has a coordinate frame to relate parent link with its child link. Usually, the coordinate frame on the link that doesn't move is chosen as base frame, so that, knowing current joint angles, the location of the end-effector in Cartesian space can be calculated [16, pg. 83].

Using homogeneous transformation T, a matrix represents position and orientation, a vector ${}^{i}r$ which is relatively expressed in i coordinate frame can be relatively expressed in j coordinate frame if the position ${}^{j}p_i$ and orientation ${}^{j}R_i$ of i frame is known relative to the j frame.

$$ {}^{j}r \;=\; {}^{j}R_i\,{}^{i}r \;+\; {}^{j}p_i \tag{2.1a}$$

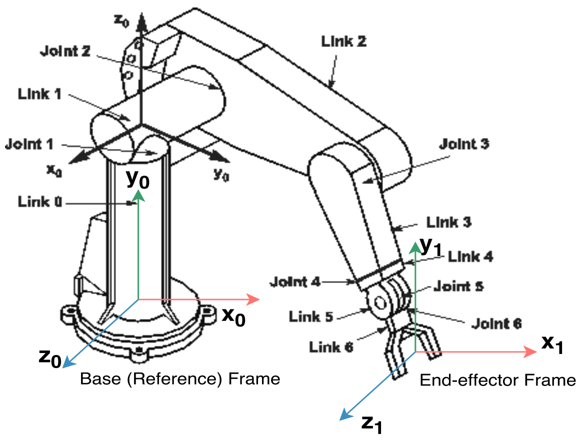


Figure 2.1: Robot Manipulator[18, pg. 11]

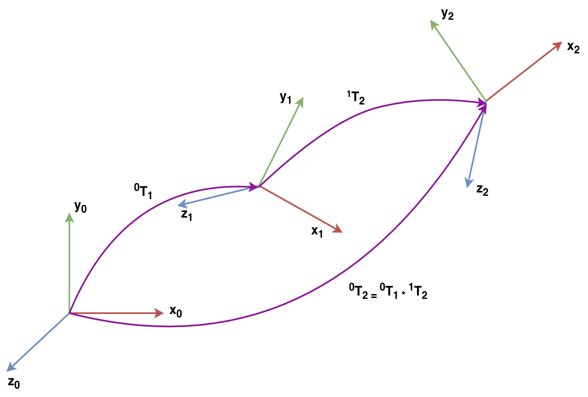


Figure 2.2: Homogeneous Transformation [19]

the above equation can be written as

$$\begin{bmatrix} {}^{i}r \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{j}R_i & {}^{j}p_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{i}r \\ 1 \end{bmatrix} \tag{2.1b}$$

where,

$$ {}^{j}T_i = \begin{bmatrix} {}^{j}R_i & {}^{j}p_i \\ 0 & 1 \end{bmatrix} \tag{2.1c}$$

$$ {}^{j}p_i = [{}^{j}p_i^x, {}^{j}p_i^y, {}^{j}p_i^z]^T \tag{2.1d}$$

${}^{j}T_i$ is a 4×4 homogeneous transformation matrix and each link i frame can be expressed in terms of the base frame as,

$$ {}^{0}T_i = {}^{0}T_1 * {}^{1}T_2 \ldots. {}^{i-2}T_{i-1} * {}^{i-1}T_i \tag{2.2}$$

Robot Kinematics

In a serially linked manipulator, the movement of each link is affected by the movement of links preceding it. This resulting movement of a link is a composition of motions with respect to its parent's link. Hence, it is necessary to describe the position and orientation of an end-effector with respect to the base coordinate system of the robot [17, pg. 21].

The study of this motion of the robot's body without the knowledge of moment or force that produces it is known as robot kinematics [20, pg. 117].

Forward Kinematics

Given all joint positions in a kinematic chain and its geometric link parameters, the problem of finding the position and orientation of the end-effector relative to the base is known as forward kinematics. This problem is solved by the transformation from a coordinate frame fixed to the end-effector to the coordinate frame(reference frame) fixed to the base of the robot [21, pg. 26].

Inverses Kinematics

In a serial chained manipulator, inverse kinematics is a problem to find joint positions with the known state of the end-effector relative to the base of the robot, provided intermediate link's geometric parameters are known [21, pg. 27].

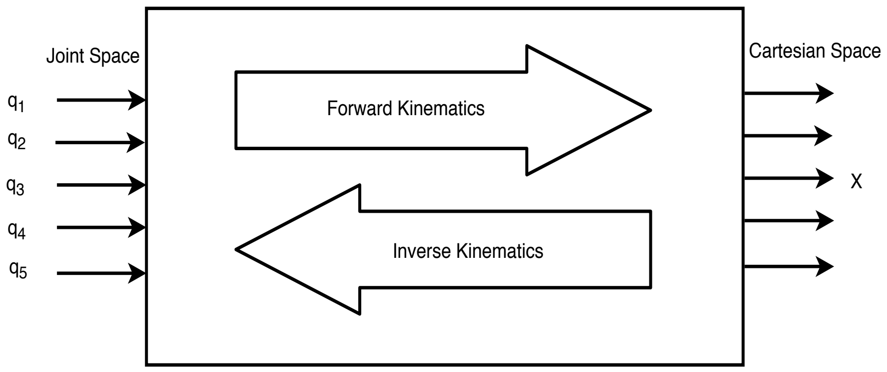


Figure 2.3: Forward and Inverse Kinematics[22]

Degrees of freedom (DOF)

DOF is the minimum number of independent variables that need to be specified to locate all parts of the robot's mechanism. In case of serially linked manipulator, the number of joints is the number of DOF that an arm posses [23, pg. 299].

Jacobian

Jacobian is an instantaneous forward kinematics problem to find the total velocity (positions and velocity) $\dot{X}$ of an end-effector with the known joint positions and velocities of a kinematic chain, where $X = [x, y, z]$. Here, the joint velocity corresponds to angular velocity in case of revolute joint and to translational velocity in case of prismatic joint.

Differentiating the forward position kinematics equation with respect to time results in equation of the following form,

$$\dot{X} = J(q)\dot{q} \tag{2.3}$$

where $\dot{q}$ is a vector of dimension N, v_n is a spatial velocity of the end-effector and $J(q)$ is called Jacobian matrix [21, pg. 29].

The Equation 2.3 relating linear and angular velocity of the end-effector, in Cartesian space, to the joint rates is known as twist [24].

Universal Robotic Description Format (URDF)

URDF is an XML specification file containing the model description of a robot, assuming that the robot joints are connected with only rigid links [25]. This file has the following information about the robot,

- Dynamic and kinematic description of the robot
- Collision information of the robot
- Visual representation of the robot

Semantic Robot Description Format (SRDF)

SRDF is again an XML specification file, which is similar to URDF, provides information that are missing in URDF [26]. Few elements contained in SRDF about the robot are,

- group: set of robot joint's link that can be used to plan trajectory for set of DOF of a robot
- group state: set of pre-defined joint values for a specified group
- chain: set representing kinematic chain of a robot
- disable collisions: set of robot link pair against which collision checking will be ignored. This is useful in case where two adjacent joints always in collision and two joints never gets collided.

Motion Planning

In robotics, a fundamental problem is to to convert high-level tasks for movements into low-level descriptions are commonly referred with the term motion planning and trajectory planning.

A motion planning problem, often referred as Piano's movers problem, it to determine appropriate motions for the robot to move without colliding any obstacles reaching a goal state [27, pg. 79].

A primary focus of robot motion planning is on the required translations and rotations to move the robot (piano), ignoring the system dynamics and differential constraints. However, recent works consider the modelling errors, differential constraints and uncertainties [27, pg. 3].

Trajectory Planning

In robotics, the term trajectory planning refers to the problem of finding both path and velocity of the robot to move, respecting its joint position and velocity limits, i.e. finding a path in space X, such that $x \in X$, where $x = (q, \dot{q})$ [27, pg. 792].

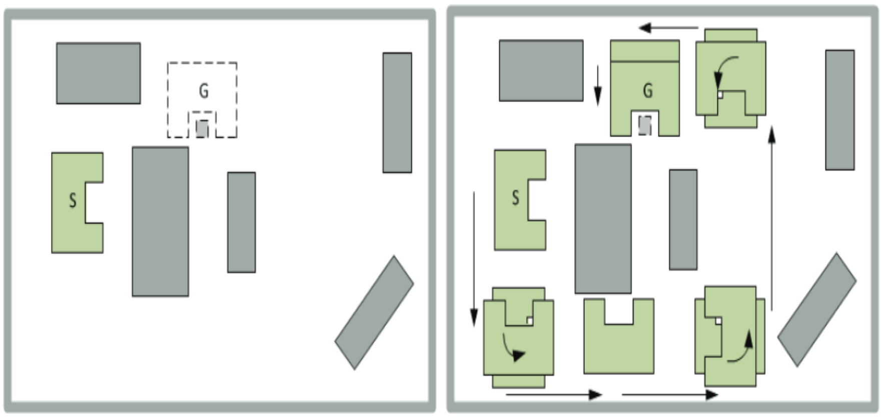


Figure 2.4: Example: Motion planning of a piano from start S to goal G [28, pg. 125]

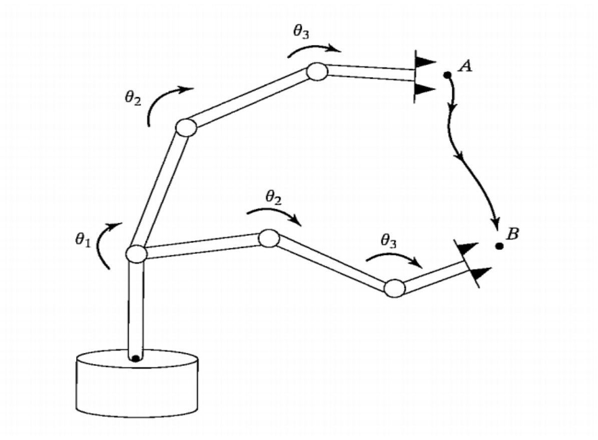


Figure 2.5: Example: Motion planning of robot Manipulator from position A to position B [18, pg. 11]

Optimization

Before discussing the concepts of optimization, following basic terms shall be introduced.

Optimization problem is defined as a problem to find a state of model, subject to constraints, for which the objective function value x^* is minimum or maximum, i.e. $f(x^*) \leq f(x)$ or $f(x^*) \geq f(x)$ for all $x \in \mathbb{R}^n$ [29, pg. 2].

Here, our goal is to minimize the robot joint velocity during manipulation subject to robot joint position and velocity limits as shown in Equation 2.4.

For example,

$$f(x) = \sum_{t \in (0,\, T)} \sum_{i \in (0,\, N)} \left\| q_{t+1,\, i} - q_{t,\, i} \right\|^2 \tag{2.4}$$

$$\tag{2.5}$$

Objective (function)

A quantitative performance measure of the system is called as objective f. In our case, energy of the system is considered as an objective. This objective varies with certain system characteristics known as variables or states. These variables are subject to restrictions due to joint position and velocity limits of the robot and with the environment (in case of collisions) constraints, in which the robot is deployed [29, pg. 2].

Model of a system

A mathematical representation of the system characteristics (energy of the manipulator) is called model of the system. The process of identifying this model, variable (or state) and its constraints is called Modelling [29, pg. 2].

Trajectory Optimization

A problem of minimizing or maximizing some performance measure of a trajectory with given a set of robot's constraints (such as joint limits and velocity limits) [30]. In the scope of this thesis, trajectory optimization means optimizing the trajectory to minimize the robot joint velocity.

Global Minimum

A point x^* is said to be global minimum point, if the objective function evaluates to least value at that point, i.e. $f(x^*) \leq f(x)$ for all $x \in \mathbb{R}^n$ [29, pg. 12].

Local Minimum

A point x^* is said to be local minimum point, if the objective function evaluates to least value in its neighborhood $\mathcal{N}$, i.e. $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}$ [29, pg. 12].. Also, if f is twice differentiable in $\mathcal{N}$, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semi-definite [29, pg. 15].

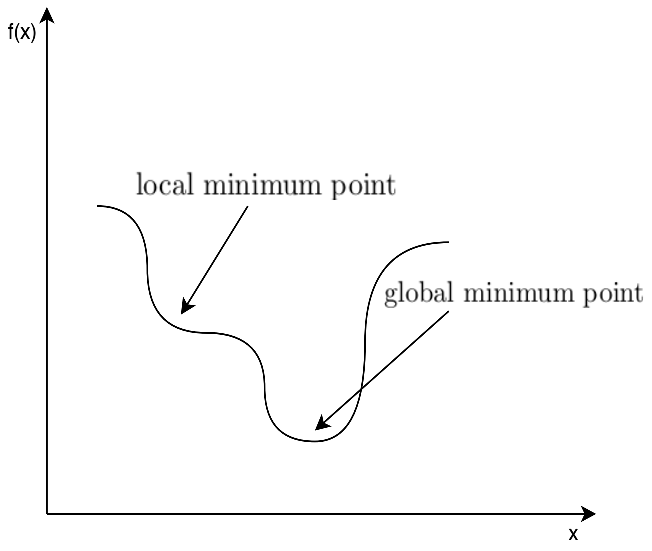


Figure 2.6: Global and local optimum

Affine functions

A sum of linear functions and a constant is known as affine function f.

An affine function has the form,

$$f(x) = Ax + b \tag{2.6}$$

where, $f : \mathbb{R}^n \to \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ [31, pg. 36].

Hessian matrix

A square matrix consisting of second-order partial derivatives of a function $f(x)$ is known as Hessian matrix H. If $f(x)$ is continuous and has all second order partial derivative terms, then the Hessian matrix has the form [32],

$$H = \begin{bmatrix} \frac{\partial f}{\partial x_1^2} & \frac{\partial f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial f}{\partial x_1\,\partial x_n} \\ \frac{\partial f}{\partial x_2\,\partial x_1} & \frac{\partial f}{\partial x_2^2} & \cdots & \frac{\partial f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_n\,\partial x_1} & \frac{\partial f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial f}{\partial x_n^2} \end{bmatrix} \tag{2.7}$$

Positive definite matrix

A square matrix P is positive definite if the product $x^T Px$ has strictly positive elements for each non-zero column of x. Similarly, if the product $x^T Px$ has positive or zero elements for each non-zero column of x, then P is called as positive semi-definite matrix[33].

2.2 Convex Optimization

A optimization problem is called a convex optimization problem when the objective and constraints are convex functions satisfying the following inequality,

$$f_i(\alpha x + \beta y) \;\leq\; \alpha f_i(x) + \beta f_i(y) \tag{2.8}$$

for all values of $x, y \in \mathbb{R}^n$ and $\in \alpha, \beta \in \mathbb{R}$ [31, pg. 1].

2.2.1 Quadratic Programming

A problem of optimizing a quadratic objective function with affine constraints is known as Quadratic Programming (QP) [31, pg. 152].

The General QP problem has the form[29, pg. 448],

$$\min_x \; f(x) = \frac{1}{2}x^T Px + q^T x \tag{2.9a}$$

$$\text{subject to } A_i^T x = b_i \qquad i = 1, 2, 3...N \tag{2.9b}$$

$$lb_j \leq G_j^T x \leq ub_j \qquad j = 1, 2, 3...M \tag{2.9c}$$

where, vector $x = [x_1, x_2, x_3, ..., x_n]^T$ is an optimization variable, N denotes number of equality constraints, M denotes number of inequality constraints. Also, $A_{N\times n}$, $G_{M\times n}$ are constraint Jacobian matrices, $b_{N\times 1}$, $lb_{M\times 1}$, $ub_{M\times 1}$ are vectors, $q_{n\times 1}$ is a Jacobian matrix, $P_{n\times n}$ is an Hessian matrix and a vector x^* is an optimal solution to the problem. Here, QP is called convex problem if P is positive semi-definite and non-convex problem if P is indefinite.

2.2.1.1 Minimization of unconstrained problem

Let us discuss the methods of minimizing unconstrained problem [31, pg. 457],

$$minimise \quad f(x) \tag{2.10}$$

where $f : R^n \to R$ is a convex function that can be twice differentiated and assume that we have a solution at x^* and its objective value is f^*. So the necessary condition for a solution at x^* to be optimal is that,

$$\nabla f(x^*) = 0 \tag{2.11}$$

We can see here that, solving Equation 2.10 is same as solving for x in Equation 2.11, hence solution can be found analytically solving Equation 2.11, but usually found by an iterative algorithm computing sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, ..., x^{(n)}$ with $\lim_{k\to\infty} f(x^{(k)}) = f^*$. The algorithm will be terminated if $f(x^{(k)}) - f^* \leq \epsilon > 0$, where ϵ is some tolerance value.

2.2.1.2 Quadratic Programming with equality constraints

The unconstrained problem doesn't arise in practical situations, where the problem are always subjected to some restrictions. The discussion for solving quadratic problem begins with equality constraints problems, but the approach is also applicable for the case of inequality constraints problems [29, pg. 451].

General QP problem with equality constraints has the form,

$$\min_x \ f(x) = \frac{1}{2} x^T P x + x^T q \tag{2.12a}$$

$$\text{subject to } A_i^T x = b_i \qquad i = 1, 2, 3...N \tag{2.12b}$$

where, $x = [x_1, x_2, x_3, ..., x_n]^T$, N denotes number of equality constraints, $A_{N\times n}$

is a constraint Jacobian matrix, $b_{N\times1}$ is a vector and $P_{n\times n}$ is an Hessian matrix and let us assume that A has a full rank N, so that the constraints in Equation 2.12 are consistent.

The necessary conditions to have x^* as a solution is that there exists a vector and the following equations are satisfied,

$$\begin{bmatrix} P & -A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix} \tag{2.13}$$

where λ is called Lagrange multipliers. We can express Equation 2.13 in a form that can be used for computation by considering $x^* = x + p$, where x is some solution estimate and p is the next step desired.

So Equation 2.13 becomes,

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ h \end{bmatrix} \tag{2.14}$$

with

$$h = Ax - b, \qquad g = q + Px, \qquad p = x^* - x \tag{2.15}$$

The matrix in 2.14 is known as Karush–Kuhn–Tucker (KKT) matrix.

If the A has full rank N and P is positive definite, then KKT matrix will be non-singular, then there is a unique solution at (x^*, λ^*) that satisfies Equation 2.13.

2.2.2 Descent methods

All the algorithms that progressively searches for the optimal solution x^* through ∇x are called descent methods [31, pg. 463] and produce sequence of $x^{(k)}$ for $k = 1, 2, ..., n$ and represented by,

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)}\Delta x^{(k)} \tag{2.16a}$$

$$\text{such that } f(x^{(k+1)}) < f(x^{(k)}) \tag{2.16b}$$

$$\text{except when } x^{(k)} \text{ is optimal solution}$$

where Δx is a vector in R^n is called step or search direction, k denotes the iteration number and $\alpha^{(k)} > 0$, except that if $x^{(k)}$ is optimal, called as step length or step size. For simplicity, the notations $x^{(k)}$, $x^{(k+1)}$, Δx_k are represented by x_k, x_{k+1}, p_k respectively and $x_{k+1} := x_k + \alpha_k p_k$.

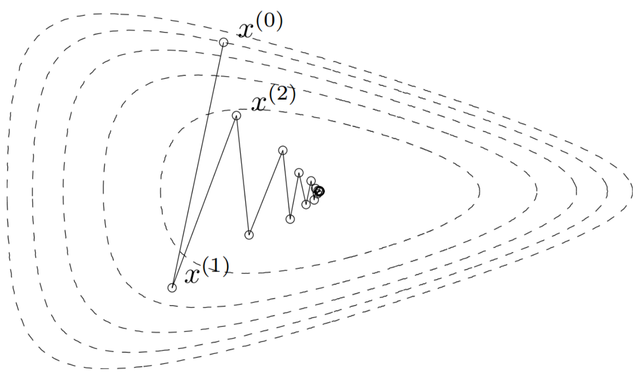


Figure 2.7: Gradient Descent Method [31, pg. 471]

Figure 2.7 shows iteration of a gradient descent problem $\in R^2$ with some objective function f, shown in dashed curves and small circles represents the iteration, solid line, connecting each iteration, shows the scaled steps of αp_k

The general descent algorithm involves two steps,

1. The initial guess has to be made for x
2. Then the descent direction has to be found for every step size α

Algorithm 1 Gradient descent method in general [31, pg. 464]

```
1: initialize initial guess $x$ and tolerance $\eta$
2: while $||\nabla f(x)||_2 \leq \eta$ do
3:     Find $p_k$
4:     For a step size $\alpha > 0$, update $x_{k+1} = x_k + \alpha p_k$
5: end while
```

Finding value for α along the line $x + \alpha p_k$ is known as line search method and α determines where next iteration is. If value of p_k is chosen as $-\nabla f(x)$, then the search method is called gradient descent method [31, pg. 466]. The line search is also known as backtracking because, the value of α starts with unit step size and reduces by a factor β until the condition $f(x + r\alpha p_k) \approx f(x) + \alpha \triangledown f(x)^T p_k < f(x) + \alpha r \triangledown f(x)^T p_k$ holds, where r ranges from 0 to 0.5 and β from 0 to 1 [31, pg. 465].

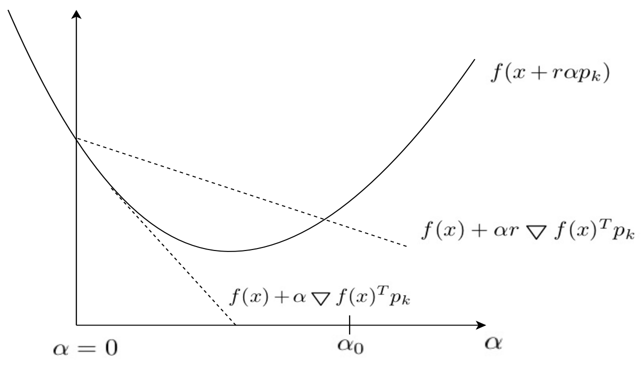

$f(x + r\alpha p_k)$
$f(x) + \alpha r \triangledown f(x)^T p_k$
$f(x) + \alpha \triangledown f(x)^T p_k$
$\alpha = 0$
α_0
α

Figure 2.8: Backtracking line search [31, pg. 465]

Figure 2.8 shows how the search for x^* is restricted between the lines, with a slope $r \in (0, t_0)$, $f(x) + \alpha \triangledown f(x)^T p_k$ and $f(x) + \alpha r \triangledown f(x)^T p_k$.

Choosing initial values of α_k and p_k determines how quick and how close the optimal value could be reached. So, the ideal choice for p_k would be of the form [29, pg. 30],

$$p_k = -H_k^{-1} \triangledown f(x_k) \tag{2.17}$$

where H_k is a non-singular and symmetric Hessian matrix.

2.3 Related work

Over the years, there has been many improvements in trajectory planning and optimization. This section briefly describes the previous and related works,

Sampling based motion planners were successful in solving wide range of problems especially the manipulation tasks [13] [34] [35] [36] [37] [38]. These algorithm randomly explores the configuration space to generate a graph with edges connecting the samples [38]. This random approach can give faster results, but in few cases the algorithm doesn't gives solution even if a solution exists [39]. Further, the output from these algorithm lacks path quality and hence needed to improve the path by smoothening [13] [40]. Another downside of these algorithm is that, at each request of the trajectory, a different solution consisting different number of sample points are generated. Though these algorithm can solve most number of problem, the solution consistency is not guaranteed.

Optimization techniques are employed, in order to have a consistent and reliable solution to the trajectory planning problem. Dynamic Programming (DP) has been used by Shareef and Steil [41] to optimize the trajectory by reducing the dimensionality of the problem to a second order differential equation by calculating arc length as path parameter from the given geometrical path. The main disadvantage of this method is that, it is computationally not efficient to find the path parameter for the given problem because all possible combinations has to be explored, making it slower and thus not suitable for real time application.

Boudjellel and Chettibi [42] treats the problem in a different way by using multi-objective optimization algorithm NSGA-II based on [43]. Here, the main focus is to minimize the travel time and the quadratic average of applied efforts on actuators by using two spline functions: a b-spline function to define path in which the robot moves and a cubic function to set-up the motion along this path. The drawback of this method is the calculation of robot's kinematic and dynamics equations. These calculations could be feasible for a simple robot but not for the complex robots and also these calculations had to be done for each robot differentlyand hence doesn't suit for real-time applications.

The motion planning problem by Oleynikova et al. [44], minimizes the cost function consisting of velocity and collision, in terms of end derivatives that can continuously compute collision-free trajectory for any newly detected obstacles. The main drawback of this approach is to efficiently segregate the explored space either as occupied or unoccupied or unknown space as the real sensor measurements is not dense enough, also additional distance field computational cost for each voxel grid on a very large environments.

In [45] Alatartsev et al. defines the problem of optimization having some relaxation in the path of end effector motion. Here, the relaxation in path means set of admissible paths that can be chosen heuristically as a continuous plan rather than point-to-point path. The limitation of this approach is the slower computation of heuristics, which is not possible to apply it in real-time.

Particle Swarm Optimization (PSO) is a stochastic optimization technique to find global solution for the given problem. This technique is used by Gao, Ding, and Yang [46] to find time optimal solution using 4-3-4 polynomial interpolation avoiding mapping relationship between motion position and polynomial interpolation coefficients. Here PSO parameters are to be calculated empirically, running time of PSO changes with the change in velocity and acceleration constraints and also uncertainty of initial movements is caused by the initial random distribution. Thus, this approach doesn't give consistent reliable results to use in real-time.

In [47] Zhu, Schmerling, and Pavone proposes Convex Elastic Smoothing (CES) algorithm to smooth out and to faster the computation of the optimization problem with a simple heuristics. Here, a tube like structure is constructed around an output trajectory of a motion planner to have a collision-free path through convex optimization technique with quadratic objective and constraints. The tube is just a bubble or circle around each way point P_i to identify colliding objects which seems to be a promising approach to find collision objects. A pitfall in this approach is that, the algorithm just smooths out the reference trajectory form sample-based motion planner and cannot operate standalone and also doesn't produces reliable results with collision-free reference trajectory.

Toussaint [48] proposed a method to solve a non-linear Stochastic Optimal Control (SOC) problem to find equivalent Linear Quadratic Guassian (LQG) perturbation model around the optimal trajectory by finding a local approximate solution with the probalisitic trajectory model coinciding the optimal trajectory. Also in [49], he formulated the problem as a Logic Geometric Program (LGP) over a end configuration space by analysing all possible end effector configurations and heuristics to inform the search over this space. Exploring the optimal solution considering all possible end effector poses needs lot of computational time making it not applicable in situations having dynamic constraints.

In the approach for trajectory optimization proposed by Kalakrishnan et al. [13], the motion planning is treated as a derivative free stochastic optimization problem that minimizes the cost of a function. The cost function, may be non-differentiable and non-smooth, is used to smoothen the trajectory along with the collisions constraints by generating and improvising the series of noisy trajectories. Ratliff et al. [50] treats the problem using a covariant gradient descent method that converges to a local optimal solution with cost considering the smoothness and obstacles. Additionally Hamiltonian Monte Carlo algorithm based on [51], [52] is used to improvise the solution better than most of the traditional optimization problem, but still it couldn't over come local minima for more difficult problem with practical time constrains.

The one important missing factor in all above algorithm is the lack of collision check between the set of generated discrete way points and hence, the trajectory may not be time continuous collision-free.

In [1] Schulman et al. treats optimization of trajectory, as a sequential convex optimization problem with l_1 penalties for inequalities and equalities constraints, converging to a local optimal solution based on Sequential Quadratic Programming (SQP). Wih efficient handling of collision avoidance constraints using Jacobian, a time continuous collision-free trajectory can be obtained. This seems to promising approach as it outperforms [50] and [13] and can also deal with the time continuous collisions. In this thesis, this approach by Schulman et al. is re-implemented in python with few adaptations explained in the following chapter.

3 Methodology

In recent years, motion planning has received interest for mobile systems and manipulation with collision avoidance as the goal. The other constraints, like torque, energy, handling constraints, generating smooth path are of less interest, also has a scope to improve [13]. Sampling based planning algorithms are also well received because it could find the path by connecting high dimensional space [50]. These methods that searches through a graph invloves workspace discretization and hence their performance decreases with increase in dimensions [53]. However, in this thesis, the motion planning task is treated as a convex optimization that finds sub-optimal collision-free trajectory using SQP method.

3.1 System Architecture

Trajectory planner as a convex optimization using SQP technique, the goal is to minimize velocity of the robot joint motion from start to goal position. The implementation of the trajectory optimization algorithm has been explained in the following sections and the system architecture is shown in Figure 3.1.

3.2 Problem Description and Modelling

The input from the user to the problem builder is as follows,

- Start and goal position of the robot
- No of samples for the trajectory
- Duration of the trajectory

The optimization algorithm takes a cost function to be optimized as an input and gives optimized output based on the constraints imposed on the cost function. Hence, the robot joint velocity has to be modelled as a cost matrix P that an optimizer solver can work on along with the robot's limitations as the constraints.

The cost function to minimize the robot joint velocity is given by [1],

$$f(x) = \sum_{t\in(0,\ T)} \sum_{i\in(0,\ N)} \left\| q_{t+1,\ i} - q_{t,\ i} \right\|^2 \tag{3.1}$$

$$\tag{3.2}$$

where,

T = No of time steps
N = No of joints
$x = [q_{0,\ start}, q_{1,\ start}, ..., q_{N,\ start}, ..., q_{0,\ goal}, q_{1,\ goal}, ..., q_{N,\ goal}]$

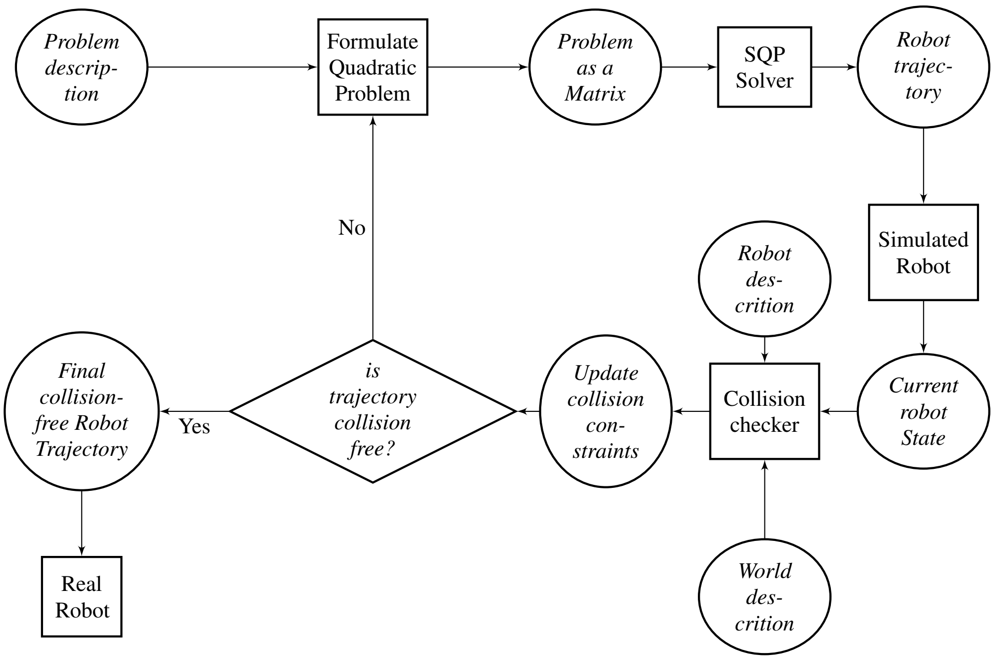


Figure 3.1: Architecture: A Robot Trajectory Optimized Planner

The cost matrix P for such a model is,

$$P = \begin{bmatrix} 2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & -2 & 0 & 0 & 0 \\ -2 & 0 & 0 & 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & 0 & -2 & 0 & 0 & 4 & 0 & 0 & -2 \\ 0 & 0 & 0 & -2 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 2 \end{bmatrix} \tag{3.3}$$

The Equation 3.3 has be to minimized respecting the robot joint position and velocity limits. These limits are considered to be constraints in the optimization problem. The Equation 3.4 represents its corresponding model G,

$$G = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

The matrix G is an inequality constraint to the problem, since it is bounded by joint position's and velocity's lower limits lbG and upper ubG limits.

$$lbG \leq G(x) \leq ubG \tag{3.5}$$

To plan a robot trajectory, apart from the robot joints position and velocity limits, robot start q_{start} and goal q_{goal} configurations has to be given as equality constraints to the problem, represented as,

$$A(x) = b \tag{3.6a}$$

where,

$$A = \begin{bmatrix} q_{start} \\ q_{goal} \end{bmatrix} \tag{3.6b}$$

The robot limitations required for optimization problem are taken from URDF file of the robot. With all above required variables the robot trajectory optimization problem is defined as,

$$\min_{x} f(x) = \sum_{t} \sum_{i} \left\| q_{t+1,i} - q_{t,i} \right\|^2 \tag{3.7a}$$

$$lbG \leq G(x) \leq ubG \tag{3.7b}$$

$$A(x) = b \tag{3.7c}$$

3.3 Sequential Quadratic Programming (SQP)

In a robot, each joint will have position and velocity limit leading to a large problem with many number of constraints. For example, a 10 Degrees of freedom (DOF) robot, will have 10 position and 10 velocity constraints summing to 20 constraints. Hence, the problem has to be scaled down into multiple smaller sub-problem, so that solving them yields a reliable solution to the original problem.

The strategy of breaking down and solving large problem into equivalent convex sub-problems by a suitable conversion is called as SQP. This SQP method can be solved using trust region method effectively to find Δx even when the constraints are non-linear [29, pg. 529].

3.3.1 Trust region method

Trust region method is similar to the linear search method as discussed in subsection 2.2.2, where the optimal value of the quadratic objective function is found iteratively from the initial guess. Line search focuses on finding step length α_k in different search direction p_k, while trust region method minimizing the step length within the trust region around the model representation of the objective function in each iterate. This minimized step is considered as minimized approximate of the model for the next iteration. Thus, the direction and step are chosen simultaneously [29, pg. 68].

The general approach to solve the quadratic problem is to construct a model function m_k with the gathered information on objective function f, whose characteristics is similar to the original f around the current iterate x_k. The search for the optimal solution x^* is restricted with a small region around x_k, so that the model m_k will be a good approximation of f.

Now, the original problem of minimizing f is reduced to solve the following subproblem in a sequence,

$$\min_p \; m_k(x_k + p) \tag{3.8}$$

where p restricted within region of trust. The original objective function f is approximated into a model function m_k, usually, by a second order Taylor-series expansion at each iterate x_k, given by,

$$f(x_k + p) = f_k + g_k^T p + \frac{1}{2} p^T \, \nabla^2 \, f(x_k + tp)p \tag{3.9}$$

where $f_k = f(x_k)$, $g_k = \nabla f(x_k)$ and t is a scaler $\in (0, 1)$. With Hessian approximation H_k for the second order term in m_k, Equation 3.9 becomes,

$$m_k = f_k + g_k^T p + \frac{1}{2} p^T H_k p \tag{3.10}$$

The difference between the Equation 3.9 and Equation 3.10 is equal to $O(||p||_2)$ will be small if p is small. Thus, when the Hessian approximation is as close to the objective function f, then $||p||$ will be small, so the now the objective is to minimize p, such that $||p|| \leq \Delta_k$

If the chosen step p_k happens to be the bad approximate of the model, then trust region is reduced or else, the trust region will be expanded. If the trust region is too small or large, the algorithm may miss the step to find the approximate minimizer m_k

closer to actual objective function. A step that has failed indicates that the model is not proper representation of the original objective function over the current iteration. Hence, the initial choice for the trust region in very crucial in solving the objective.

$$\min_{p \in R^n} m_k(p) = f_k + g_k^T p + \frac{1}{2} p^T H_k p \tag{3.11a}$$

$$\tag{3.11b}$$

subject to linearized constraints,

$$A_k + \nabla A_k^T p = 0, i = 1, 2, 3...N \tag{3.11c}$$

$$G_k + \nabla G_k^T p \geq 0, j = 1, 2, 3...M \tag{3.11d}$$

along with,

$$||p|| \leq \Delta_k \tag{3.11e}$$

where, $A_k = A_i^T(x) - b_i(x)$, $G_k = G_i^T(x) - ubG_i$ or $lbg_i - G_i^T(x)$ and Δ_k is called trust region radius within which the algorithm searches for the step minimizer for f and the norm $||\cdot||$ is an Euclidean norm.

The main advantage of imposing trust region constraint is that [29, pg. 546],

- the Hessian matrix H_k doesn't need to be a positive definite matrix
- the trust region decides how good the step has to be even, if there are singular Hessian and Jacobian matrices and provides the mechanism to enforce global convergence

Figure 3.2 shows difference in approach between trust region and line search. From the previous iteration and steps, the model m_k is constructed from the function $f(x_k)$ and its derivative at $\nabla f(x_k)$, whose contours are shown in dashed ellipse. Here, we can see that the step direction from line search along the the minimizer step m_k causes small reduction in f, although initial step length is optimal, whereas, the reduction of m_k within the trust region (dotted circle) produces more reduction in f making it better progress in achieving the solution.

The key aspect of minimizing the objective function f is to choose the right value of Δ_k for each iteration, such that there is a good agreement in reduction between the

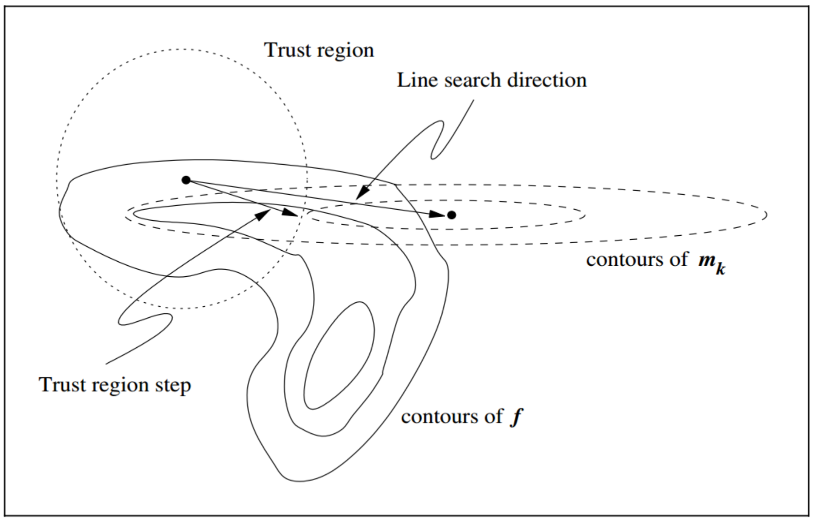


Figure 3.2: Trust region vs Line search [29, pg. 67]

original objective function f and the model objective function m_k. To evaluate this, for given p_k x_k, following expression is used,

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} \tag{3.12}$$

In the Equation 3.12, the numerator is known as actual reduction in original objective function and denominator is called as predicted reduction (i.e., reduction by the model function in f). Since the predicted reduction is computed between $p = 0$ and p_k, it will be always positive, hence the ρ_k will be negative only when the objective value of $f(x_k + p_k)$ is greater than $f(x_k)$. Also, ρ_k is closer to 1 if the model m_k and the objective f are in good agreement, therefore the trust region radius Δ_k is increased and vice versa, when ρ_k is closer to 0 or less than 0, the trust region radius Δ_k is decreased for the next iteration [29, pg. 69]. The Figure 3.3 shows how search is done on each sub problems at each iteration with different trust region radii Δ,

Different norms can be used to define the trust region the Euclidean norm, e.g.,

$$||p||_1 \leq \Delta_k \qquad \text{or} \qquad ||p||_2 \leq \Delta_k \qquad \text{or} \qquad ||p||_\infty \leq \Delta_k \tag{3.13}$$

When the norm is Euclidean, then the trust region will be a sphere around the

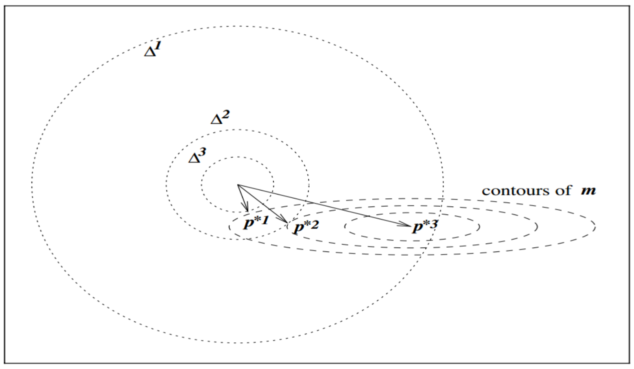


Figure 3.3: Trust region for different radii $\Delta^1, \Delta^2, \Delta^3$ [29, pg. 70]

model and when ∞ norm is used, the trust region will be just a rectangular box defined by,

$$x_k + p_k \geq 0, \qquad p \geq -\Delta_k e, \qquad p \leq \Delta_k e \tag{3.14}$$

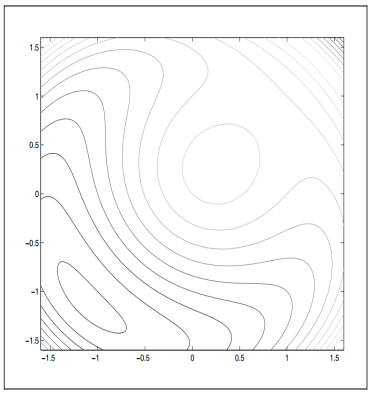
where $e = (1, 1, ..., 1)^T$ and the problem solution can be easily calculated using bounded-constrained quadratic programming. When practical problems are larger, the calculation of Hessian H_k is expensive, the use of ∞ norm with a rectangular trust region is a convenient method to solve the problem defined by Equation 3.11 [29, pg. 97].

3.3.2 L1 Penalty method

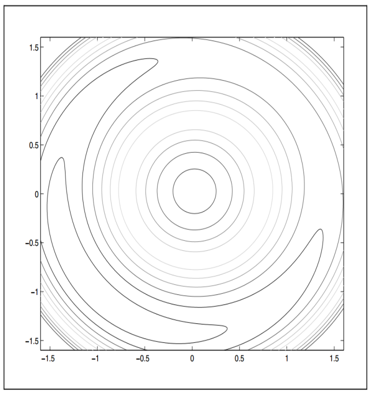
The original constrained problem can converted into its equivalent sequence of unconstrained sub-problem by adding the constraints back to the original problem [29, pg. 497].

- To the original objective function,
- an additional penalty term is added for each constraint, which is positive when the minimizer x_k violates the constraints or zero otherwise.

The common approach would be to multiply the additive penalty term with a larger coefficient value that penalizes for every constraint violations, thus making sure that the penalty minimizer function stays within the feasible region of the constraints.



(a) Penalty method: Contours of some $Q(x, \mu)$ at $\mu = 1$ [29, pg. 500]



(b) Penalty method: Contours of some $Q(x, \mu)$ at $\mu = 10$ [29, pg. 500]

Figure 3.4: Penalty method: Contours of $Q(x, \mu)$ at $\mu = 1, 10$ [29, pg. 499, 500]

This type of adding the constraints to the main objective function is called as penalty methods. Such an unconstrained problem will be of the form,

$$Q(x, \mu) = f(x) + \frac{\mu}{2} \sum_{i=1}^{Z} c_i^2(x) \tag{3.15}$$

where, Z is the number of constraints, $c_i^2(x)$ is square of the constraint, $\mu > 0$ is called the penalty parameter and the Equation 3.15 is called quadratic penalty function, also when the μ increase to ∞, the constraint violations are penalized more severely. Since the penalty terms are smooth, the techniques used for unconstrained optimization can be used to solve the problem over x_k at each iteration.

The Algorithm 2 describes how the problem is solved iteratively for different values of μ_k at each iteration. Here, the penalty parameter μ_k can be adaptively chosen depending on how difficult to solve the penalty function. The termination condition $\|\nabla_x Q(x, \mu_k\| \leq \tau_k$ doesn't satisfies as the iteration moves the minimizer x_k out of feasible region or the case where the constrains violations doesn't decrease significantly, the penalty parameter has to be increased to a very larger value to bring the problem back in track within the feasible region.

The simple way to update the penalty term μ_k in Algorithm 2 is to initialise with some small value and to increase in terms of 5 or 10 till the minimiser x_k pulled into the

Algorithm 2 Penalty Method Algorithm [29, pg. 501]

```
 1: initialize initial, max trust region size $\mu_0$ and $\mu_{max}$ respectively
 2: initialize non-negative sequence $\tau_k$ with $\tau_k \rightarrow 0$
 3: initialize initial guess $x_0$
 4: initialize penalty increase ratio $\mu_r$
 5: set $x_k = x_0$
 6: set $\mu_k = \mu_0$
 7: for $k = 0, 1, 2, ..., \mu_{max}$ do
 8:     Minimise for $x_k$ from $Q(x_k, \mu_k)$
 9:     find $\rho_k = \frac{f(x_k)-f(x_k+p_k)}{m_k(0)-m_k(p_k)}$, terminating when $||\nabla_x Q(x, \mu_k)|| \leq \tau_k$
10:     if converged then
11:         return $x_k$
12:     end if
13:     Set new penalty $\mu_k = \mu_r * \mu_k$
14:     choose different starting point $x_k$
15: end for
```

feasible region with some tolerance. The initial choice of μ_0 is very important because, choosing it very small may results in bad minimizer x_k and needs many iterations to converge. In these iterations, there is a possibility that x_k can be pulled away before reaching the solution x^*, in such a situations the algorithm has to be terminated and restarted with initial guess x_0 and different μ_0. In contrast to this, choosing very large value of μ_0, the algorithm needs many number of iterations before the penalty function can be minimized [29, pg. 511].

Only when the problem contains equality constraint, the function $Q(x, \mu_k)$ is smooth and the unconstrained problem techniques can be used to solve the problem. However, in a practical application where there would be inequality constraints the problem is non-smooth. Also, the quadratic penalty function doesn't gives same minimizer x_k solution of the non-linear program for different positive values of μ. In such a situations, exact penalty functions can be used, which yields single minimizer x_k leading to a exact solution to a large non-linear problems.

General non-linear programming problem with $l1$ (exact) penalty function is expressed as,

$$\phi_1(x, \mu) = f(x) + \mu(\sum_{i \in \varepsilon} |c_i(x)| + \sum_{i \in \Gamma} |c_i(x)|^-) \tag{3.16}$$

where, $[y]^- = max(0, -y)$. The name is $l1$ penalty because the penalty term

μ is $l1$ norm times the constraint violation. Also, it should be here noted that the Equation 3.16 is not differential at some value of x causes $c_i(x) = 0$. for $i \in N \cup M$. Due to this, the algorithms that were discussed previously can't be used directly to solve $\phi_i(x, \mu)$. So the constraints c_i has to be linearised and the non-linear part of the objective function f has to replaced by it equivalent quadratic function, as below [29, pg. 512]:

$$q(x,\mu) = f(x) \ + \ \nabla f(x)^T p + \frac{1}{2} p^T H p + \mu\Big(\sum_{i \in \varepsilon} |c_i(x)p \ + \ \nabla c_i(x)^T p| + \sum_{i \in \Gamma} |c_i(x) + \nabla \ c_i(x)^T p|^- \Big) \tag{3.17}$$

where, H is a symmetric matrix containing second derivative of f.

3.3.3 Merit Function

Merit functions are used to decide whether the iteration has to be accepted or not. This merit function decides how the step size changes in line search method and how the trust-region changes (expanded or shrinked). The following discussion shows how the non-smooth, exact $l1$ merit function can be used to solve the problem of type,

$$\min_x \ f(x) \tag{3.18}$$

$$\text{subject to } c(x) = 0 \tag{3.19}$$

The inequality constraints $c(x) > 0$ can be converted into a equality constraints of the form,

$$c(x, s) = c(x) - s = 0 \tag{3.20}$$

where $s \geq 0$ is slack variable vector. Therefore, any problem with inequality and constraints can be converted into a form expressed in Equation 3.18 and the corresponding $l1$ merit function takes the form,

$$\phi_1(x, \mu) = f(x) + \mu \, ||c(x)||_1 \tag{3.21}$$

The Equation 3.21 can be solved iteratively by choosing different values of μ_k wih increase in value at each iteration.

3.3.4 Sequential l_1 Quadratic Programming (SQl_1P)

In some cases, the problem doesn't gives solution because of trust region constraint, even if the constraints Equation 3.11c and Equation 3.11d are compatible. In Figure 3.5 shows an example, having only one equality constraint and any step p satisfying equality constraints should lie outside of trust region Δ_k. This example shows how a consistent equality and inequality constraints may not give solution if the norm of the solution is restricted.

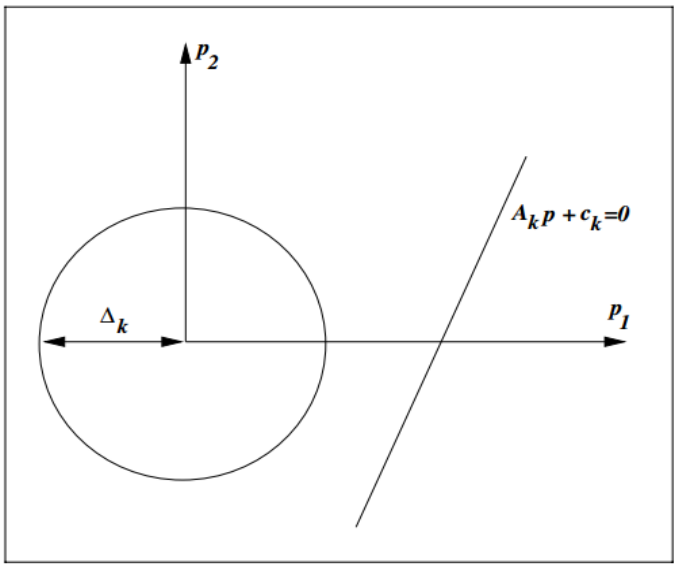


Figure 3.5: Inconsistent trust region model [29, pg. 547]

One possible way to get the solution, in such a case, is simply to increase Δ_k until the step p that satisfies linear constraint lies with in the trust region. This strategy doesn't serve the purpose of the trust region constraint within which the approximation of the original objective function and constraints are trusted [29, pg. 546].

An infeasible problem can be turned into feasible problem, by linearizing the constraints into the objective function with l_1 penalty term into the problem as,

$$\begin{aligned}\min_p \; q_\mu(p) = f(x_k) \; + \; \nabla f(x_k)^T p + \frac{1}{2} p^T H p \; + \\ \mu(\sum_{i \in \varepsilon} |G_i(x_k) \; + \; \nabla G_i(x_k)^T p| \; + \sum_{i \in \Gamma} |A_i(x_k) \; + \; \nabla A_i(x_k)^T p|^-)\end{aligned} \tag{3.22}$$

subject to,

$$\|p\|_\infty \leq \Delta_k$$

where, $y^- = max(0, -y)$. This type of problem is a smooth quadratic problem, which is always consistent because the trust region has ∞ norm, can be be solved by quadratic programming algorithm.

The SQl_1P approach has following advantages,

- Inconsistent problem has been converted into consistent problem, which always ensures the satisfaction of trust region constraint
- The second order term can be used directly or replaced by quasi-Newton approximation and doesn't need to be positive definite

In Equation 3.22, l_1 merit function decides step acceptance p_k. After compuation of p_k in each iteration, the ratio ρ_k is calculated using Equation 3.12 and the trust region rules, according to the Algorithm 3, decides the step acceptance p_k.

Algorithm 3 Trust Region Algorithm [29, pg. 549]

```
1:  initialize initial and max trust region size $\Delta_0$ and $\Delta_{max}$ respectively
2:  initialize $\epsilon > 0, \eta, \gamma \in [0, 1]$
3:  initialize initial guess $x_0$ and initial $\Delta_0 > 0$
4:  set $x_k = x_0$, $\Delta_k = \Delta_0$
5:  for $\mu = 10, 100, 1000, ..., \mu_{max}$ do
6:      for $k = 0, 1, 2, ..., iterationLimit$ do
7:          compute $p_k$ from Equation 3.22
8:          find $\rho_k$ from Equation 3.12
9:          if $\rho_k > \eta$ then
10:             $x_{k+1} = x_k + p_k$
11:             choose $\Delta_{k+1}$ such that $\Delta_{k+1} \geq \Delta_k$
12:         else
13:             $x_{k+1} = x_k$
14:             choose $\Delta_{k+1}$ such that $\Delta_{k+1} \leq \gamma \|p_k\|$
15:         end if
16:     end for
17: end for
```

3.4 Collision checker

The goal of robot motion planning is to make robot perform some task. While performing a given task, the robot shouldn't collide with itself (self-collision) or with any obstacles in the surroundings. Hence, collision checking is more important step in motion planning. The following sections will discuss techniques to detect collision and to find distance from the obstacles.

3.4.1 Gilbert–Johnson–Keerthi (GJK) algorithm

In [54], Gilbert, Johnson, and Keerthi proposed an efficient way to detect collision between two objects and to measure distance between them along a path describing both position and orientation in configuration space continuously.

An object in an Euclidean space is a set of non-empty points. Let A, B denotes two rigid bodies in R^3 and the natural choice to measure the proximity between them is with Euclidean distance, i.e. shortest line segment connecting two objects, as it signifies the physical distance and invariant with different choices of the coordinate system [54],

The minimum distance between two objects A and B is given by,

$$d(A, B) = min\{|a - b|,\ a \in A, b \in B\} \tag{3.23}$$

Instead of computing Equation 3.23, the same can be obtained by calculating distance between the origin and the difference d given by [55],

$$d(A, B) = ||\upsilon(A - B)|| \tag{3.24a}$$

where,

$$\upsilon(C) \in C \tag{3.24b}$$

$$||\upsilon(C)|| = min\{||x||\,, x \in C\} \tag{3.24c}$$

The difference in Equation 3.24a is known as Minkowski Difference (MD) and $\upsilon(C)$ is a nearest point to the origin in configuration space C. Minkowski Sum (MS) of two objects A and B, as shown in Figure 3.6 and MD is sum of A and -B with -A denotes the reflection of A about origin O, can be calculated by adding A_i and B_i, where i represents every point in an object [56]. MD is also known as Configuration Space

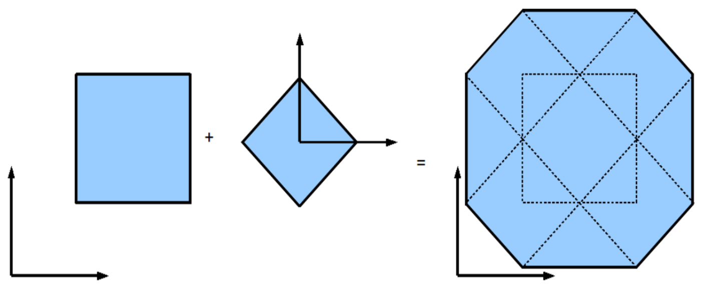


Figure 3.6: Minkowski sum: geometric representation [56]

Obstacle (CSO) and the goal of the GJK algorithm is find if the region enclosed by CSO contains origin [57].

When two objects intersects, there will be common points on both the objects and hence the difference between those points will be zero. Hence, in order to detect collision, it is sufficient to check if the CSO contains origin.

It is a tedious procedure to find CSO for each point in A against each point in B and hence it is sufficient to find difference between a farthest point in A along a direction v and a farthest point in B along a direction $-v$, which is guaranteed to appear in the CSO region.

$$S_{A,B}(v) = S_A(v) - S_B(-v) \tag{3.25a}$$

where,

$$S_A(v) = max\{v \cdot a, a \in A\} \tag{3.25b}$$

The Equation 3.25b is knowns support function, which maps a vector v to a farthest point, called as support point, on an object A along a given direction v and the plane containing support point is referred as supporting plane. If the vector is pointing perpendicular to a surface, then all the points on a supporting plane are farthest points, in this case the support function return its centroid [57], [56].

For a primitive shapes, its not necessary to compute all possible $v \cdot x$ and their precise support mapping function can be directly used, thus GJK algorithm can give faster results [55]. Few examples are,

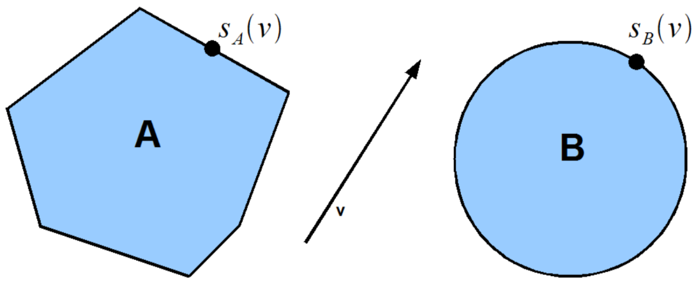


Figure 3.7: Support points on a polygon A and a circle B along the vector V [56]

For a Polytope,

$$S_A(v) = max\{v \cdot a, a \in vertices(A)\} \tag{3.26a}$$

For a Box with extents $2\eta_x, 2\eta_y, 2\eta_z$,

$$S_{A(x,\ y,\ z)}(v) = (sgn(x)\eta_x, sgn(y)\eta_y, sgn(z)\eta_z)^T \tag{3.26b}$$

For a Cylinder with origin as center, y being central axis, radius ρ, top at y $= \eta$ and bottom y $= -\eta$,

$$S_{A(x,\ y,\ z)}(v) = \begin{cases} (\frac{\rho}{\sigma}x, sgn(y)\eta, \frac{\rho}{\sigma}z), & \text{if } x < 0. \\ (0, sgn(y)\eta, 0), & \text{otherwise.} \end{cases} \tag{3.26c}$$

where $sgn(x) = 1$, if $x > 0$, and 1, otherwise.

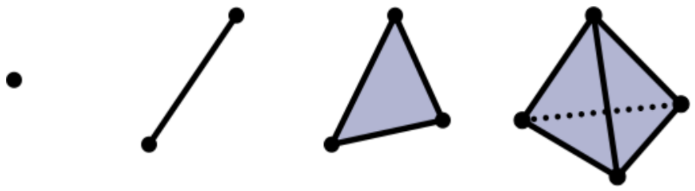


Figure 3.8: Different types of Simplex [57]

Let n be a fixed integer, n-simplex is a convex hull with $n + 1$ points in n-dimensional space, as shown in Figure 3.8. Thus, a point, a line segment, a triangle and tetrahedron has 0-, 1-, 2-, 3- simplex respectively, as shown in Figure 3.8.

The GJK algorithm explained in Algorithm 4, is a descent method, begins with

Algorithm 4 GJK Algorithm [55]

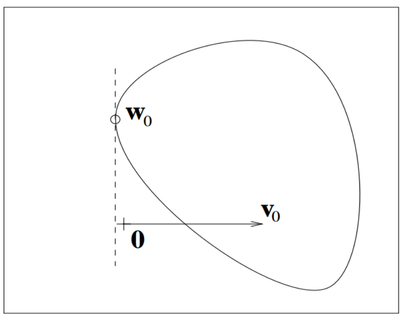
```
1: initialize set of vertices of simplex in $k$-th iteration $W_k = \emptyset$
2: initialize closest point to the origin inside the simplex convexhull $v_k$ with any
   arbitrary point $v_0$ in $C$
3: initialize $close\_enough := false$ and a tolerance value $\epsilon = 0.01$
4: initialize lower bound value $\mu = 0$
5: while ($close\_enough$ is $False$ and $v_k$ is not 0) do
6:     $w_k = S_{A,B}(-v_k)$
7:     $\delta = v_k \cdot \frac{w_k}{||v_k||}$
8:     $\mu = max(\mu, \delta)$
9:     $close\_enough = ||v_k|| - \mu \leq \epsilon$
10:    if $close\_enough$ is $false$ then
11:        $v_k = \upsilon(ConvexHull(W_k \bigcup w_k))$
12:        $W_k$ = smallest $X \subseteq W_k \bigcup$ such that $v_k \in ConvexHull(X)$
13:    end if
14: end while
15: return $||v_k||$
```

an arbitrary point v_0 within CSO and an empty simplex set W_k, on each iteration a support point v_{k+1} along the direction $-v_k$ is found and the smallest simplex enclosing the support closest to origin added into W_k until no further v_k other than itself is found [57].

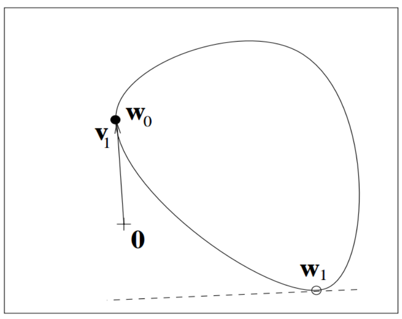
Figure 3.10 shows case of collision, where the algorithm started with an arbitrary point v_0 and an empty simplex set $W = \emptyset$, finds an support point v_1 and adds a vertex w_0 into W. Since, there is only one vertex in the simplex, the vertex itself is a support point and the next support point is found towards the origin to find new vertex w_1 forming 2-simplex segment. With 2-simplex a line segment is formed and the closest point to the origin, on the line segment, is taken as a new support point to find next vertex w_2 normal. With 3-simplex, the closest point to the origin again on a triangle gives same vertex w_2, hence the algorithm is terminated. With the simplex set enclosing the origin, represented by plus sign, the existence of collision between objects can be found.

3.4.2 Expanding Polytope Algorithm (EPA)

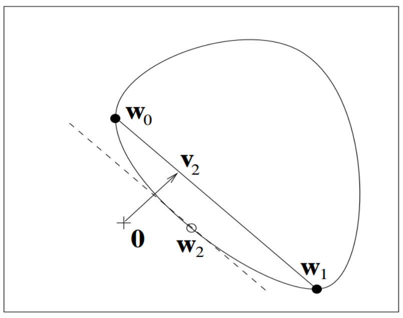
To move objects in space without colliding, it is important to know if the path taken by an object can lead into collision. In such a case, to detect collision between the objects, it is sufficient to know if they are in contact and the distance between them



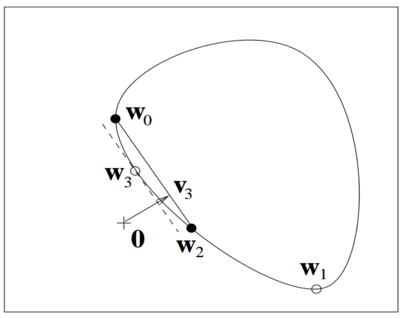
(a) $k = 0, W = \emptyset$



(b) $k = 1, W = \{w_0\}$

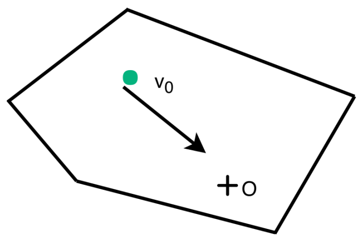


(c) $k = 2, W = \{w_0, w_1\}$

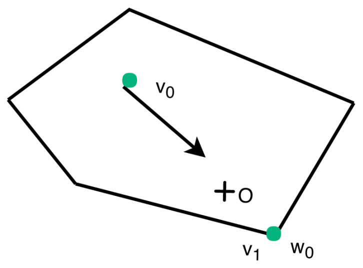


(d) $k = 3, W = \{w_0, w_2\}$

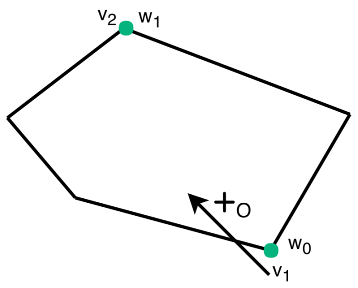
Figure 3.9: Iterations of GJK algorithm without collision. The dashed line denotes support planes $H(-v_k, v_k \cdot w_k)$ and W_k drawn in solid line [55]



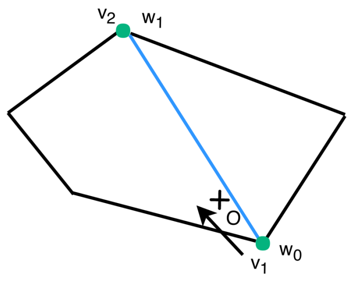
(a) $k = 0, W = \emptyset$



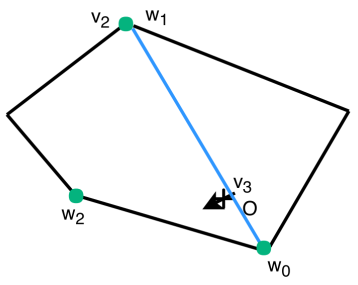
(b) $k = 1, W = \{w_0\}$



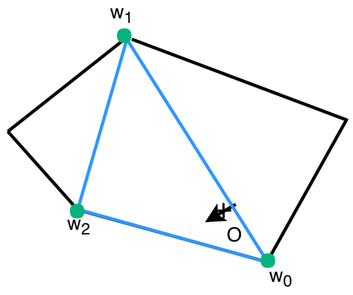
(c) $k = 2, W = \{w_0\}$



(d) $k = 2, W = \{w_0, w_1\}$

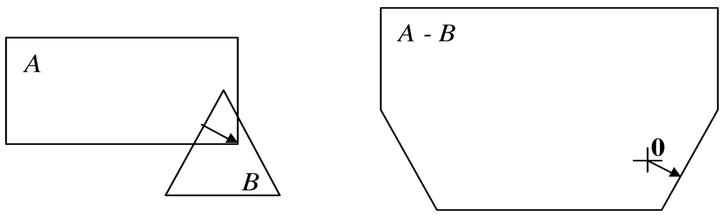


(e) $k = 3, W = \{w_0, w_1\}$



(f) $k = 3, W = \{w_0, w_1, w_2\}$

Figure 3.10: Iterations of GJK algorithm without collision. The dashed line denotes support planes $H(-v_k, v_k \cdot w_k)$ and W_k drawn in solid line [55]



(a) Intersection of objects A and B (b) $k = 1, W = \{w_0\}$

Figure 3.11: Intersection of two objects A and B and corresponding penetration depth [58]

is not required. But, in order to bring the objects out of collision and to continue the motion in the planned path, it is necessary to know how far the two objects have been overlapped. This shortest vector (distance) between the intersecting objects along which either of the object has to be translated, to bring them just in contact, is known as Penetration Depth (PD) [58].

From Equation 3.23, the distance between two objects is given by

$$d(A, B) = min\{\|x\|, x \in A - B\} \tag{3.27a}$$

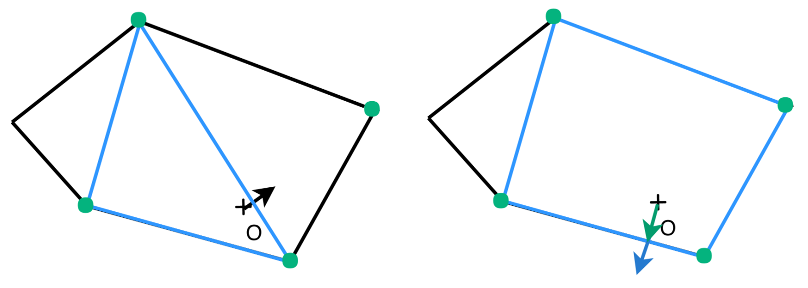
where as, the penetration depth p is given by [58],

$$p(A, B) = inf\{\|x\|, x \notin A - B\} \tag{3.27b}$$

PD given by infimum (greatest lower bound), is a vector between the origin and a non-unique point lying anywhere on the boundary of CSO formed by intersection of objects A and B as shown in Figure 3.11.

In [58], Van Den Bergen, had used output of GJK and continues to find PD by,

1. starting with last iteration polytope of GJK containing origin and vertex on the boundary of CSO,
2. find a support point in the direction of normal vector pointing to the origin from an edge closer to it,
3. add the found vertex w to the simplex list.
4. repeat this procedure until no more support vertex w, other than itself can be found



(a) $k = 0$ (b) $k = 1$

Figure 3.12: Iterations of EPA. Green arrow and blue arrow represents PD and contact normal $\vec{n}$ respectively [58]

5. length of the vector point from the origin to the supporting plane is the required penetration depth, as shown in green color in Figure 3.12 and the blue arrow denotes the contact normal $\vec{n}$ pointing from one object to the other.

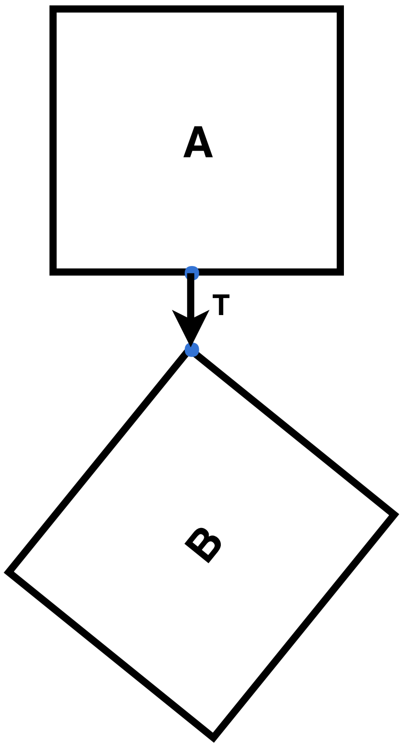
PD expressed in terms of length of the vector pointing from origin to the supporting plane is given by,

$$p(A, B) = v \cdot \frac{w}{||v||} \tag{3.28}$$

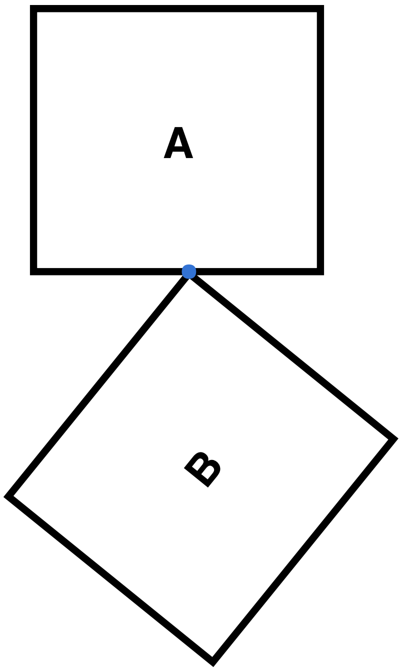
With PD, the objects can be moved to bring them just in contact, but in order to ensure the moving object is free of collisions, these objects have to be moved further to maintain some distance $d_{threshold}$. Hence, it is preferable to have a parameter describing collision and separation between the objects, that is zero when the objects are just in contact on the boundary, positive when there is no collision and negative when there is collision as shown in Figure 3.13. Such a parameter is called as Signed Distance (SD), defined as [1],

$$sd(A, B) = d(A, B) - p(A, B) \tag{3.29}$$

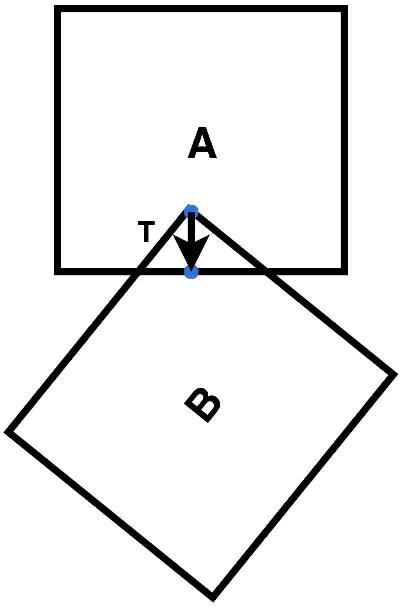
In order to ensure the robot is not colliding with any objects during motion, it is important to perform collision check between each link of the robot and the obstacles and between each pair of robot links. Also, it is sufficient to check nearby obstacles for collision within certain distance d_{check}, such that $d_{threshold} \leq d_{check}$. Now, with SD, the collision constraint for the optimization problem can be defined as [1],



(a) $sd \geq 0$



(b) $sd = 0$



(c) $sd \leq 0$

Figure 3.13: Signed distance between objects A and B [1]

For collision with obstacles,

$$sd(A_i, O_j) \geq d_{threshold}, \; i = 1, 2, 3\text{...No of robot links} \qquad (3.30a)$$
$$i = 1, 2, 3\text{...No of obsctales}$$

For robot self collision,

$$sd(A_i, A_j) \geq d_{threshold}, i, j = 1, 2, 3\text{...No of robot links} \qquad (3.30b)$$

Let ${}^{A}A$ and ${}^{W}A_A$ represents an object A in space R^3 in its local and world coordinate system respectively, a point p_A be a contact point on A, $\hat{n}$ be the normal vector along SD, ${}^{W}U_A$ be a statinary object's pose and ${}^{W}U_A(\theta)$ be an object's pose that varies with a parameter θ. With these terms, the SD can be expressed as [1],

$$sd(A(\theta), B) \approx \hat{n} \cdot ({}^{W}U_A(\theta)\, p_A - {}^{W}U_B\, p_B) \qquad (3.31)$$

With robot's Jacobian with respect to its DOF of a point p_A, the Equation 3.31 can be linearised as,

$$\nabla_{\theta_0} sd(A(\theta), B)\Big|_{\theta_0} \approx \hat{n} \cdot J_{p_A}(\theta_0) \qquad (3.32a)$$
$$sd(A(\theta), B) \approx sd(A(\theta_0), B) + \hat{n}^T \cdot J_{p_A}(\theta_0)(\theta - \theta_0) \qquad (3.32b)$$

3.5 Time continuous Collision free trajectory

Using the Equation 3.32 as a constraint, the SQP solver produces robot trajectory consisting of discrete way-points. These way-points have to be interpolated to form time continuous trajectory before feeding into the robot, might lead into collision as shown in Figure 3.14. Hence, the collisions must be checked between the convex hull swept by the states $A(t)$ and $A(t+1)$ of moving object A and an obstacle B, in order to generate a continuous in time collision free trajectory.

Let the object A move in space between the time interval $[t,\; t+1]$ and the SD of the convex hull sweept in this interval is defined as [1],

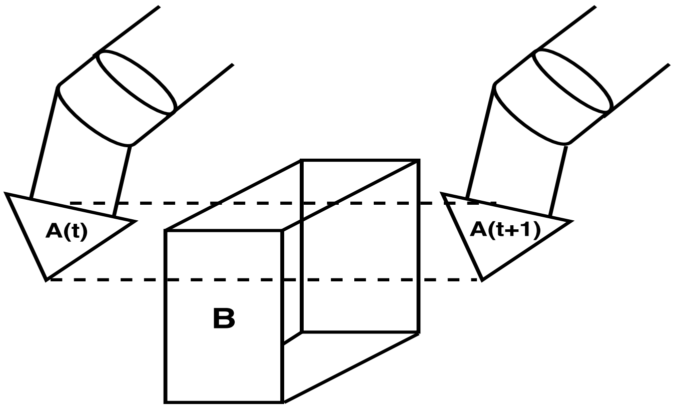


Figure 3.14: Swept convex hull volume of a time continous trajectory having collision

$$sd_{(A(t),\, A(t+1))}(v) = \begin{cases} sd_{A(t)}, & \text{if } sd_{A(t)} > sd_{A(t+1)} \\ sd_{A(t+1)} & \text{otherwise} \end{cases} \tag{3.33}$$

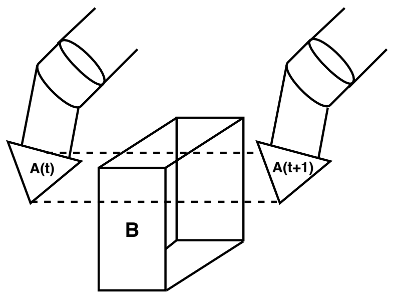
Since the object A is moving in space, there are three possibilities for the collision to occur:

1. Collision at the state $A(t)$
2. Collision at the state $A(t + 1)$
3. Collision between the states $A(t)$ and $A(t + 1)$

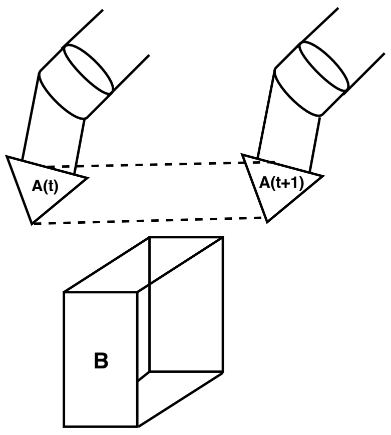
The constraint with Equation 3.32 takes care first two discrete collision cases. For the third case, it is necessary to know collision hit fraction Ω between the states $A(t)$ and $A(t + 1)$, so that accordingly collision constraint for the SQP solver can be formed. Let $p_{A(t)}$, $p_{A(t+1)}$ and p_c be the closest point on object A at state t, $t + 1$ and on the swept volume of convex hull respectively. The contact fraction Ω can be approximated, assuming linear approximation between the states $A(t)$ and $A(t + 1)$ as,

$$\Omega = \frac{\left\| p_{A(t+1)} - p_c \right\|}{\left\| p_{A(t+1)} - p_c \right\| + \left\| p_{A(t)} - p_c \right\|} \tag{3.34a}$$

With contact fraction Ω, Equation 3.32 can be extended as [1],



(a) robot link having collision



(b) robot link free from collision

Figure 3.15: Illustratoin of robot having collision and free from collision. The dashed line represents interpolated sample points between the states $A(t)$ and $A(t+1)$

$$
\begin{aligned}
sd(A(\theta(t),\ \theta(t+1)), B) \approx\ & sd(A(\theta_0),\ \theta(t+1)), B) \\
& + \Omega\, \hat{n}^T \cdot J_{p_{A(t)}}(\theta_0(t))(\theta(t) - \theta_0(t)) \\
& + (\Omega - 1)\, \hat{n}^T \cdot J_{p_{A(t+1)}}(\theta_0(t+1))(\theta(t+1) - \theta_0(t+1))
\end{aligned}
\tag{3.35}
$$

With the Equation 3.35 as a constraint, the SQP solver generates a trajectory that will move the in between region (swept convex hull volume) covered out of collision during robot link motion as shown in Figure 3.15. The Equation 3.35 not only covers the case where the robot link, during motion, undergoes translation but also rotation, as the correction factor is always below 1 cm [1].

Since the problem is solved using l_1 penalty method, the optimization problem reduced to minimizing the variable p_k according to the Equation 3.22 instead of original minimizer x_k, where p_k represents change in the original optimization variable Δx. Hence the term $\theta(t) - \theta_0(t)$ can be replaced by p_k.

Now, with all the required constraints, the final SQP optimization is given by,

$$\begin{aligned} \min_p \; q_\mu(p) = f(x_k) \;+\; \nabla f(x_k)^T p + \frac{1}{2} p^T H p + \\ \mu(\sum_{i\in\varepsilon} |G_i(x_k) \;+\; \nabla G_i(x_k)^T p| \;+ \sum_{i\in\Gamma} |A_i(x_k) \;+\; \nabla A_i(x_k)^T p|^-) \end{aligned} \tag{3.36a}$$

subject to,

$$\|p\|_\infty \leq \Delta_k \tag{3.36b}$$

$$\begin{aligned} C_i(x_k) = sd(A(p_k), B) - \Omega\, \hat{n}^T \cdot J_{p_{A(t)}}(\theta_0(t)) p_k \\ -(\Omega - 1)\, \hat{n}^T \cdot J_{p_{A(t+1)}}(\theta_0(t+1)) p_{k+1} - d_{threshold} \geq 0 \end{aligned} \tag{3.36c}$$

Further the collision constraint in the Equation 3.36c can be formulated as an l_1 penalty term and added back to the Equation 3.36a to form,

$$\begin{aligned} \min_p \; q_\mu(p) = f(x_k) \;+\; \nabla f(x_k)^T p + \frac{1}{2} p^T H p + \\ \mu(\sum_{i\in\varepsilon} |G_i(x_k) \;+\; \nabla G_i(x_k)^T p| \;+ \sum_{i\in\Gamma} |A_i(x_k) \;+\; \nabla A_i(x_k)^T p|^-) \\ \mu(\sum_{i\in\varepsilon} |C_i(x_k) \;+\; \nabla C_i(x_k)^T p| \end{aligned} \tag{3.37a}$$

subject to,

$$\|p\|_\infty \leq \Delta_k \tag{3.37b}$$

With the above Equation 3.37a, the optimization problem is further reduced to have only trust region constraint and hence, it can be easily solved. Now, this problem can be solved using SQP solver with a linearly interpolated trajectory discrete points as an initial guess x_0, from start to goal position, using Algorithm 3.

3.6 Software

Having formulated necessary objective cost and collision constraints for the SQP solver, a nice and preferably easy interface is required to interpret robot model, state, limitations and to query collision information from the simulation environment. This interface designed using a software architecture as shown in Figure 3.16.

As seen in the Figure 3.16, the robot is loaded into simulation environment reading its links geometry, collision information from Universal Robotic Description Format (URDF) and Semantic Robot Description Format (SRDF) files and also any objects such as table or shelf on which the robot has to perform task are similarly added into the planning scene. The parameters such as d_{check}, $d_{threshold}$ and the default parameters of SQP solver are similarly load loaded from a configuration file. Once the basic configurations for the robot and the SQP solver are loaded, the user just have to choose goal state either from the SRDF or can define new goal pose to which the robot should move from the current state.

With the robot limitations, start state and goal state a model of the problem is built in the Problem Modelling class along with the linearly interpolated initial guess x_0. This problem model is then sent to the SQP problem solver class. With x_0, the optimization problem produces a minimizer x_1 respecting all constraints and checked for collision for each time step of a trajectory between each link and obstacle and also between each link of the robot (self-collision). Hence, a new set of collision constraints is generated and the SQP problem is solved again for x_k until all constraints including collision constraints are satisfied.

The joint limitations of the robot are read from the URDF using urdf parser [59]. The l_1 penalty constraints representing robot joint position and velocity limits remains same throughout the problem, but the collision constraints and accordingly the trust region Δ_k gets altered until collision free solution is found.

Few tools libraries used in trajectory optimization application are:

- Python [60] : an high level interpreter language
- Numpy [61] : a fundamental scientific fundamental python package
- Kinematics and Dynamics Library (KDL) [62] : a framework to model a robot and to calculate kinematics of a chain

In the following section the other libraries used to plan an optimized collision free trajectory are discussed.

3.6.1 Convex optimization solver

In order to solve a complex problem, it is preferable to express and model a problem in a natural way rather than in a restricted standard form. Python-embedded modeling language for convex optimization (CVXPY) is one such interface that can be used to solve complex convex problem and also supports extensions for parallel solving of non-convex problems with high-level abstractions [63].

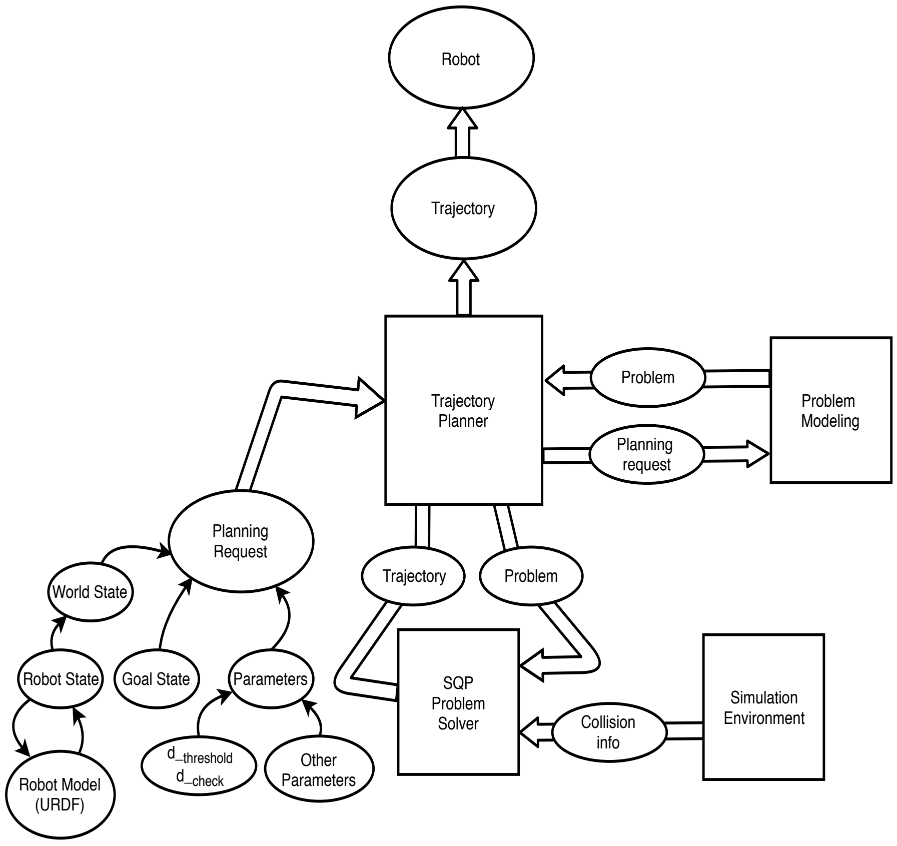


Figure 3.16: Software Architecture

The main advantage of CVXPY is that, it internally has an interface to different types of solvers: Embedded Conic Solver (ECOS) [64], Embedded Conic Solver with a Branch-and-Bound procedure (ECOSBB) [64], Splitting Conic Solver (SCS) [65] and A Python package for convex optimization (CVXOPT) [66], so that only once the problem needed to be modelled and can solved across all solvers.

3.7 Bullet physics

Bullet physics engine [67] is an open source library to simulate soft and rigid body dynamics and to detect and resolve continuous and discrete collision detection between convex and convex objects of all primitive shapes by providing updated objects' world transform [68].

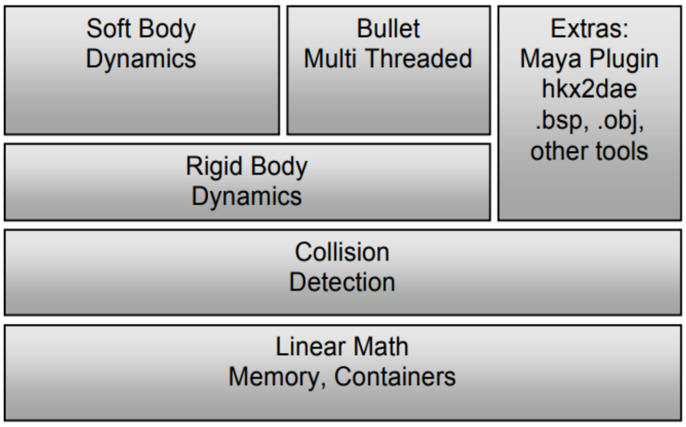

Soft Body Dynamics
Bullet Multi Threaded
Extras: Maya Plugin hkx2dae .bsp, .obj, other tools
Rigid Body Dynamics
Collision Detection
Linear Math Memory, Containers

Figure 3.17: Main components of bullet physics [68]

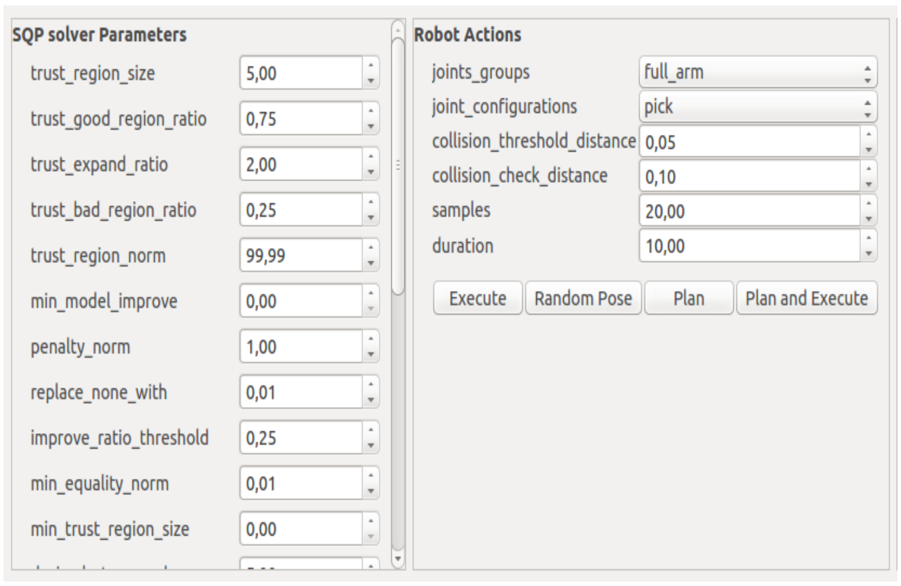
In each iteration of the SQP solver, the optimized solution x^* is checked for collision against all obstacles in the environments and between each link of the robot (self-collision). In such a case, the calculations can be made faster by ignoring few collisions check between robot links, thus generating less number of constraints. For e.g. collision check between the adjacent links that are always in contact and between links that will not collide at all can be avoided. These filter for the collision check are loaded from SRDF using srdfdom parser [69].

Since the python version of Bullet physics doesn't have continuous collision check functionality at the time of implementation, its python wrapper was extended to support the needed functionality according to the Equation 3.33 and Equation 3.34. This extended version of PyBullet can be found at [70] in convexsweep branch.

3.8 Graphical User Interface (GUI)

The default SQP solver parameters are stored in a configuration file. Any change in these parameters, the trajectory solver application has to be restarted and hence it is difficult to find appropriate solver parameters in run-time. To ease the use of trajectory planner application, a GUI, using qt framework [71], is designed to choose SQP solver parameters, as shown in Figure 3.18. An user can also interact with the GUI to change different goal states and joint configurations, that are predefined in the SRDF file, $d_{threshold}$ and d_{check} distance and to plan and execute the trajectory.

The SQP solver parameters, that are stored in a yaml file, are loaded using [72] into the GUI. Changing these parameters are automatically get saved to same yaml file.



SQP solver Parameters

trust_region_size	5,00
trust_good_region_ratio	0,75
trust_expand_ratio	2,00
trust_bad_region_ratio	0,25
trust_region_norm	99,99
min_model_improve	0,00
penalty_norm	1,00
replace_none_with	0,01
improve_ratio_threshold	0,25
min_equality_norm	0,01
min_trust_region_size	0,00

Robot Actions

joints_groups	full_arm
joint_configurations	pick
collision_threshold_distance	0,05
collision_check_distance	0,10
samples	20,00
duration	10,00

Figure 3.18: A GUI application for Trajectory Optimization Planner

4 Evaluation

The trajectory planning algorithm was evaluated with two different robots in different use cases: a 7 DOF robot arm mounted on a table and an 11 DOF robot with 37 links is placed in-front of a shelf. In both the case, the algorithm is evaluated for,

- Reliability: how many problems got solved vs total number of problem given
- Consistency: given a same problem, how far the solution is close to one another in terms of solving time and generated trajectory
- Performance: given different problems, how fast the planner produces the trajectory result
- Reachability: how does the planner reacts if the given problem's solution can't be reached
- Sensitivity: how does the planner reacts to the different SQP solver parameters

4.1 Experiment Setup

The trajectory optimization planner algorithm is evaluated in a simulated environment using bullet physics [67] running on a computer with the configurations given in Table 4.1

Table 4.1: Configuration of the system on which the algorithm was evaluated

Processor	Intel core i7 @ 3.40 GHz x
Memory	32 GB
Graphics	NVIDIA GeForce GTX 670
Operating system	Ubuntu 14.04 LTS

4.1.1 Pick and place using Kuka arm

As a simple example, a 7 DOF lbr iiwa [73] Kuka arm mounted on a table as shown in Figure 4.1 is considered. With this setup, trajectory was planned for the robot arm moving from random start to random goal positions with randomly placed box (obstacle) on the table.

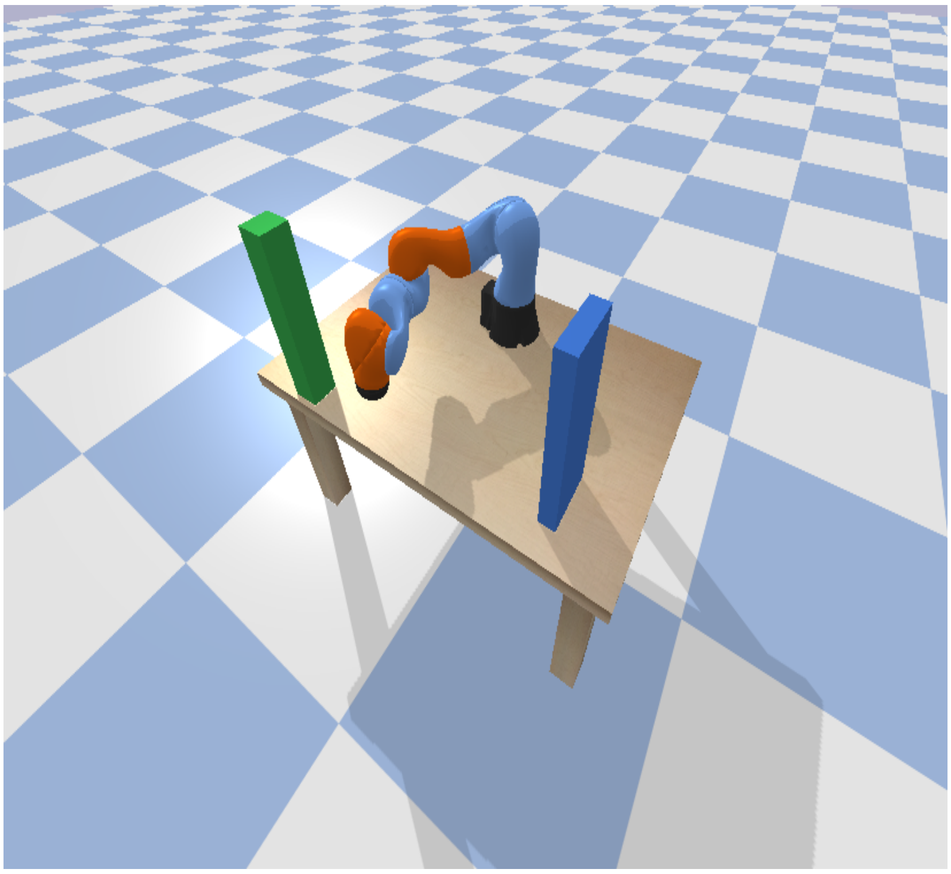


Figure 4.1: A 7 DOF lbr iiwa arm mounted on a table

4.1.2 Pick and place of supermarket products using Donbot robot

In this case, an 11 DOF Donbot robot with 37 links is considered. This robot has a 6 DOF UR5 [74] arm, with 2 DOF end-effector, mounted on a 3 DOF mobile base to move in x, y, z directions. This robot is a complex use case, as the robot robot's arm can be driven separately or the whole robot body could be moved.

As an evaluation case, this Donbot robot was placed in front of a supermarket shelf as shown in Figure 4.2 is considered. With this setup, trajectory was planned for the robot arm moving from random start to random goal positions with random number of objects placed on the shelf randomly. Also, in a similar situation the trajectory is planned for the entire robot.

Apart from the default ignore collision between the links, the collision check between the links ur5_wrist and ur5_forearm of the Donbot robot were ignored, as the Bullet collision checker always reports a collision of $0.02m$ between them.

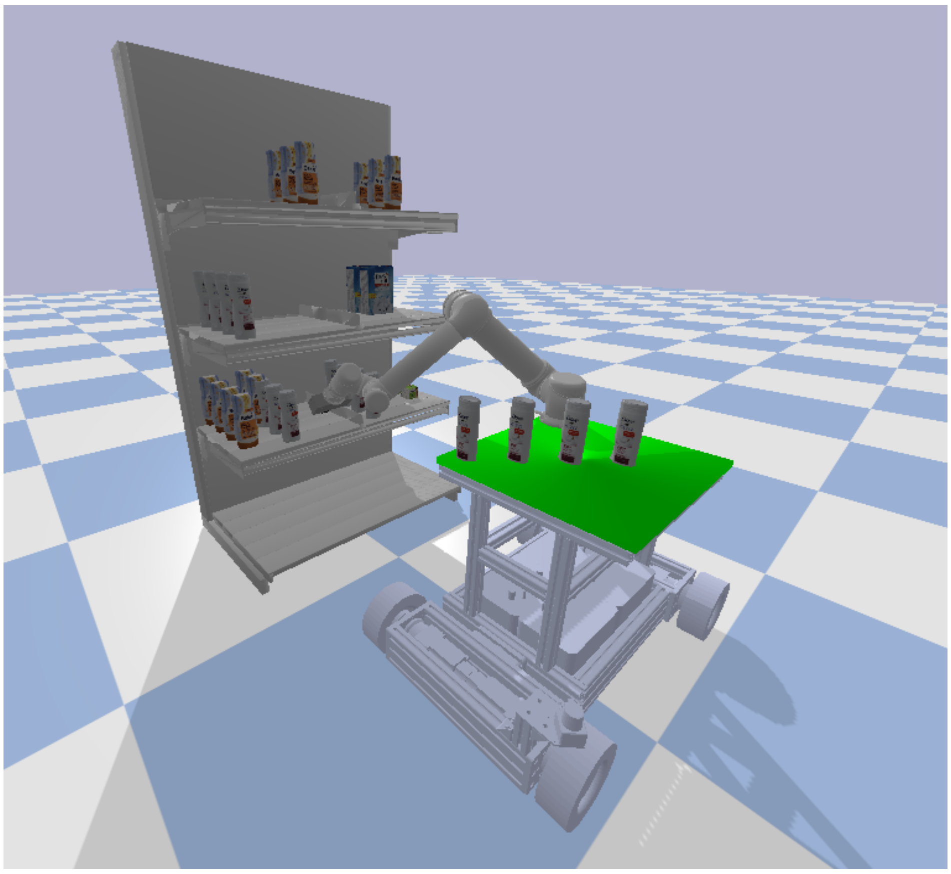


Figure 4.2: A 11 DOF Donbot robot in front of supermarket shelf

4.2 Results

A sample trajectory optimization planner output for the Kuka arm, Donbot-arm and Donbot-whole body is shown in Figure 4.3, Figure 4.4 and Figure 4.5 respectively. The blue line in these figure represents given initial linearly interpolated guess from start to goal position and green line represents the out of the trajectory planner. In spite of existence of collision in the initial trajectory, the developed algorithm could generate a collision-free trajectory in most of the cases considered.

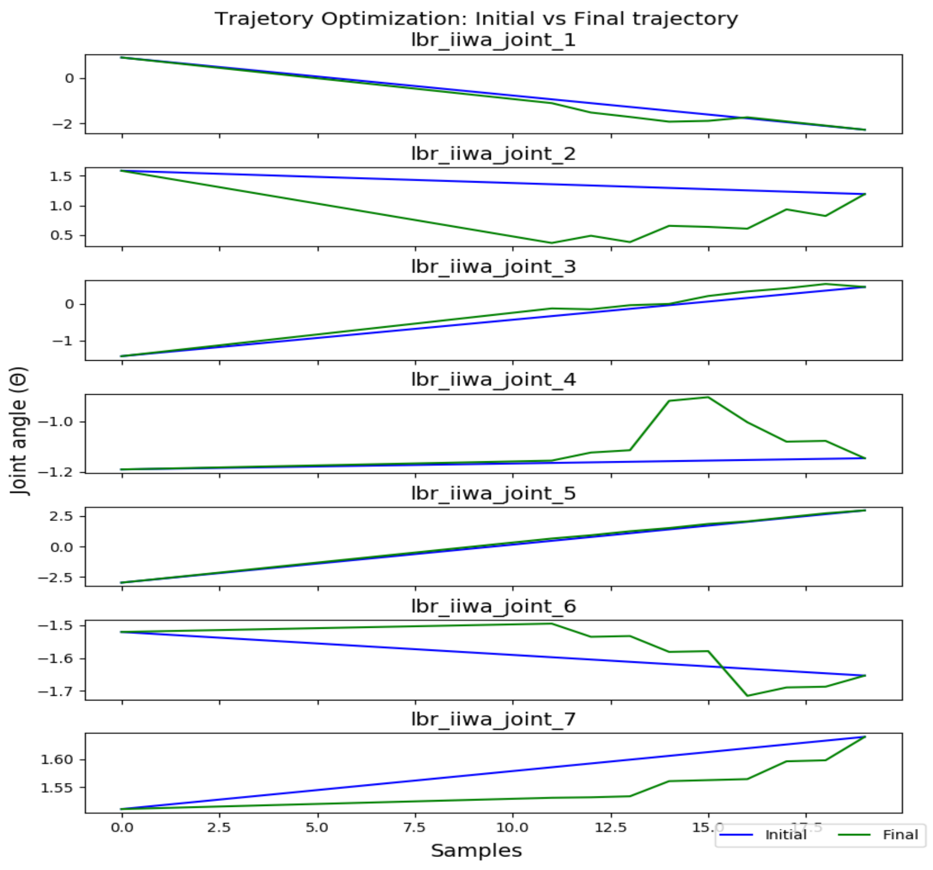


Figure 4.3: A sample planned trajectory of a 7 DOF Kuka arm

Run-time performance and reliability

In order to test the reliability and run-time performance of the developed trajectory optimization planner, the algorithm was tested with a random start and goal positions,

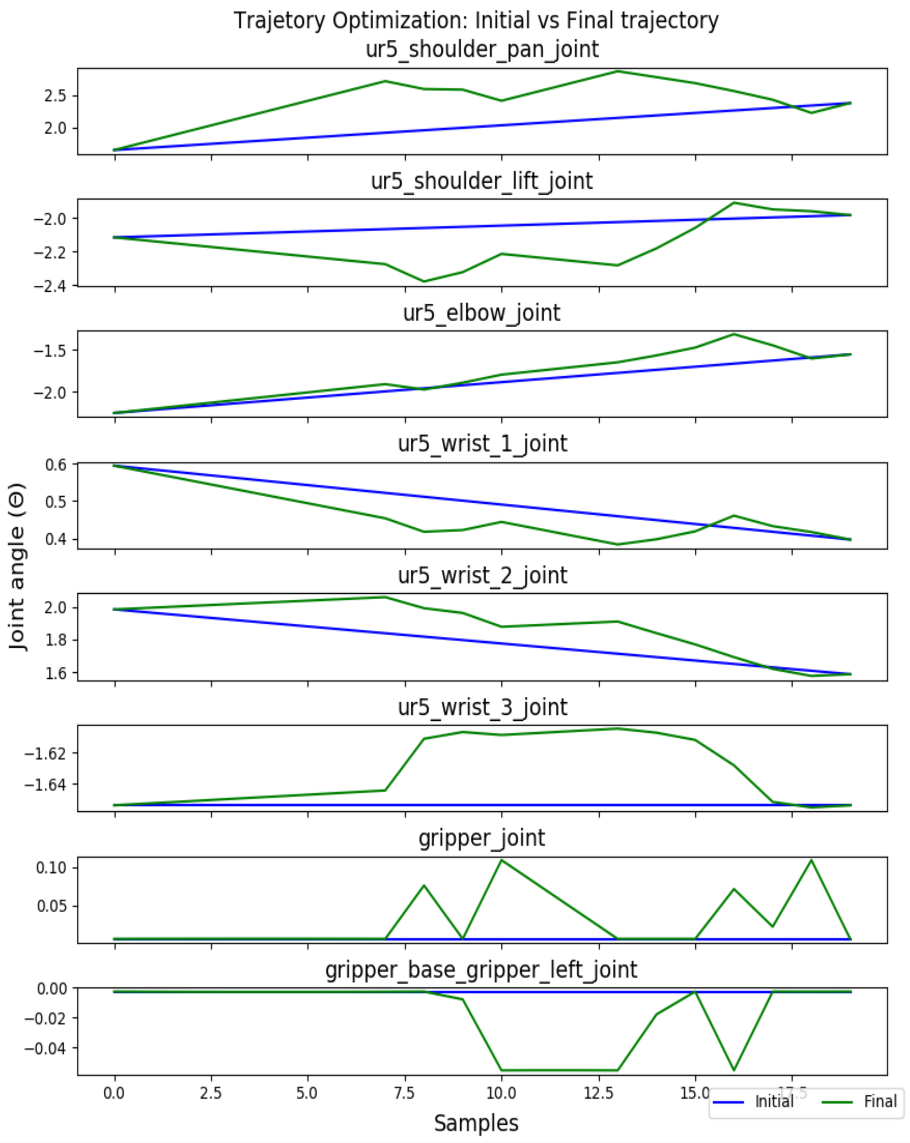


Figure 4.4: A sample planned trajectory of a 6 DOF arm of Donbot

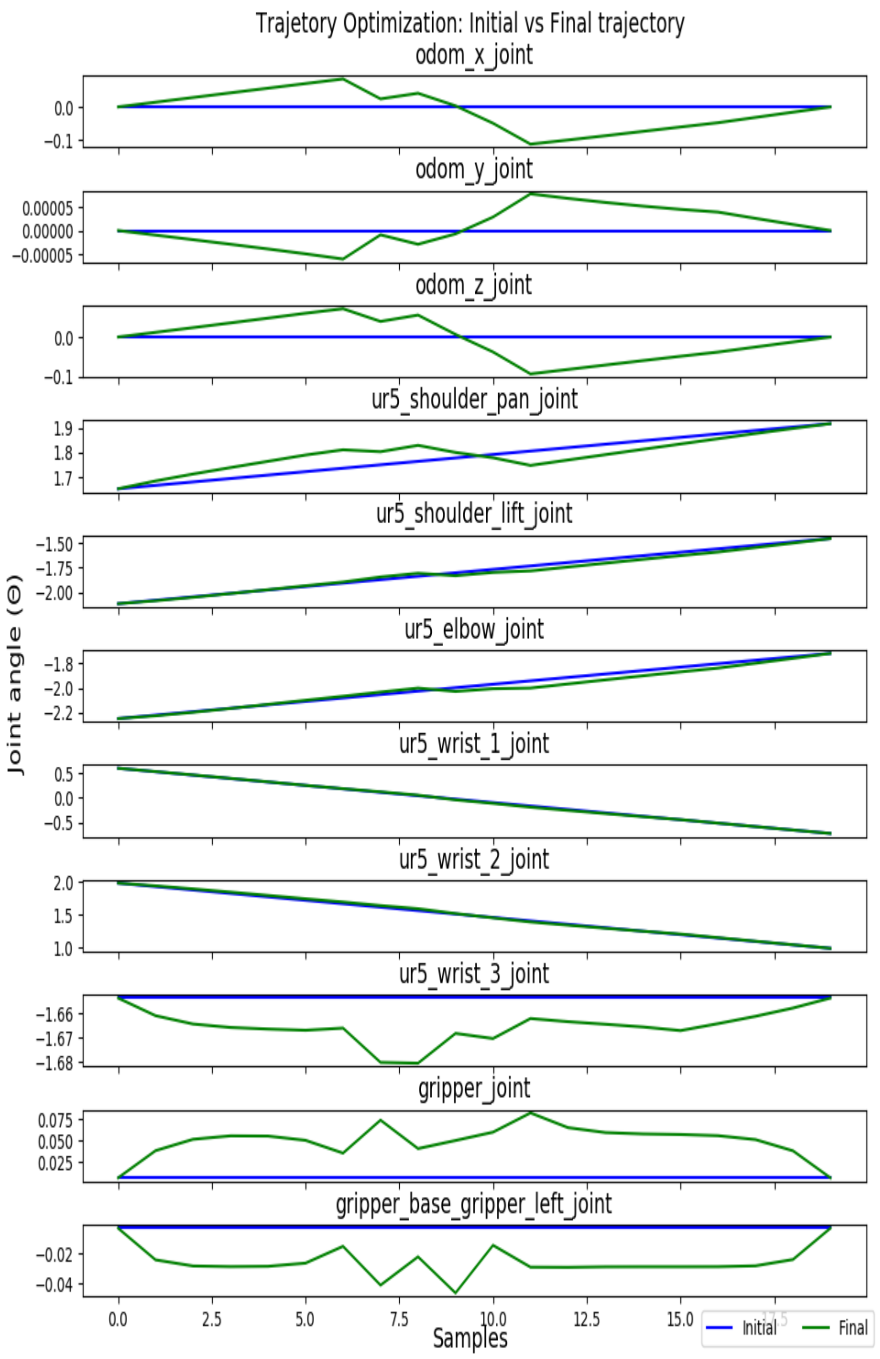


Figure 4.5: A sample planned trajectory of an 11 DOF Donbot: whole body

random number of obstacles placed at random locations and with a constant SQP parameters for all the three cases: a 7 DOF Kuka arm, a 6 DOF arm of Donbot and an 11 DOF Donbot: whole body. The results for the same is shown in Figure 4.6, Figure 4.7 and Figure 4.8 respectively.

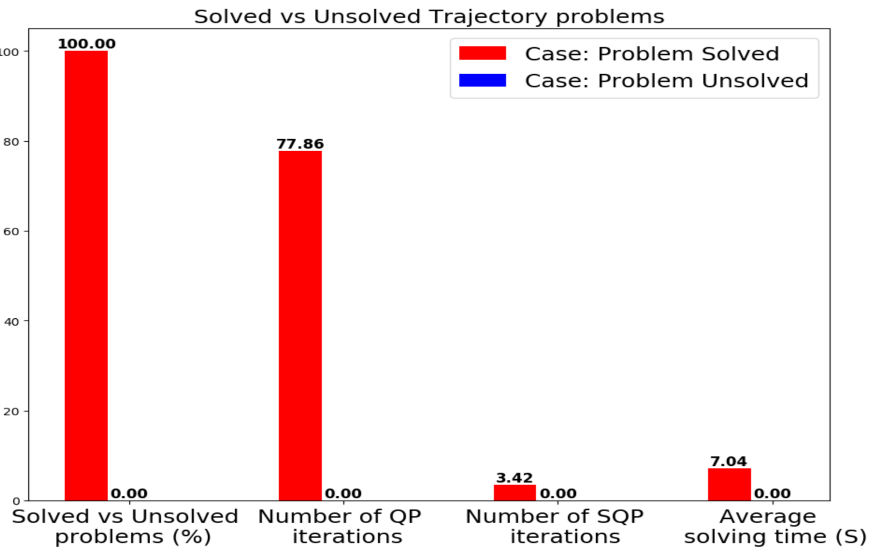


Figure 4.6: Reliability test for a 7 DOF Kuka arm

From Figure 4.6, Figure 4.7 and Figure 4.8, it can be seen that the trajectory optimization planner could generate a collision-free results in atleast 90% of the given random problems. The problem that couldn't be solved with just 6 DOF of Donbot got solved, when the trajectory is planned for the whole body. In the latter case, the additional 3 DOF helped to move the robot out of collision while reaching the target.

Also, with these random problems posed to the trajectory planner, it could solve the problem as fast as 3.76, 7.04 and 8.57 seconds for 6, 7 and 11 DOFs respectively.

From, Table 4.2, it can be seen that the average cost improvement for most of the problems are negative. This indicates the problem cost after optimization is greater than the initial problem cost from the initial guess. The initial guess to the problem is just a straight interpolated line from start to goal position, which could be reason for the initial lower problem cost. Despite the negative cost improvement, the optimized trajectory was free from collision in all the solvable cases.

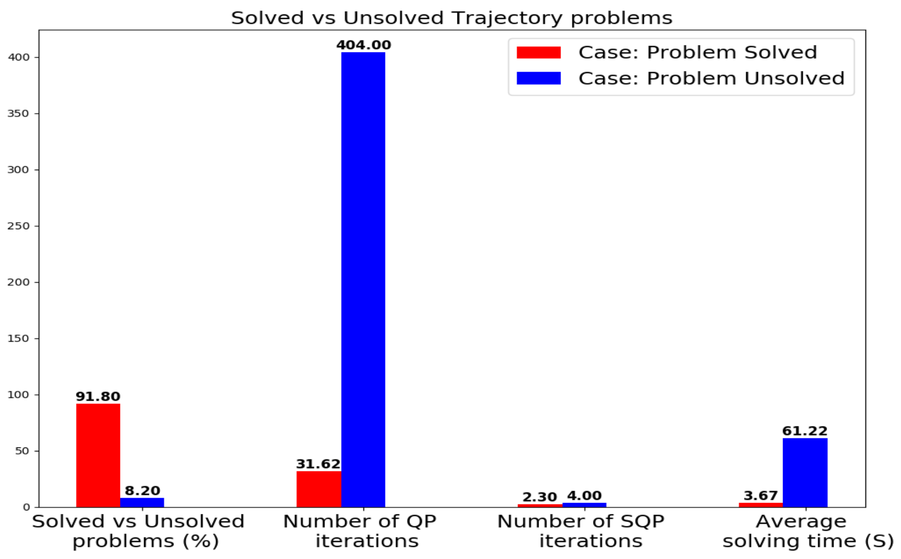


Figure 4.7: Reliability test for a 6 DOF arm of Donbot

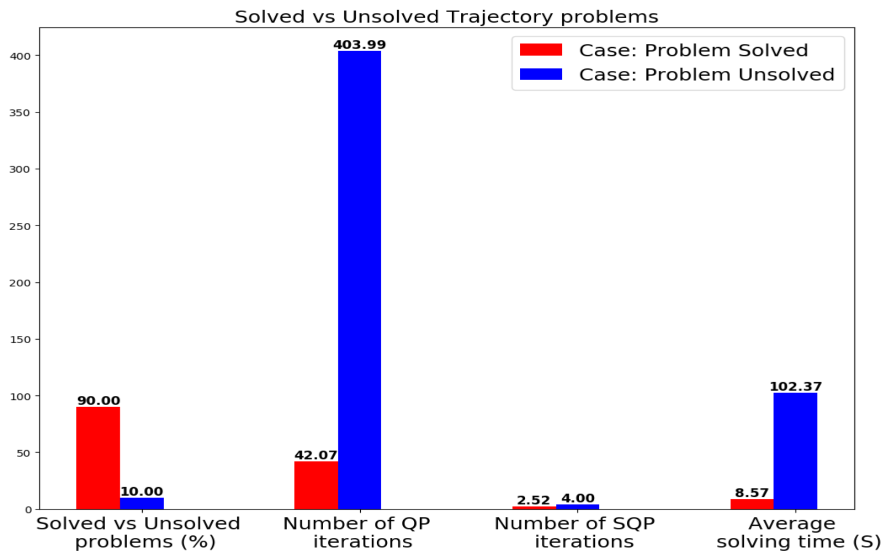


Figure 4.8: Reliability test for an 11 DOF Donbot: whole body

Table 4.2: Reliability test

Number of samples	Average total sovling time in s	Average planning time in s	Average collision check time in s	Average modelling time in s	Average Cost Improvement
6	0.183	0.132	0.05	0.00085	-13.205
8	0.086	0.054	0.032	0.00089	0
9	0.389	0.286	0.103	0.00084	-29.565
10	0.874	0.66	0.214	0.00091	-17.096
11	2.15	1.665	0.485	0.00096	-8.804
13	0.544	0.411	0.132	0.00095	18.289
14	1.444	1.008	0.435	0.00095	-1.108
15	3.488	2.501	0.987	0.00101	-33.377
16	4.605	3.349	1.254	0.00103	-0.134
17	1.02	0.724	0.295	0.00113	53.922
18	1.615	1.17	0.443	0.0012	28.355
19	1.553	1.147	0.405	0.00122	44.38
20	2.055	1.536	0.517	0.00116	-617.111
21	3.137	2.084	1.052	0.00126	8.128
22	7.826	5.874	1.951	0.00129	69.172
23	1.307	0.974	0.331	0.0016	-22.741
24	7.614	5.854	1.759	0.00126	-289.877
25	8.788	6.822	1.965	0.00133	47.344
26	3.437	2.691	0.745	0.00149	39.435
27	7.101	5.627	1.473	0.0015	-56.489
28	13.599	10.373	3.225	0.00149	-1013.8
29	17.517	14.229	3.286	0.0016	-345.484
30	7.625	6.237	1.385	0.00255	38.501
31	1.658	1.327	0.33	0.00173	-7.322
32	8.153	6.691	1.459	0.00217	23.172
33	4.739	3.93	0.808	0.00184	9.63
34	15.089	12.789	2.297	0.00194	25.94
35	5.103	4.143	0.958	0.00174	68.738

Consistency

To test the consistency of the trajectory optimization planner, the above said three cases were run with the same start, goal position and SQP parameters and result of the same is shown in Figure 4.9, Figure 4.10 and Figure 4.11 respectively, where a straight line can be observed for the solving time, SQP and Quadratic Programming (QP) iterations. Also, the consistency of the trajectory planner giving same trajectory results can be inferred from the Figure 4.12, Figure 4.13 and Figure 4.14, as all they are overlapping perfectly with each other.

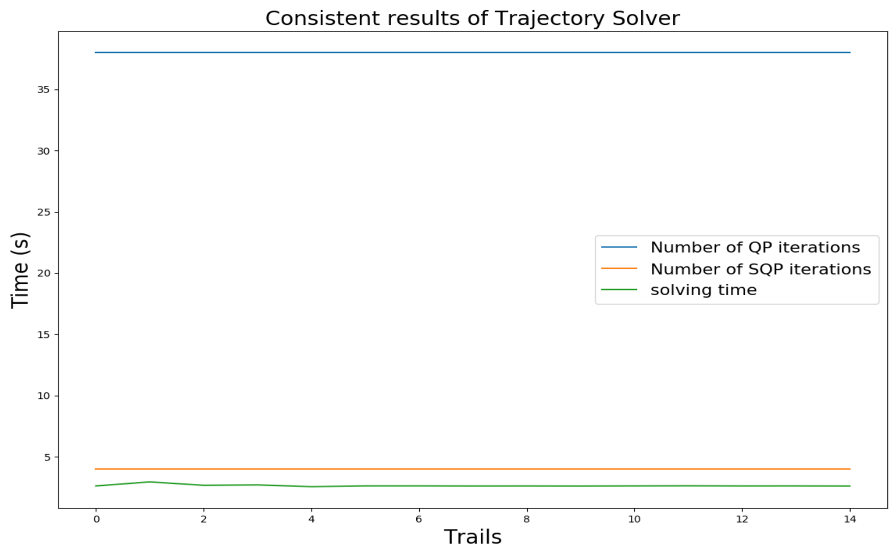


Figure 4.9: Consistency test for a 7 DOF Kuka arm: moving from same start to goal position

Reachability

From Figure 4.6, Figure 4.7 and Figure 4.8, it can also be observed that the trajectory planner tries to solve the given problem for longer time (around 100 seconds for the worst case). This is not desirable, as the planner failed to report to the user, that the problem couldn't be solved in less time.

Sensitivity

In order to test how the trajectory optimization planner behaves for the different SQP solver parameter, a few tests ran for the above said cases with random initial trust

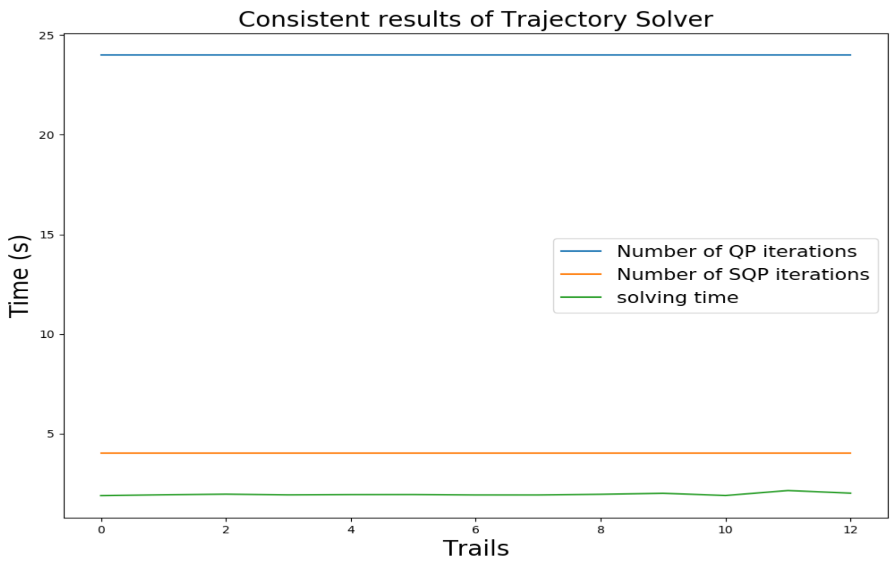


Figure 4.10: Consistency test for a 6 DOF arm of Donbot: moving from same start to goal position

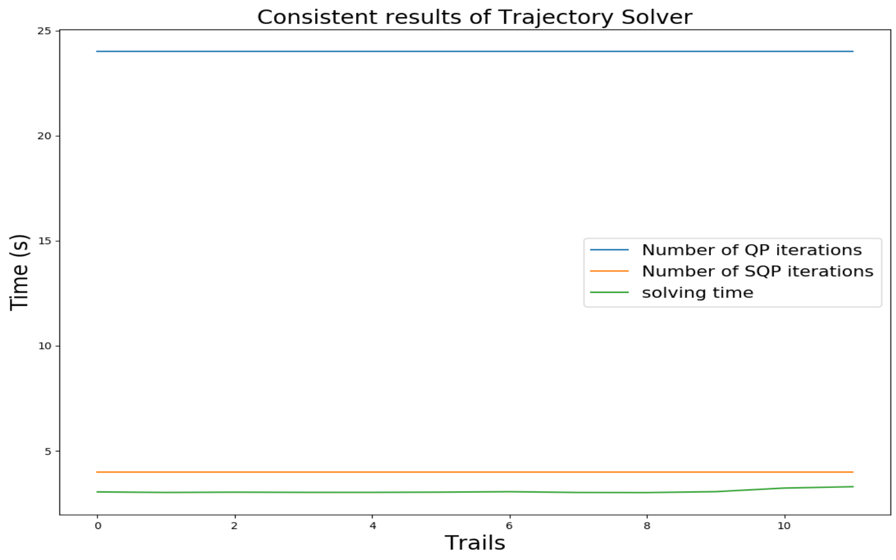


Figure 4.11: Consistency test for an 11 DOF Donbot (whole body): moving from same start to goal position

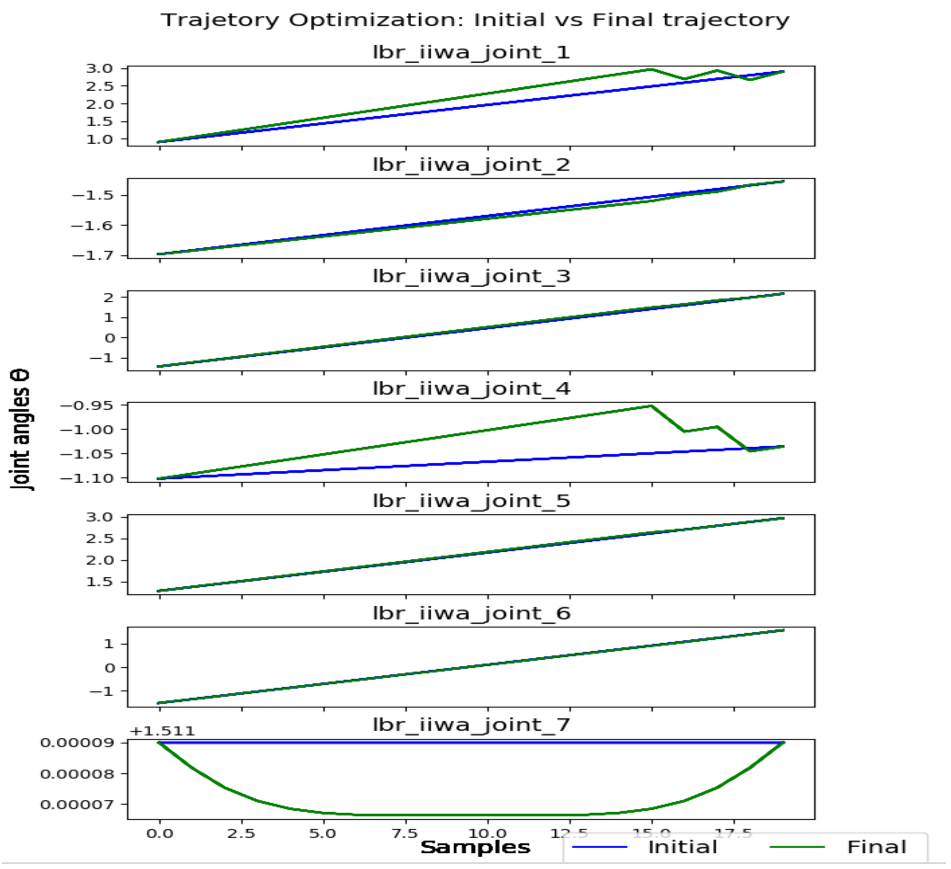


Figure 4.12: Consistency test for a 7 DOF Kuka arm: all trials of the trajectory perfectly overlapped with each other

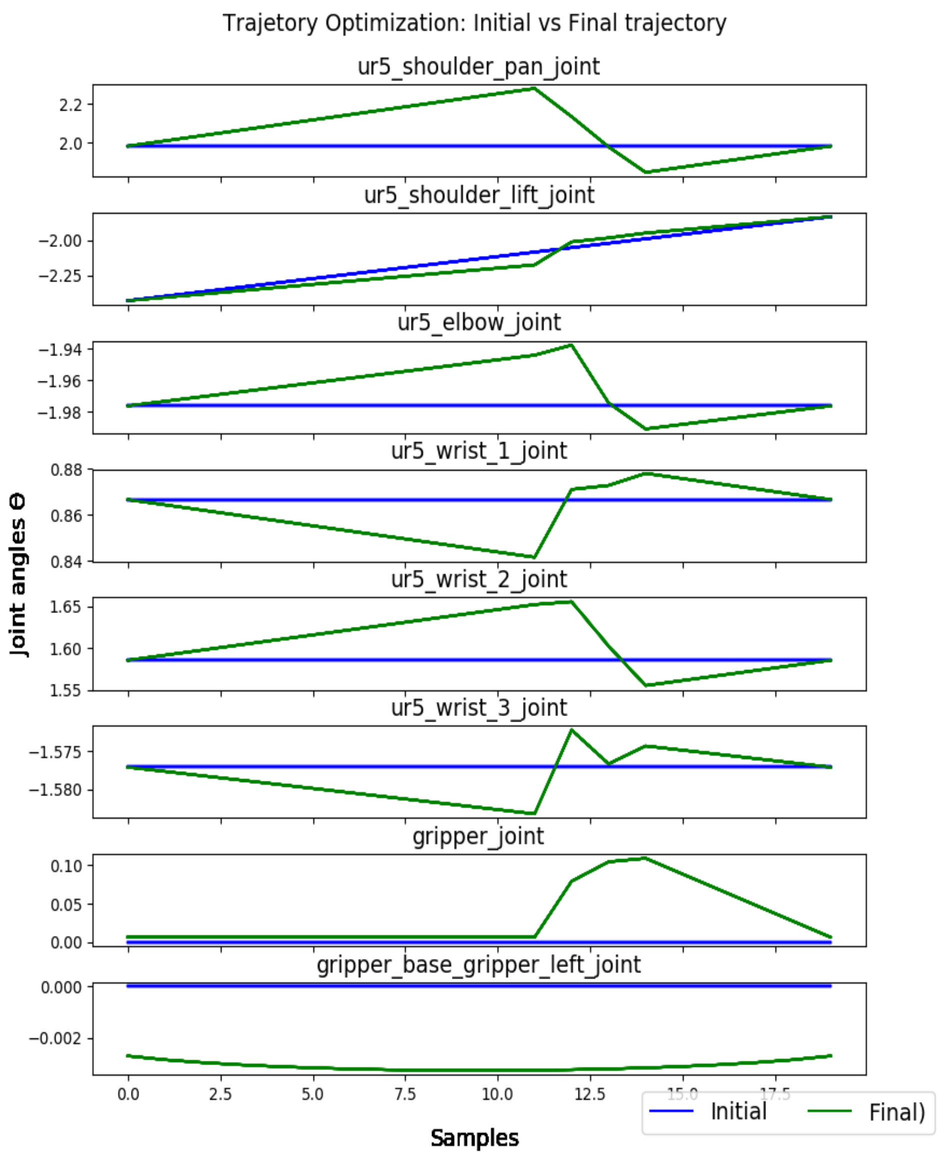


Figure 4.13: Consistency test for a 6 DOF arm of Donbot: all trials of the trajectory perfectly overlapped with each other

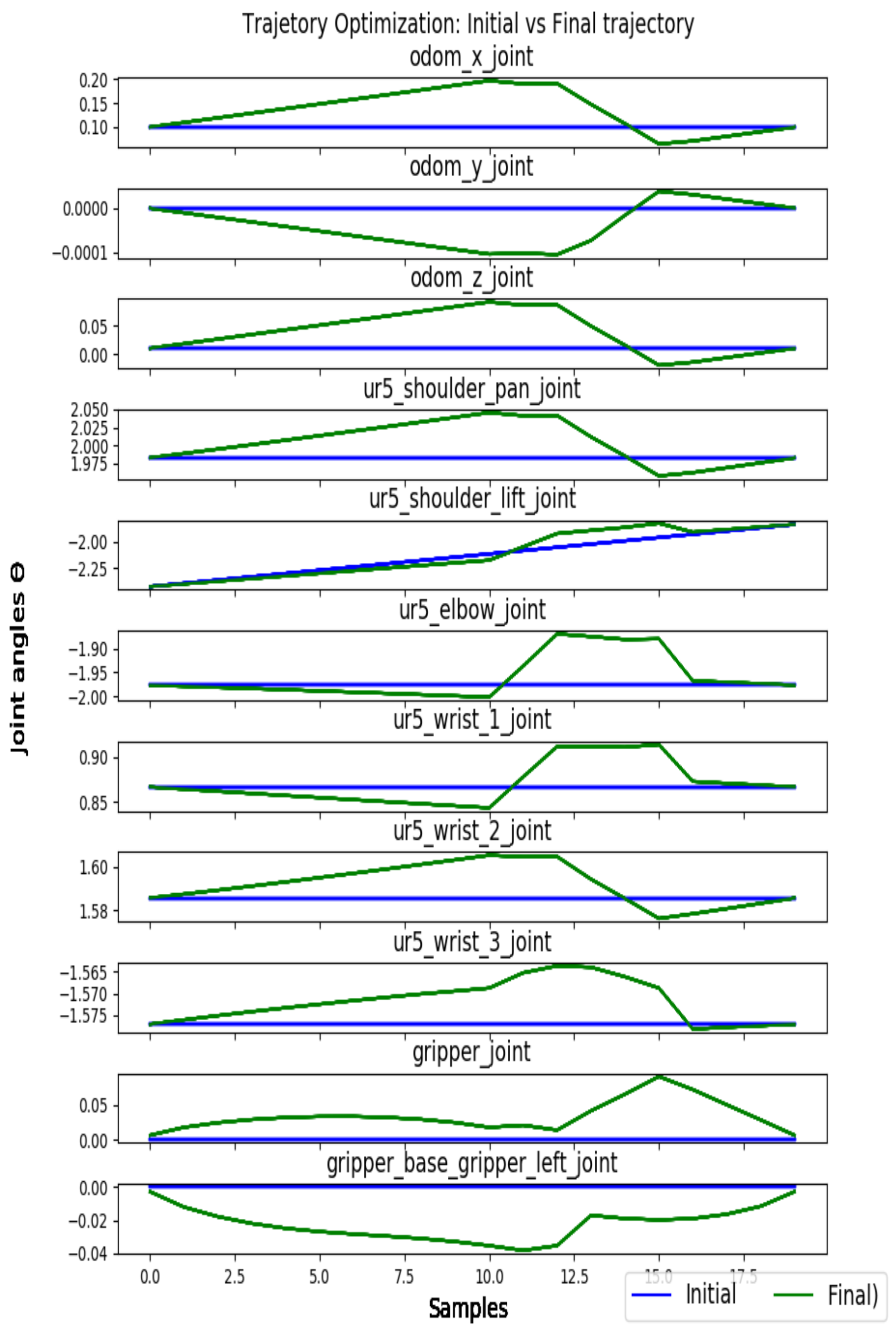


Figure 4.14: Consistency test for an 11 DOF Donbot (whole body): all trials of the trajectory perfectly overlapped with each other

region sizes, random number of samples and random trust region sizes. The result for the same is shown in Figure 4.15, Figure 4.16 and Figure 4.17. Also, additionally, these tests were ran with l_1 and l_2 penalty norm, to see how solver performance varies with respect to penalty norms as shown in Figure 4.18.

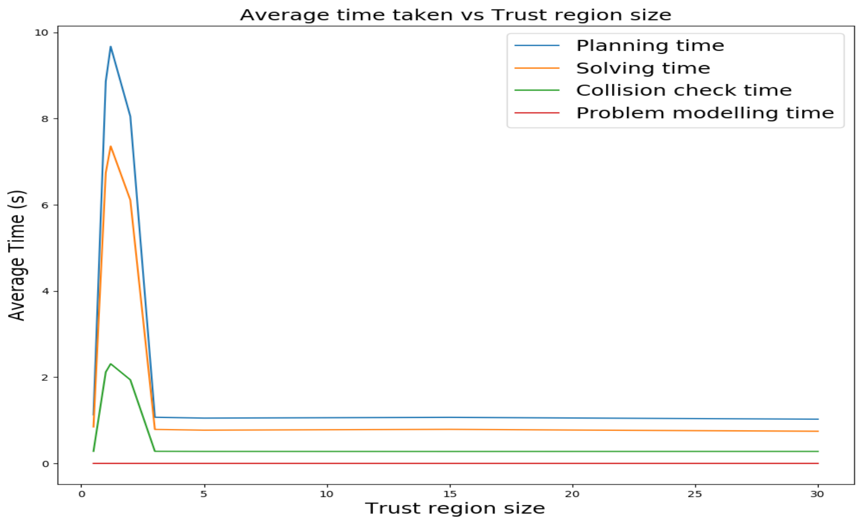


Figure 4.15: Adaptability test: Trust region vs Average time taken to solve the trajectory planning problem

From Figure 4.15, it can be observed that the problem solving time decreases with the increase in trust region size and the solving time stays constant when the trust region size increased beyond a certain threshold value. From Figure 4.16, similar results can be observed with the number of SQP and QP iteration taken to solve the problem with respect to the increase in trust region size. Figure 4.17 shows, average time taken to solve the trajectory planning problem with respect to different number of samples. Also, the Figure 4.18 shows a comparison for the time taken to solve the trajectory planning problem between l_1 and l_2 penalty norms.

Comparison between the old SQP and adapted SQP solver

The aforesaid three cases were again ran with random number of obstacles, random start and goal position to compare the behaviour performance of the old re-implemented SQP solver from [1] and the new adapted SQP solver and the corresponding results are shown in Figure 4.19 and Figure 4.20

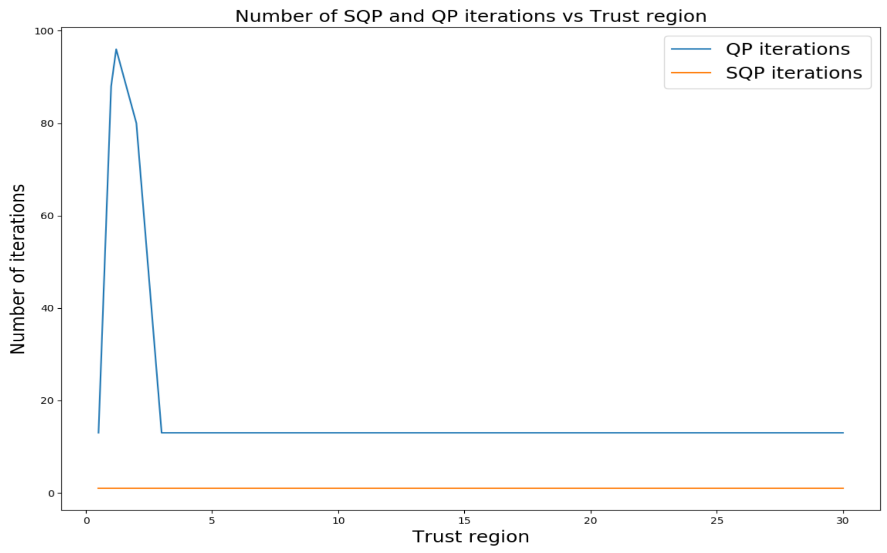


Figure 4.16: Adaptability test: Trust region vs number of QP and SQP iterations taken to solve the trajectory planning problem

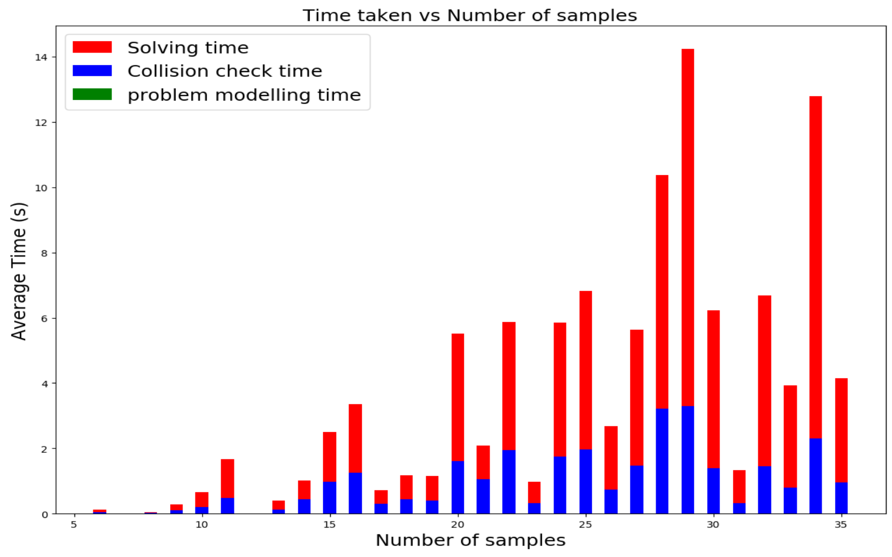


Figure 4.17: Adaptability test: number of samples vs average time taken to solve the trajectory planning problem

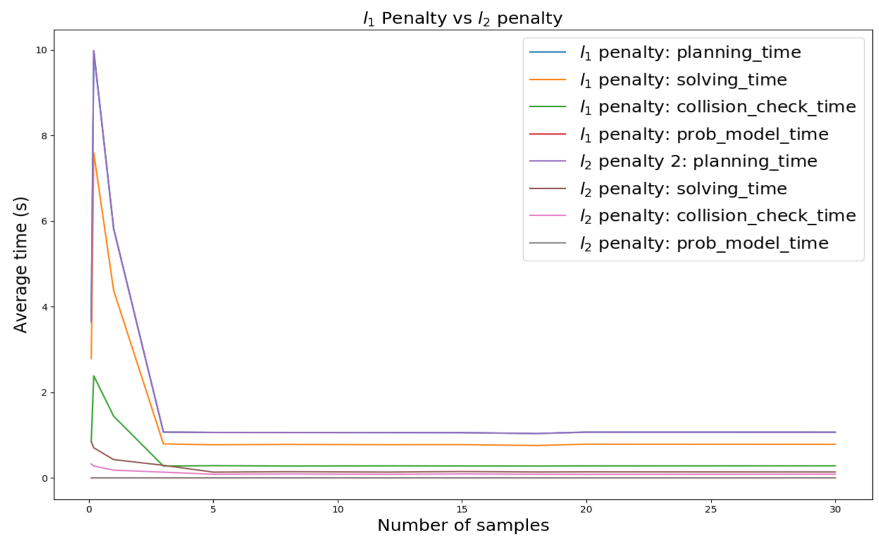


Figure 4.18: Adaptability test: l_1 vs l_2 penalty norm: number of samples vs average time taken to solve the trajectory planning problem

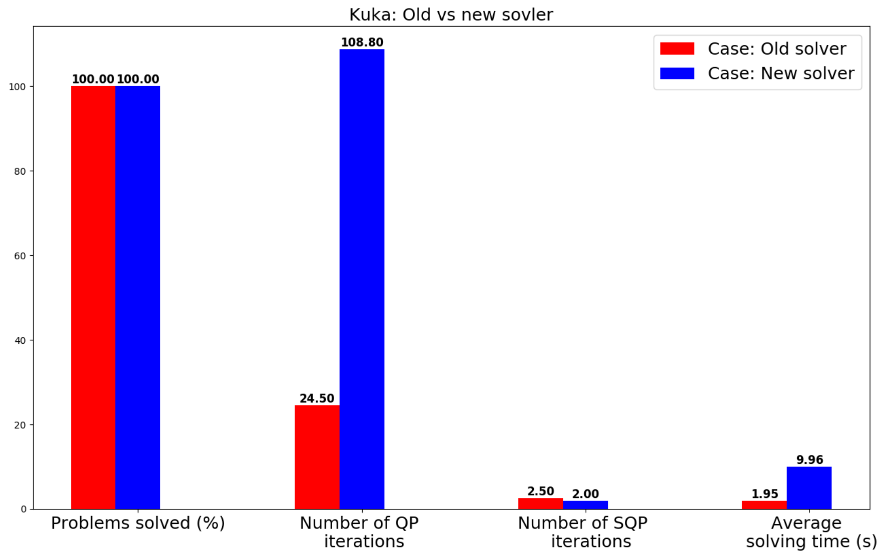


Figure 4.19: Comparison test: Kuka arm - Old vs New adapted SQP

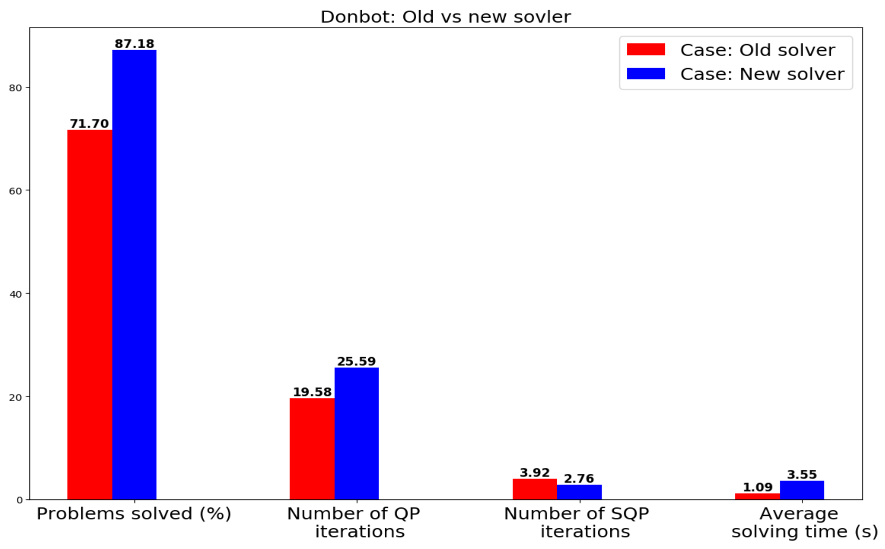


Figure 4.20: Comparison test: Donbot robot - Old vs New adapted SQP

For a simpler case of Kukka arm having 7 DOF arm, both the solvers could solve 100 % of the problem given to it as shown in Figure 4.19. Also, it can been seen that the new adapted solver takes more iterations and time to solve the given trajectory planning problem. But in a complex case of Donbot robot having 11 DOF, the new adapted solver has better reliability ratio and takes less number of SQP iterations to solve the given problem, as shown in Figure 4.20. Also, it can be seen that the new adapted algorithm takes little more time to solve the problem, because it need more iterations to solve the problem that couldn't be possible with the re-implemented old SQP solver.

4.3 Discussion

The results from the former sections shows that the new adapted algorithm is little slower than the original re-implemented algorithm. This slowness can be attributed to the better reliable results from the new algorithm, as it takes little time and iterations to find solution for the unsolvable problem by the re-implemented old algorithm. Moving collision constraints to the objective function helped the new solver to solve the unsolvable problem of the re-implemented old solver.

In the case of unsolvable problem, the algorithm takes much time and iterations before it gets terminated. This is not desirable and hence the algorithm could be further optimized to terminate with a reason after some specified time.

The newly adapted algorithm was test with random collision obstacles and robot configuration to show the performance, reliability, consistency, sensitivity and reachability of the solver for different parameters. These results shows a smooth trajectory resulted by trajectory planner even with the initial guesses having collisions.

The trajectory planner produces consistent results even in case of unsolvable situations by the original re-implemented SQP solver. The addition of collision constraint to the objective function made the adapted trajectory problem easier to solve, as there is only trust region constraint for the adapted problem

The SQP solver parameters can bring drastic change in the behaviour of the trajectory planner is also shown and also the speed of the trajectory planner can also be improved by using l_2 penalty norm instead of l_1 penalty norm.

In all of the test conducted, the final problem cost was always greater than the initial problem cost and the reason could be because of the bad initial guess having collisions.

In a very few cases, there is was a collision between the Donbot robot's end-effector and the obstacles found during the execution of an optimized trajectory, but the collision checker doesn't report this collision. This collisions might be occurred, because the collision cost constraints only includes the moveable links to be planned for trajectory and the fixed links constraints were not included. It would be interesting to see, how these costs can be included into the problem without affecting the SQP solver to plan for the requested trajectory. Also, since there was an extra collision pair ignored other than the default collision pairs, it would also be interesting to see the behaviour of the algorithm for the complex meshes.

The adapted trajectory optimization algorithm strongly relies on the SQP solver parameters and initial guess. Hence, choosing proper set of SQP solver parameters is more important to have a better results. Also, it will be interesting to know the behaviour of the algorithm, if the initial guess is given from the recorded movement of human hand on similar pick and place situations.

5 Conclusion

In this thesis, the trajectory planning problem has been reimplemented using SQP solving technique from [1]. Further the algorithm is adapted to formulate the collision constraints as l_1 penalty term and added to the objective cost function. Also, the collision constraints cost formulated to consider time continuous robot's self collision. Finally, a GUI application has been developed to tune the SQP solver parameters and to interact with trajectory planner.

The re-implemented and adapted algorithm was evaluated on different random scenarios to check the behaviours: run-time performance, reliability, consistency, sensitivity and l_1 vs l_2 penalty norms. Also, the adapted solver is compared with the re-implemented original solver for the run-time performance and the case where the new adapted solver outperforms the re-implemented original solver.

The evaluation results showed that the new adapted trajectory planning algorithm can solve good ratio of previously unsolvable problems by a re-implemented algorithm with an extra computational cost

6 References

[1] John Schulman et al. “Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization.” In: *Robotics: science and systems*. Vol. 9. 1. 2013, pp. 1–10.

[2] IN LOGISTICS ROBOTICS. “A DPDHL perspective on implications and use cases for the logistics industry”. In: *DHL March* (2016).

[3] Erwin Praßler and T Kaempke. “Mobile robots in office logistics”. In: *PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ROBOTS*. Vol. 27. INTERNATIONAL FEDERATION OF ROBOTICS, & ROBOTIC INDUSTRIES. 1996, pp. 153–160.

[4] Michael Rüßmann et al. “Industry 4.0: The future of productivity and growth in manufacturing industries”. In: *Boston Consulting Group* 9 (2015).

[5] Michaela Sprenger and Tobias Mettler. “Service Robots”. In: *Business & Information Systems Engineering* 57.4 (2015), pp. 271–274.

[6] *Roomba*. URL: `http://www.irobot.de/`.

[7] Jodi Forlizzi and Carl DiSalvo. “Service robots in the domestic environment: a study of the roomba vacuum in the home”. In: *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM. 2006, pp. 258–265.

[8] Thomas W Gruen, Daniel Corsten, and Sundrr Bharadwaj. “Retail out of stocks: A worldwide examination of causes, rates, and consumer responses”. In: *Grocery Manufacturers of America, Washington, DC* 1 (2002).

[9] Hai Che, Jack Chen, and Yuxin Chen. *Investigating effects of out-of-stock on consumer SKU choice*. 2011.

[10] *Stockout*. URL: `https://en.wikipedia.org/wiki/Stockout`.

[11] *Institute for Artificial Intelligence*. URL: `http://ai.uni-bremen.de/`.

[12] *Robotics Enabling Fully-Integrated Logistics Lines for Supermarkets — REFILLS*. URL: `http://www.refills-project.eu/`.

[13] Mrinal Kalakrishnan et al. “STOMP: Stochastic trajectory optimization for motion planning”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 4569–4574.

[14] *A re-implemented trajectory optimization planner from John Schulman*. URL: https://github.com/k-maheshkumar/trajopt_reimpl.

[15] Dr.-Ing. Danijela Ristic-Durrant. “ROBOTICS I lecture notes, University of Bremen”. 2015.

[16] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1994. ISBN: 0849379814.

[17] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and control of robot manipulators*. Springer Science & Business Media, 2012.

[18] J.J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley series in electrical and computer engineering: control engineering. Pearson/Prentice Hall, 2005. ISBN: 9780201543612. URL: https://books.google.de/books?id=MqMeAQAAIAAJ.

[19] *Kinematics of Robot Manipulator*. slide no: 69. Slideplayer. URL: http://slideplayer.com/slide/7768574/.

[20] Serdar Kucuk and Zafer Bingul. “Robot kinematics: Forward and inverse kinematics”. In: *Industrial Robotics: Theory, Modelling and Control*. InTech, 2006.

[21] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN: 354023957X.

[22] Serdar Kucuk and Z Bingul. *The inverse kinematics solutions of industrial robot manipulators*. July 2004, pp. 274–279. ISBN: 0-7803-8599-3.

[23] Sébastien Briot, Wisama Khalil, et al. *Dynamics of Parallel Robots*. Springer, 2015.

[24] *Robot kinematics*. Wikipedia. URL: https://en.wikipedia.org/wiki/Robot_kinematics.

[25] *XML Robot Description Format (URDF)*. URL: https://wiki.ros.org/urdf/XML/model.

[26] *Semantic Robot Description Format (SRDF)*. URL: https://wiki.ros.org/srdf.

[27] S. M. LaValle. *Planning Algorithms*. Available at http://planning.cs.uiuc.edu/. Cambridge, U.K.: Cambridge University Press, 2006.

[28] Lucia Pallottino. *Introduction to Motion Planning*. Centropiaggio. URL: www.centropiaggio.unipi.it/sites/default/files/course/material/srd_cap4_mp_0.pdf.

[29] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

[30] *Wikipedia*. https://en.wikipedia.org/wiki/Trajectory_optimization.

[31] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787.

[32] *Wikipedia*. https://en.wikipedia.org/wiki/Hessian_matrix.

[33] *Wikipedia*. https://en.wikipedia.org/wiki/Positive-definite_matrix.

[34] D. Berenson et al. "Manipulation planning with Workspace Goal Regions". In: *2009 IEEE International Conference on Robotics and Automation*. May 2009, pp. 618–624. DOI: 10.1109/ROBOT.2009.5152401.

[35] James Kuffner et al. "Motion planning for humanoid robots". In: *Robotics Research. The Eleventh International Symposium*. Springer. 2005, pp. 365–374.

[36] Radu Bogdan Rusu et al. "Real-time perception-guided motion planning for a personal robot". In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE. 2009, pp. 4245–4252.

[37] Ioan A Şucan, Mrinal Kalakrishnan, and Sachin Chitta. "Combining planning techniques for manipulation using realtime perception". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 2895–2901.

[38] Abraham Sánchez López, René Zapata, and Maria A Osorio Lama. "Sampling-based motion planning: A survey". In: *Computación y Sistemas* 12.1 (2008), pp. 5–24.

[39] M. Elbanhawi and M. Simic. "Sampling-Based Robot Motion Planning: A Review". In: *IEEE Access* 2 (2014), pp. 56–77. DOI: 10.1109/ACCESS.2014.2302442.

[40] Kris Hauser and Victor Ng-Thow-Hing. "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 2493–2498.

[41] Zeeshan Shareef and Jochen Steil. "Trajectory optimization of COmpliant HuMANoid (COMAN) robot arm using path parameter based dynamic programming". In: *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE. 2016, pp. 705–710.

[42] Mohamed El Amine Boudjellel and Taha Chettibi. “Optimal trajectory planning for a mobile robot in presence of obstacles using multi-objective optimization techniques”. In: *Modelling, Identification and Control (ICMIC), 2016 8th International Conference on*. IEEE. 2016, pp. 509–514.

[43] Florent Boithias, Mohamed El Mankibi, and Pierre Michel. “GENERIC MULTI-OBJECTIVE OPTIMIZATION METHOD OF INDOOR AND ENVELOPE SYSTEMS’ CONTROL”. In: *University” Politehnica” of Bucharest Scientific Bulletin, Series C: Electrical Engineering* 74.1 (2012), pp. 57–66.

[44] Helen Oleynikova et al. “Continuous-time trajectory optimization for online UAV replanning”. In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE. 2016, pp. 5332–5339.

[45] Sergey Alatartsev et al. “Robot trajectory optimization for the relaxed end-effector path”. In: *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on*. Vol. 1. IEEE. 2014, pp. 385–390.

[46] M. Gao, P. Ding, and Y. Yang. “Time-Optimal Trajectory Planning of Industrial Robots Based on Particle Swarm Optimization”. In: *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*. Sept. 2015, pp. 1934–1939. DOI: 10.1109/IMCCC.2015.410.

[47] Zhijie Zhu, Edward Schmerling, and Marco Pavone. “A convex optimization approach to smooth trajectories for motion planning with car-like robots”. In: *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*. IEEE. 2015, pp. 835–842.

[48] Marc Toussaint. “Robot trajectory optimization using approximate inference”. In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 1049–1056.

[49] Marc Toussaint. “Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning.” In: *IJCAI*. 2015, pp. 1930–1936.

[50] Nathan Ratliff et al. “CHOMP: Gradient optimization techniques for efficient motion planning”. In: *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*. IEEE. 2009, pp. 489–494.

[51] Christian Gehring et al. “Control of dynamic gaits for a quadrupedal robot”. In: *Robotics and automation (ICRA), 2013 IEEE international conference on*. IEEE. 2013, pp. 3287–3292.

[52] Farhad Aghili. “A unified approach for inverse and direct dynamics of constrained multibody systems based on linear projection operator: applications to control and simulation”. In: *IEEE Transactions on Robotics* 21.5 (2005), pp. 834–849.

[53] Mohamed Elbanhawi and Milan Simic. “Sampling-based robot motion planning: A review”. In: *Ieee access* 2 (2014), pp. 56–77.

[54] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. “A fast procedure for computing the distance between complex objects in three-dimensional space”. In: *IEEE Journal on Robotics and Automation* 4.2 (Apr. 1988), pp. 193–203. ISSN: 0882-4967. DOI: `10.1109/56.2083`.

[55] Gino Van den Bergen. “A Fast and Robust GJK Implementation for Collision Detection of Convex Objects”. In: *J. Graph. Tools* 4.2 (Mar. 1999), pp. 7–25. ISSN: 1086-7651. DOI: `10.1080/10867651.1999.10487502`. URL: `http://dx.doi.org/10.1080/10867651.1999.10487502`.

[56] Patrick Lindemann. “The gilbert-johnson-keerthi distance algorithm”. In: *Algorithms in Media Informatics* (2009).

[57] Jeff Linahan. “A Geometric Interpretation of the Boolean Gilbert-Johnson-Keerthi Algorithm”. In: *CoRR* abs/1505.07873 (2015). arXiv: `1505.07873`. URL: `http://arxiv.org/abs/1505.07873`.

[58] Gino Van Den Bergen. “Proximity queries and penetration depth computation on 3d game objects”. In: *Game developers conference*. Vol. 170. 2001.

[59] *Standalone URDF parser for Python*. URL: `https://github.com/ros/urdf_parser_py`.

[60] *Python*. URL: `https://www.python.org/`.

[61] *NumPy*. URL: `http://www.numpy.org/`.

[62] *Orocos Kinematics and Dynamics*. URL: `http://www.orocos.org/kdl`.

[63] Steven Diamond and Stephen Boyd. “CVXPY: A Python-Embedded Modeling Language for Convex Optimization”. In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[64] A. Domahidi, E. Chu, and S. Boyd. “ECOS: An SOCP solver for embedded systems”. In: *European Control Conference (ECC)*. 2013, pp. 3071–3076.

[65] B. O’Donoghue et al. “Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding”. In: *Journal of Optimization Theory and Applications* 169.3 (June 2016), pp. 1042–1068. URL: `http://stanford.edu/~boyd/papers/scs.html`.

[66] J. Dahl M. S. Andersen and L. Vandenberghe. *CVXOPT: A Python package for convex optimization, version 1.1.6*. online. cvxopt.org, 2013.

[67] Erwin Coumans. "Bullet Physics Simulation". In: *ACM SIGGRAPH 2015 Courses*. SIGGRAPH '15. Los Angeles, California: ACM, 2015. ISBN: 978-1-4503-3634-5. DOI: `10.1145/2776880.2792704`. URL: `http://doi.acm.org/10.1145/2776880.2792704`.

[68] Erwin Coumans. *Bullet 2.83 Physics SDK Manual*. 2015. URL: `https://github.com/bulletphysics/bullet3/blob/master/docs/Bullet_User_Manual.pdf`.

[69] *SRDF parser for python*. URL: `https://github.com/ros-planning/srdfdom`.

[70] *Extended version of Bullet Physics SDK for continuous collision detection*. URL: `https://github.com/k-maheshkumar/bullet3`.

[71] *Qt for Python*. URL: `http://wiki.qt.io/Qt_for_Python`.

[72] *Canonical source repository for PyYAML*. URL: `https://github.com/yaml/pyyaml`.

[73] *LBR iiwa*. URL: `https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa`.

[74] *UR5 - A HIGHLY FLEXIBLE ROBOT ARM*. URL: `https://www.universal-robots.com/products/ur5-robot/`.