Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische Friedrich-Wilhelms-Universität Bonn

Master thesis

Efficient Stochastic Multicriteria Arm Trajectory Optimization

Author:
Dmytro Pavlichenko

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Maren Bennewitz

Advisors:
Prof. Dr. Sven Behnke

Diego Rodriguez

Submitted: December 7, 2016

Declaration of Authorship

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Abstract

Autonomous robots perform a vast range of tasks in different environments which vary from household to aerospace. In most cases they should interact with the environment using articulated human-like arms. That is why there is a necessity in algorithms which are able to effectively plan feasible trajectories for manipulation. However, as manipulators have many Degrees Of Freedom (DOF), trivial planning algorithms such as A* are not able to plan in reasonable amount of time due to the curse of dimensionality. Moreover, often except of collision avoidance, there is a necessity to account for orientation constraints, torque minimization and trajectory duration. All together, these factors make planning of manipulation trajectories a challenging problem.

In this thesis an approach to optimize the manipulation trajectories is presented. The optimization is performed with respect to the following criteria: collision avoidance, preserving joint limits and any additional orientation and/or position constraints, trajectory duration and torque minimization. This approach is based on Stochastic Trajectory Optimization for Motion Planning (STOMP) method by Kalakrishnan et al. [1]. Our modifications serve to decrease the overall runtime of the algorithm and include additional criteria, such as trajectory duration. We present our own cost function which is designed in order to enable flexible tuning of trajectory properties for each particular planning task.

We evaluate our method in simulation using tasks of different nature and difficulty. We measure the average runtime, success rate, trajectory length and compare our approach with other planning algorithms: RRTConnect, LBKPIECE and STOMP-Industrial. We demonstrate that our approach shows better runtime and comparable high success rate. We analyse the effects of torque and duration optimization as well.

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

1.1 Motivation

The area of autonomous robotics improves and develops from year to year. More complex and agile robots are being designed with an aim to improve current capabilities of robots and to solve previously unapproachable tasks. Autonomous robots are being applied into vast range of environments: household, industrial, outdoors, space, etc. Majority of these robots are designed to interact with their surroundings by means of robotic arms. This makes an arm manipulation planning problem one of the essential tasks for this type of robots.

Therefore, an arm manipulation planning has been investigated by many researchers. Even though this extensive research has created lots of different methods, the problem can not be considered as solved. This is caused by the curse of dimensionality and by overall complexity of the task. Usually a robotic arm has at least 6 Degrees Of Freedom (DOF) and must precisely act in unstructured environments. Complex geometric shape of the arm complicates the collision checking, meanwhile multiple additional criteria, such as: joint limits, orientation constraints and torque minimization produce solution spaces with many disjoint local minima. As planning of a manipulation trajectory usually is a part of more a complex task, it is desirable to obtain a trajectory as fast as possible. High success rate is required as well, as it allows to solve the task reliably and allows to avoid performing several attempts in order to solve the task, which prolongs the spent time. Altogether, these requirements and characteristic of the problem make the task of planning a feasible manipulation trajectory a challenging problem.

Furthermore, if a robot has additional joints, such as torso yaw or pitch, it may be feasible to involve them into the planning process in order to obtain a possibility to solve wider range of tasks. However, adding more joints increases the amount of dimensions and makes the planning more computationally expensive. In addition, manipulation of certain objects imposes orientation constraints to the end-effector. For instance: a glass with liquid which can not be spilled. Moreover, an object may have a significant mass, which can not be neglected. This creates a necessity to account for the torques as well. Besides that, the environment around the robot may change while the planned motion is being executed. This situation requires

immediate replanning in order to adapt the movement to the new state of the environment. In order to accomplish this task it is necessary to have a planner which can find a feasible solution very fast.

It is possible to see from this brief overview that there are many challenges to overcome. In this work we present an approach which addresses some of them. The problem we aimed to solve is defined in more details in the next section.

1.2 Problem Definition

The task of motion planning can be formulated as finding a feasible path in unstructured environment, given a start configuration A and a goal configuration B. This path must be collision free in a way that both collisions with the environment and self-collisions are avoided. Any additional constraints, such as joint limits, orientation and/or position constraints, torque and velocity constraints, must be satisfied as well. Moreover, the path must be as smooth as possible since jerky movements may potentially cause a damage to the motors.

This work has been inspired by the CENTAURO project [2]. The purpose of this project is to design a centaur-like robot with four legs and human-like upper body with two arms. This robot is developed to act in disaster scenarios where human intervention is not possible due to high risk for human health and life. The CENTAURO robot is assumed to be teleoperated by the human operator during most part of the mission. However, there may arise situations where wireless connection has insufficient quality or is absent at all. In this case it is necessary to have a possibility to perform all tasks autonomously, including arm motion planning. Moreover, the manipulation planning is usually a part of a more complex task. Thus, it is profitably to reduce the load of the main operator by performing planning of certain manipulation tasks autonomously even in a presence of good connection.

A disaster scenario demands tasks to be executed as quick as possible due to potential danger to human lives. Thus, not only the planning itself should be performed fast, but the execution of obtained motions must allow to achieve desired configuration in a short amount of time. In a disaster scenario the robots often need to operate with heavy objects, such as different tools or fragments of destroyed structures. In this case the trajectories must also be optimal with respect to torques affecting the arm, as this prevents motors from being damaged and saves the battery power which raises the chances of mission being completed successfully.

In many cases a trajectory may be generated with a help of motion primitives. However, these trajectories usually are not optimal with regards to the current

specific situation. Thus, it is necessary to have a capability to optimize these trajectories to meet the requirements discussed above.

In summary, a trajectory should fulfill the following requirements:

- A trajectory must be able to move the arm to a given goal configuration.
- A trajectory must avoid any collisions with the environment or the robot itself. Joint limits have to be satisfied. Any additional orientation or position constraints have to be satisfied.
- Trajectories must be generated as quick as possible.
- Trajectories must be planned in a way that their execution time is as short as possible.
- Trajectories must be planned so that operations with heavy objects generate as less torque as possible.

1.3 Approach

As it was discussed previously, multiple criteria have to be considered in order to successfully solve a manipulation planning task. Moreover, the process of planning have to be performed as fast as possible. In order to employ an efficient multicriteria optimization we modify an existing algorithm: Stochastic Trajectory Optimization for Motion Planning (STOMP) [1]. The algorithm is capable of optimizing non-differentiable costs, and thus can be applied to deal with orientation constraints and torque minimization. In addition, STOMP has complexity linear in number of dimensions, which allows to solve tasks with 6 and more DOF in reasonable amount of time. We describe the algorithm in details all together with its advantages and disadvantages in chapter 4. Below we give a brief overview of the proposed modification.

In order to achieve decrease of runtime and ability to optimize for required criteria, we introduce other way of state cost computation. In the original STOMP a state cost of a trajectory is a sum of costs of each keyframe. Instead, we propose to compute state cost of a trajectory as a sum of costs of the transitions between consequent pairs of keyframes. This model allows to plan trajectories with much smaller number of keyframes. Moreover, this model fits continuous nature of trajectory better, as it considers consequent transitions instead of separate keyframes. This allows to account for the duration of these transitions, and hence, to optimize the overall duration of the trajectory.

Arm manipulation planning requires many complex computations to be done. In order to reduce the amount of computations for collision checking we improve a representation of a robot compared to idea of the original STOMP.

We also introduce our own cost function which target is to account for many criteria while having fast runtime. The cost function is designed in a way that each term has its importance weight. One may achieve trajectories which favour different behaviours by manipulating these weights. In order to optimize the duration of the trajectories, we add one more dimension to configuration space: a velocity for each transition from current to next keyframe.

These modifications allow to take into account additional criteria during optimization and to lower the runtime. Moreover, the execution time of trajectories itself can be optimized which gives overall improvement of runtime for the pipeline plan-execute.

1.4 Structure of Thesis

The thesis has the following structure:

- Chapter 2 provides a review of related work in a context of a trajectory planning for manipulation tasks. In this chapter researches, which seemed to be the most interesting to the author of this thesis are briefly described. The overview of existing manipulation trajectory planners allows to show how proposed method fits into the current state of the art.

- Chapter 3 describes several key prerequisites for the motion planning: kinematic chain, forward and inverse kinematics, world and configuration spaces.

- Chapter 4 gives a detailed description of the original STOMP algorithm, which is used as base for this work. The chapter is finalized with analysis of advantages and disadvantages of the algorithm.

- Chapter 5 presents the proposed modifications. First, the changed cost computation policy is discussed. Second, the proposed cost function is described in details. Brief overview of the implemented system concludes this chapter.

- Chapter 6 contains the criteria we used in order to evaluate the performance of proposed method. The performed experiments are described and their results are presented. A comparison of several algorithms against our method is shown as well.

- Chapter 7 is a summary of the thesis. First we summarise the contributions of this thesis, then we discuss known limitations, and, finally, we present possibilities for the future work.

2 Related Work

Arm manipulation planning is an essential task for autonomous robots. That is why a lot of research has been done in this field. However, manipulators have quite complex structure with respect to their geometry and numerous DOF. Moreover, manipulation tasks often require several constraints to be fulfilled. These characteristic make arm manipulation planning a challenging problem. Besides that, robots are getting more complex and are required to perform more complex tasks. Thus, this area demands further investigation from the side of a research community.

Many different approaches have been proposed in order to address the problem of manipulation planning. Sampling-based planners have achieved noticeable popularity and success [3] [4] [5] [6] [7]. Many approaches try to downscale the dimensionality of the problem in order to avoid exhaustive search due to curse of dimensionality. In this section we give a brief overview of the related work has been done in this area.

James J. Kuffner et al. [8] utilized Rapidly-Exploring Random Trees (RRTs) [9] in his work: RRT-Connect: An Efficient Approach to Single-Query Path Planning. The proposed approach is to incrementally grow two trees towards each other from the start and goal configuration respectively. A simple greedy heuristic is used in order to bias the trees towards exploration of large unseen areas as well as to grow towards each other. A solution is found when the trees are connected. This technique allows to find solutions even in complex cluttered environments with narrow passages of compound shape. Moreover, the approach is quite robust and does not require fine parameter tuning. The method can also be used for navigation planning. However, the obtained trajectories are not smooth and often contain many unnecessary movements. Thus the postprocessing is necessary which still not always provides a sufficiently smooth paths.

Another interesting approach is based on RRTs as well: Constrained Bi-directional Rapidly-Exploring Random Tree (CBiRRT) [10]. The main focus of this work is to solve planning problems which involve such constraints as: torque limits, orientation constraints, and constraints for following workspace surfaces. As an example, the authors use a task where robot should move a heavy dumbbell from one table to another. This task encourages the robot to slide the dumbbell along the

surface of the table as long as possible before lifting it in the air. However, a set of configurations which satisfy this constraint occupies a very small volume in the configuration space. Thus, it is very unlikely that direct sampling in configuration space will result in successfully found solution. This problem can not be solved efficiently with planners which sample the C-space directly. Instead, the authors propose to project the samples onto constraint manifolds. They utilize the same approach of growing two trees towards each other biased by the greedy heuristic. This allows to find a connections between complex constraint manifolds efficiently which results in feasible trajectories.

Ioan A. Sucan et al. proposed a different method - Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) [11]. The approach is based on a multiple-level grid discretization of the state space. The motivation for this work are systems which have complex dynamics described by physical models which creates a necessity for physics-based simulation. Even though simulation introduces higher computational costs, it allows to obtain trajectories of higher quality. This is because "physics simulators take into account more dynamic properties of the robot (such as gravity, friction) and constructing models of systems is easier and less error prone than deriving equations of motion" [11]. Moreover, the method does not require state sampling method and distance metric. This method takes an advantage of dimensionality reduction as well. Thus the state space Q is replaced by projection space $E(Q)$. The multiple-level discretization is used to estimate the coverage of the state space which allows to detect and explore large unexplored areas which potentially may contain a solution. Each next level has finer resolution than its parent. This structure allows to efficiently represent the tree of motions. A cell is considered to be exterior if it has less than $2n$ neighbours where n is the dimensionality of $E(Q)$. The exploration is biased towards exterior cells. A multiresolution representation of the state space allows to find a solution efficiently. However, KPIECE requires a postprocessing to obtain more smooth and short trajectory.

In work of Lydia E. Kavraki et al. [12] another method for planning in static environments is used: Probabilistic Roadmaps for Path Planning in Hihg-Dimensional Configuration Spaces. The method consists of two phases: learning and query. During the learning phase a probabilistic roadmap is constructed. It is represented as a graph where the nodes are collision free configurations, and the edges are a collision-free transitions between these configurations. These a transitions are generated by some local planner. In the second phase any required initial and goal configurations are connected to the roadmap. Than the local planner attempts to find a feasible plans to transition from initial and goal configurations to the closes nodes of the roadmap. This method was used in [13] and [14] as

well. The approach is well generalized and any planner can be used to plan local transitions. However, it fits well only environments which are completely static during the mission, as frequent reconstruction of the roadmap is a high-cost operation. The runtime of roadmap construction heavily depends on dimensionality and complexity of collision checking.

Another approach is to use an artificial potential field proposed by O. Khatib et al. [15]. The idea is that the environment is represented as a set of force sources. The goal has an attractive force, meanwhile the obstacles have repulsive forces. These force influence the end-effector, which makes end-effector attempt to smoothly approach the goal avoiding the obstacles at the same time. However, in a cluttered environment there is a high risk of being stuck in one of the numerous local minima. Nevertheless, this approach can be effectively used for real-time control.

A noticeable method is Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [16]. In contrast to previously reviewed sampling-based planners, in CHOMP the covariant gradient technique is used. The idea is similar to elastic bands planning [17] [18]. However, the input trajectory does not have to be collision-free. The algorithm quickly converges to the locally optimal trajectory due to use of gradient descend. The authors use signed distance field as environment representation, so that cells on the border of obstacles have 0 value, cells outside - positive and cells inside - negative values respectively. Such a representation allows to perform collision checking fast. At the same time negative distances inside the obstacles allows to obtain gradients even for non-collision-free parts of the trajectory. The experiments have shown that CHOMP produces smooth trajectories which do not need further processing and are ready to be executed on the real robot. Though, the costs used in this method must be differentiable which puts certain limitations on the available constraints. Moreover, as any gradient-based algorithm, CHOMP tends to get stuck in local minima.

In [19] the CHOMP algorithm is extended to support goal sets. In most cases it is possible to grasp an object in many different ways, the same applies to placing the object. A set of possible goal configurations is called a goal set. The resulting trajectory may vary significantly when different goal poses are chosen. Thus, supporting goal sets can lead to better solutions. It is achieved by employing a constrained optimization solution. They constraint the goal configuration to the goal set, in addition, they allow CHOMP to change the goal. The needed behaviour is achieved by these steps. However, it is noticeable, that in some situations the goal-set implementation provides worse results than original CHOMP. This is explained by the goal converged to the local minima.

Another extension of CHOMP is T-CHOMP [20]. The authors extend the con-

figuration space by one additional dimension: time. This allows to optimize in space-time, which leads to trajectories which differ from those obtained by the original CHOMP. In general, such approach gives more freedom to the algorithm, thus increasing the available range of valid solutions. The authors mention that the implementation is very sensitive to parameters and without good tuning it is possible to obtain a "collision-free" solution where the robot slowly goes through the obstacles. Overall, the idea behind this work is similar to ours.

Stochastic Trajectory Optimization for Motion Planning (STOMP) [1] adopted the distance representation from CHOMP. However, instead of using gradient descent it uses sampling technique. This allows to use non-differentiable cost functions and decrease the chance of being stuck in the local minima. As in this work we modify the STOMP algorithm, it is described in more detail in the next section.

Ricarda Steffens in her Master thesis [21] proposed a modification of STOMP. The idea is to consider the uncertainty of the future through employing multiresolution in time. An initial part of a trajectory is being planned with high resolution. However, parts of the trajectory which are located further away in time are planned with lower resolution. This allows to decrease the runtime and apply the method in dynamic environments through frequent replanning. The approach has been applied to service robot Cosero [22].

In the work of Chonhyon Park et al. [23] the problem of planning in dynamic environments is covered. Authors present an incremental approach: Incremental Trajectory Optimization for Real-time Replanning in Dynamic Environments (IT-OMP). They use a stochastic trajectory optimization framework in order to obtain a collision-free smooth trajectories. The method does not require any prior information. In order to handle the dynamics of the environment, a local bound around each obstacle is computed over a short time interval and after that is used to plan a collision-free path in incremental manner. The future positions of the obstacles are estimated using previous measurements, taking the sensor noise into account. That is why the plan is guaranteed to be safe only during short amount of time after it was generated. Thus, a frequent replanning is needed. In order to support this, the robot must be able to perceive the information about the environment frequently enough. This approach uses Euclidean Distance Transform (EDT) [24] as representation for the environment as well as many previously described methods. The limitation of this method is that it is necessary to set the overall trajectory duration and a duration for each waypoint before the optimization process starts.

In Motion Planning with Sequential Convex Optimization and Convex Collision Checking [25] authors use convex-convex collision checking instead of signed distance fields and spheres. The motivation behind that is the increased accuracy of this approach which allows to plan for complex tasks as surgery with a needle. In-

stead of gradient descend this method uses sequential quadratic programming [26] with exact penalty method. It minimizes the cost function by iterative construction of local approximation of it. However, in order to perform a convex-convex collision checking one needs to obtain an environment representation as a set of convex shapes from the raw sensor data. This operation is computationally expensive and may be imprecise.

In contrast to previously discussed approaches, Nikita Kitaev et al. [27] viewed the problem of approaching a pre-grasp poses from different perspective. In their work they investigate a problem of reaching a pre-grasp pose when there is no collision-free trajectory at all due to highly cluttered environment, for instance, a refrigerator shelf. In order to solve this task, the robot must move the objects away from the vicinity of the target pose. The authors propose a physics-based trajectory optimization. They utilize a full physics simulator in order to solve such problems. The authors assume that the objects are placed on a horizontal surface, and that the objects may be either movable or static. Meanwhile the collisions with the static objects still must be avoided, the collisions with movable objects are the key to success in these tasks. Using physics simulator it is possible to chose a trajectory, that minimizes the harm to moved objects, i.e. they do not fall and do not push other objects. The issue is that physics simulation is costly. Thus, only simple shapes are used. Moreover, an assumption that all objects are positioned on a horizontal plane is made. This method is an example of a completely different view on the problem.

In this section we briefly described some significant related work. It is possible to see that different approaches has been proposed to solve the manipulation planning and optimization problem. Several authors viewed the problem from different perspectives and came up with various possible solutions. However, there is a lot of room for improvement of a planner/optimizer which tackles the manipulation planning in any context.

3 Theoretical Background

In this chapter we present a background which is necessary for the discussion of our approach. First, we review fundamental concepts as: kinematic chain, forward and inverse kinematics. Finally, we define the manipulation planning problem and its core components.

3.1 Kinematic Chain

A goal of a manipulation planning is to find a feasible path of a manipulator of a robot. Nowadays most robots are composed from set of rigid bodies which are connected with joints. A kinematic chain is used in order to model this structure. In this section we describe, what is a kinematic chain.

A kinematic chain is a model which describes a mechanical system of n rigid bodies connected to each other by J joints [28]. A rigid body in this system is called link. The very first link in the kinematic chain is called fixed link. The very last link in the kinematic chain is often called end-effector. In context of manipulators, the fixed link is usually the body of a robot to which the manipulator is connected. Consequently, the end-effector is a gripper which is used to interact with the environment. An example of a kinematic chain is shown in Figure 3.1.

A mobility M of a kinematic chain is defined by its number of degrees of freedom [29]. Each link has 6 DOF: 3 for position and 3 for orientation. Thus, a system of N links has mobility $M = 6N$. However, in most cases a joint imposes constraints on movement of links, which are connected by this joint. For example, hinge or prismatic joints remove five degrees of freedom, thus allowing transforms which involve one dimension only. The freedom f of a joint can be determined as $f = 6-c$ where c is the number of degrees of freedom which are constrained by this joint. Therefore, the mobility of a kinematic chain with N links and J joints is defined as:

$$M = 6N - \sum_{J} (6 - f_j) \tag{3.1}$$

A state of a kinematic chain is defined by a vector θ which has M dimensions.

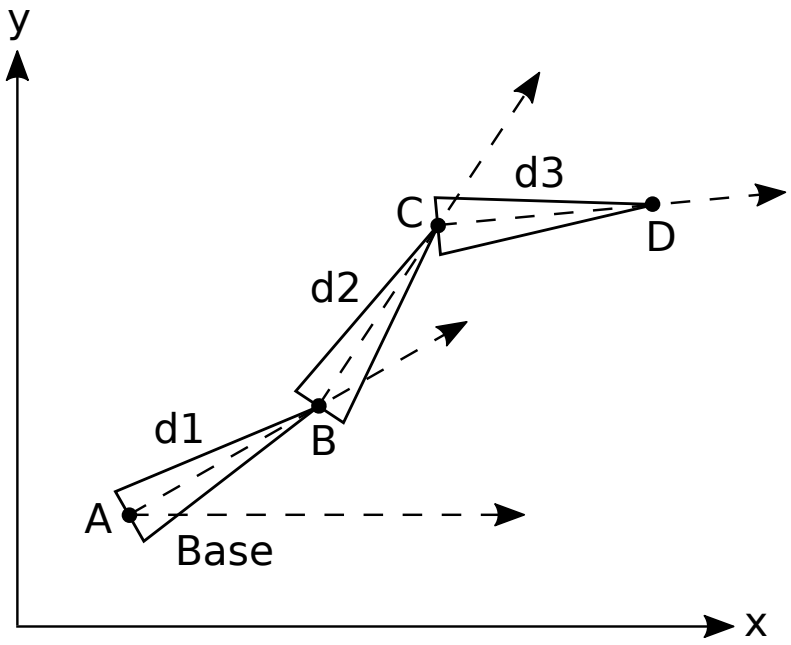


Figure 3.1: Example of the kinematic chain in 2D, where A, B, C, D are the joints and d1, d2, d3 are the links.

Given such a vector one can obtain the exact positions of each link. This is done using forward kinematics.

In order to define a forward kinematics for a robot, a suitable kinematic model is necessary. One of the most often used methods is Denavit-Hartenberg method [30]. It uses four parameters in order to describe a kinematic model: link length, link twist, link offset and joint angle. Given these parameters, it is possible to define a rigid transformation matrix ${}^{i-1}_{i}T$ for any link i, relatively to previous link $i-1$. The forward kinematics of the end-effector relatively to the base frame, given N links, is defined as:

$$ {}^{\text{base}}_{\text{end-effector}}T = \prod_{i=1}^{N} {}^{i-1}_{i}T \tag{3.2} $$

Note that the rigid transformations used in the above equations are assumed to be proper. It is possible to decompose any proper rigid transformation into rotation followed by a translation. Given vector v the rigid transformation is

defined as:

$$T(v) = Rv + t \tag{3.3}$$

Where R is a rotation matrix and t is a translation vector. For the rigid transformation to be proper it is necessary that $det(R) = 1$, i.e. the reflection is excluded.

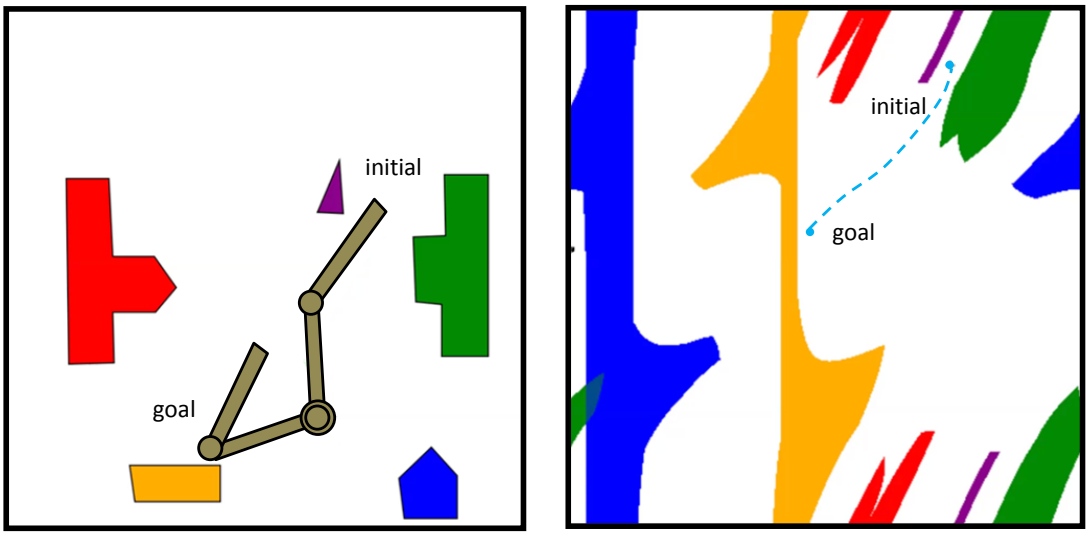
In order to solve the opposite problem i.e., to find a joint configuration which corresponds to a given target link pose, the inverse kinematics is used. Inverse kinematics addresses a problem of finding joint positions which produce a desired position of the end-effector. Often there is more than one solution. It is possible to solve inverse kinematics problem both with analytical and numerical approaches. In most cases it is much faster to use an analytical closed-form solution [31].

3.2 Manipulation Planning

Manipulation planning can be split up into two main phases. In first phase a goal pose is generated, for example: the exact pre-grasp pose. The second part is to plan a feasible trajectory which will allow to reach the pose obtained from the previous step. In this work we focus on the second part of the manipulation planning pipeline. Thus, it is assumed that the goal pose is given.

A manipulation planning algorithm is provided with start and goal configurations and is expected to output a feasible plan of movement from the start to the goal. This plan consists of a finite number of waypoints N. Each waypoint is a valid configuration and is referenced as keyframe further in this work. The trajectory is composed in a way that consequent movement from current configuration n to the next configuration $n + 1$ will lead to the goal configuration.

It is possible to express a motion planning problem as a search in configuration space for a sequence of valid configurations which consequently lead from a start pose to a goal pose. However, large portion of possible configurations are not valid due to various reasons, such as: violated joint limits, self collisions, collisions with the environment, violated orientation or position constraints, violated torque limits. That is why a configuration space for the manipulation planning has complex structure. In Figure 3.2(a) a simple 2D world space with 2 DOF chain (grey) in it is shown. There are initial and goal positions. The movement of the chain is limited by several obstacles. In Figure 3.2(b) a respective configuration space is shown. It is possible to see that even simple scene has quite complex configuration space. In configuration space one can easily observe that the desired movement is possible. However, for example a large portion of region between blue and yellow obstacle is unreachable by the given chain.



(a) World space (b) Configuration space

Figure 3.2: World and configuration spaces for a simple 2 DOF chain in 2D. The figure is taken from [32].

For low-dimensional configuration spaces it is feasible to discretize them into grid-based structure and after solve the problem with a help of trivial path-finding algorithms such as A*. However, for configuration spaces with higher dimensions, this approach is not feasible because the number of cells grows exponentially while the number of dimensions grows linearly, which is called a curse of dimensionality. Thus, several other approaches are utilized in order to overcome the curse of dimensionality.

In this section a task of manipulation planning was reviewed. We formulated inputs and outputs of a general motion planning algorithm and outlined main difficulties of this task. It is possible to conclude that there is no trivial solution which reliably and effectively solves this task.

4 STOMP

There exist many different approaches to manipulation planning problem. Stochastic Trajectory Optimization for Motion Planning (STOMP) by Kalakrishnan et al. [1] is one of them. We have chosen this algorithm to be a core of our approach due to its properties: runtime linear in a number of dimensions and ability to handle non-differentiable costs. In this chapter we discuss the original algorithm in detail. First, we give a brief overview of the method, then we present all the details about the algorithm. In the end we discuss its advantages and disadvantages and formulate a conclusion about why exactly this algorithm has been chosen.

4.1 Overview

In STOMP a stochastic trajectory optimization technique is utilized. It is provided with an initial trajectory, which may be a simple interpolation between start and goal configuration. The trajectory is defined in joint space and consists of fixed predefined number of equally spaced in time keyframes and has a predefined fixed duration. When optimization process is completed, STOMP outputs resulting trajectory.

STOMP is based on another optimization-based algorithm - CHOMP. CHOMP uses covariant gradient descend technique in order to optimise an initial trajectory. Because of the use of gradient based methods it is not possible to use non-differentiable costs. Moreover, CHOMP as any gradient-based algorithm suffers from local minima. STOMP adopts world and robot representation methods, as well as similar cost function, but with use of random sampling instead of gradient descent, it manages to overcome the problem of being stuck in local minima, which is typical for CHOMP. At the same time more freedom in cost function design is obtained.

STOMP is given an initial trajectory, which is often just a straight interpolation between start and goal configurations. Then, random samples are generated using the initial trajectory as a mean. By doing so the space around initial trajectory is explored. Each of the samples is evaluated with a cost function. At the end the distribution of the costs is estimated and the mean and standard deviation are

adjusted to fit this distribution and minimize the cost. This procedure is similar to the expectation maximization [33]. A process described above is a single STOMP iteration. The method performs iterations until a termination criteria is met.

This procedure allows to obtain smooth trajectories which are ready to be executed without additional postprocessing. However, a lot of computations are involved. Thus, an efficient representation of the robot and the environment is required. STOMP approximates the robot bodies with a set of spheres. The environment is represented using Euclidean Distance Transform (EDT) [24] which can be efficiently precomputed from point cloud obtained from a sensor, or from any set of geometric primitives, such as spheres and parallelepipeds.

4.2 Algorithm

In STOMP the planning task is considered as an optimization problem. Thus, the purpose of the algorithm is to find a trajectory which has the minimal cost according to a given cost function.

The input of the STOMP is an initial trajectory vector $\theta \in R^N$ which consists of N equally spaced in time keyframes in joint space, and has a predefined fixed duration T. In most simple cases it is a straight interpolation between a start configuration x_{start} and a goal configuration x_{goal}. During the optimization process, start and goal configurations remain unchanged. STOMP outputs an optimized trajectory $\theta_{optimized}$. In order to keep the explanation simple, the algorithm is presented for 1-dimensional case. However, in order to apply it to problem with higher dimensions, the steps described below are applied for each dimension. Thus, STOMP has a complexity linear in number of dimensions and can be scaled easily for a task with arbitrary number of dimensions.

The optimization problem which STOMP attempts to solve can be formulated as:

$$\min_{\tilde{\theta}} \mathbb{E}\left[\sum_{i=1}^{N} q(\tilde{\theta}_i) + \frac{1}{2}\tilde{\theta}^\top R\tilde{\theta}\right] \tag{4.1}$$

where $\tilde{\theta} = \mathcal{N}(\theta, \Sigma)$ is a noisy joint parameter vector, given that θ is the mean and Σ is the covariance, $q(\tilde{\theta}_i)$ is a defined state cost function which may include components as: obstacle cost, torque cost, constraints cost. Each state $\tilde{\theta}_i$ of the trajectory θ is evaluated using this cost function. The term $\tilde{\theta}^\top R\tilde{\theta}$ is a sum of squared accelerations along the trajectory, where R is a positive semi-definite matrix which represents control costs. In order to compute R, the authors use a kernel A which produces acceleration $\ddot{\theta}$ when multiplied by the joint position

vector θ. Matrix A is defined using finite differencing [34] as:

$$A = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Thus, it holds that

$$\ddot{\theta} = A\theta \tag{4.3}$$

$$\ddot{\theta}^\top \ddot{\theta} = \theta^\top (A^\top A)\theta \tag{4.4}$$

Finally, the R is defined as $R = A^\top A$. Therefore, it is ensured that $\tilde{\theta}^\top R\tilde{\theta}$ is a sum of squared accelerations.

The predecessor of STOMP, CHOMP is taking a derivative of the expectation from equation 4.1 in order to minimize it. STOMP computes the gradient of the expectation from 4.1 with respect to $\tilde{\theta}$ instead:

$$\nabla_{\tilde{\theta}} \left(\mathbb{E} \left[\sum_{i=1}^{N} q(\tilde{\theta}_i) + 0.5\tilde{\theta}^\top R\tilde{\theta} \right] \right) = 0 \tag{4.5}$$

By differentiating the second term $\tilde{\theta}^\top R\tilde{\theta}$, we obtain:

$$\mathbb{E}(\tilde{\theta}) = -R^{-1}\mathbb{E} \left(\nabla_{\tilde{\theta}} \mathbb{E} \left[\sum_{i=1}^{N} q(\tilde{\theta}_i) \right] \right) \tag{4.6}$$

which can also be rewritten as $E(\tilde{\theta}) = -R^{-1}\delta\hat{\theta}_G$, with $\delta\hat{\theta}_G$ defining the gradient estimate

$$\delta\hat{\theta}_G = \mathbb{E} \left(\nabla_{\tilde{\theta}} \left[\sum_{i=1}^{N} q(\tilde{\theta}_i) \right] \right) \tag{4.7}$$

Instead of analytical functional gradient, the authors propose an estimated gradient formulated as follows:

$$\delta\hat{\theta}_G = \int \delta\theta dP \tag{4.8}$$

The use of an estimated gradient is inspired by research in probability matching [35] and path integral reinforcement learning [36]. This equation represents the expectation of the noise $\delta\theta$ in the parameter vector $\tilde{\theta}$ given the probability metric $P = \exp(-\frac{1}{\lambda}S(\theta))$. The term $S(\tilde{\theta})$ corresponds to the state dependent cost which is computed as $S(\tilde{\theta}) = [\sum_{i=1}^{N} q(\tilde{\theta}_i)]$. Finally, we can formulate the stochastic gradient as

$$\delta\hat{\theta}_G = \int \exp\Big(-\frac{1}{\lambda}S(\theta)\Big)\delta\theta d(\delta\theta) \tag{4.9}$$

STOMP utilizes the idea of path integral stochastic optimal control [36]. In the path integral stochastic optimal control the purpose is to find controls, that minimize the performance criteria, and thus, are optimal with respect to it. The controls are computed for each state x_{ti}, as $\delta\hat{u} = \int p(x)\delta u$ where δu are sampled control costs and $p(x)$ represents a probability for each trajectory τ_i which starts in x_{ti} and ends in x_{tN}. This probability is computed as $p(x) = \exp(-S(\tau_i))$ where $S(\tau_i)$ is the cost of the path $\tau_i = (x_{ti}, \ldots, x_{tN})$. It is possible to see that probability $p(x)$ is inversely proportional to the cost $S(\tau_i)$. That is why the trajectories with high costs will influence the optimal controls much less than trajectories with lower costs. The steps presented above are performed for each state x_{ti} until the terminal state x_{tN} is reached. After this the controls are being updated: $u = u+\delta\hat{u}$. Finally, the new set of trajectories is generated.

The assumption that state cost of each state $S(\theta_i)$ depends only on the parameters θ_i and does not depend on the previous or future costs is made in order to simplify the problem and achieve faster convergence. Thus, the problem is simplified to $S(\theta_i) = q(\theta_i)$. Complete STOMP for R^1 algorithm is presented in Algorithm 1.

In order to produce adequate trajectories, the noise ϵ has a zero mean normal distribution with a covariance matrix $\Sigma_\epsilon = R^{-1}$. This allows to obtain samples ϵ with low control costs $\epsilon^T R\epsilon$. Moreover, such a sampling insures that trajectory smoothly starts at the start configuration and smoothly converges towards the goal configuration. This sampling strategy allows to exclude many certainly unfeasible trajectories before spending computational time to evaluate them. At the same time decent exploration of the state space is still possible.

4.3 Cost Computation

The cost for the trajectory θ consists from state cost and control costs. The procedure of control cost computation is described in the previous section. The

Algorithm 1: STOMP for R^1

Given:
Start and goal positions x_0 and x_N
Initial trajectory vector θ
A state-dependent cost function $q(\theta_i)$
Precompute:
A - finite difference matrix
$R^{-1} = (A^\top A)^{-1}$
$M = R^{-1}$
while *Cost* $Q(\theta)$ *has not converged* **do**
  // Sample K random trajectories
  for $k = 1$ *to* K **do**
    $\tilde{\theta}_k = \theta + \epsilon_k$, where $\epsilon_k = \mathcal{N}(0, R^{-1})$
  // Evaluate each sample, compute probabilities
  for $k = 1$ *to* K **do**
    for $i = 0$ *to* N **do**
      $S(\tilde{\theta}_k, i) = q(\tilde{\theta}_k, i)$
      $P(\tilde{\theta}_k, i) = \frac{e^{-\frac{1}{\lambda} S(\tilde{\theta}_k, i)}}{\sum_{l=1}^{K} [e^{-\frac{1}{\lambda} S(\tilde{\theta}_l, i)}]}$
    for $i = 1$ *to* $N - 1$ **do**
      $|\delta\tilde{\theta}|_i = \sum_{k=1}^{K} P(\tilde{\theta}_k, i)[\epsilon_k]_i$
  // Smooth noisy parameter update and update trajectory
  $\delta\theta = M\delta\tilde{\theta}$
  $\theta = \theta + \delta\theta$
  // Calculate trajectory cost
  $Q(\theta) = \sum_{i=1}^{N} q(\theta_i) + \frac{1}{2}\theta^\top R\theta$

state cost is computed as

$$q(\theta) = \sum_{t=0}^{T} q_o(\theta_t) + q_c(\theta_t) + q_t(\theta_t) \tag{4.10}$$

where $q_o(\theta_t)$ is obstacle cost, $q_c(\theta_t)$ is constraint cost and $q_t(\theta_t)$ is torque cost. Below we discuss each cost component in more detail.

Before the planning process has been started, the EDT is computed from a point cloud obtained from a sensor. The robot body B is approximated with a set of spheres $b \in B$. Given such a setup, collisions as well as contact distances may be computed fairly fast. An obstacle cost, which penalises collisions and being close

to the obstacles is defined as

$$q_o(\theta_t) = \sum_{b \in B} \max(\epsilon + r_b - d(x_b), 0)||\dot{x_b}|| \tag{4.11}$$

where ϵ is the minimum allowed distance to the obstacle, r_b is the radius of the sphere b and $d(x_b)$ is the distance to the closest obstacle relative to center of the sphere b obtained from the signed EDT. The term is multiplied by the magnitude of the workspace velocity of the sphere $||\dot{x_b}||$ which ensures that planning algorithm will not move quickly through high obstacle cost regions in order to reduce the total cost.

In order to satisfy the constraints on the position or orientation of the gripper the authors introduce the constraint cost which is computed as

$$q_c(\theta_t) = \sum_{c \in C} |v_c(\theta_t)| \tag{4.12}$$

where $c \in C$ is a constraint from the defined set of constraints C and v_c is a function which determines the extent of constraints violation in configuration θ_t.

Torque minimization allows to perform manipulation tasks in energy efficient way. Moreover, it ensures that motors do not experience load beyond their limits. It is possible to compute torque for each joint in any point of time, given position velocity and acceleration and dynamics model of the robot. The magnitudes of the torques τ are summed together and added to the total cost function:

$$q_t(\theta_t) = \sum_{t=0}^{T} |\tau| dt \tag{4.13}$$

4.4 Analysis

STOMP does not use any gradient method explicitly. Instead, it utilizes a derivative-free stochastic optimization method. This allows to construct a cost function with parts which are not smooth and non-differentiable, such as position/orientation constraints and joint torques. These constraints are of significant importance when performing manipulations with objects, especially with heavy ones. In addition, STOMP has complexity linear in a number of dimensions. This gives an opportunity to effectively scale it and apply to problems of higher dimensionality, when additional joints, such as torso yaw or pitch, are involved into the planning.

In contrast to gradient-based approaches, STOMP does not suffer from local minima. Also, a smaller amount of iterations is necessary in order to converge, as much larger steps can be made during the exploration of the configuration space.

STOMP outputs trajectories which are smooth and do not contain unnecessary movements, which is the case for many other planners. Thus, it does not need an additional postprocessing of the trajectory and hence, the runtime of the whole manipulation planning pipeline is reduced. Moreover, as STOMP is an optimization-based algorithm, the resulting trajectories are not only smooth but also optimal with respect to constraints introduced in the cost function.

However, STOMP has several major disadvantages. As it uses a stochastic approach, it needs to evaluate many samples in order to perform the exploration. This leads to large amount of cost computations which is an expensive operation. The speed of convergence as well as finding a feasible solution heavily depends on the initial trajectory.

The duration of the trajectories is fixed. That is why even when the end-effector should move for a short distance, the trajectory is executed very slowly. Also this setting does not allow to optimize the velocities of the trajectory and hence, minimization of torques can be performed better. In addition, fully discrete policy for cost computation requires large amount of keyframes, i.e. 100. This not only slows down the planning process due to many unnecessary cost computations but also leads to trajectories which are uncomfortable to execute due to enormous amount of keyframes.

Considering the advantages of the STOMP, discussed in this section, i.e., ability to optimize non-smooth costs, linear complexity, ability to overcome a local minima, we have chosen this algorithm as the basis for our work. In order to achieve better performance, we propose a series of modifications which will allow to reduce the planning time, rise the success rate and improve the torque minimisation altogether with ability to optimize the execution time of the trajectories.

5 Approach

5.1 Overview

We have chosen the STOMP algorithm as a starting point for our research of an arm manipulation planning in a disaster scenario in order to meet the requirements formulated in Section 1.2. In this section we give an overview of modifications we have made to the original algorithm.

In the original STOMP, the state cost function is applied separately to each keyframe of a trajectory. This allows to threat each keyframe independently and speed up the process of convergence. However, it requires a substantially large amount of keyframes in order to guarantee that the generated trajectory is collision free and there are no places, where there is an obstacle in between two collision-free keyframes which leads to a collision when the trajectory is executed. In the original STOMP 100 keyframes are usually used. This large amount of keyframes introduces a heavy computational load, which is not necessary in many cases. For instance, this amount is reasonable for sufficiently long trajectories, meanwhile in case of very short trajectories, most part of the computations are not necessary. However, it is not possible to determine in advance the length of the trajectory, and hence, the amount of keyframes necessary to check the whole trajectory for collisions. Thus, there is an obligation to use a large amount of keyframes. Moreover, this discrete cost computation makes estimation of continuous characteristic of a trajectory, such as duration, difficult.

In contrast to the original STOMP, we propose to compute the state cost not for separate keyframes, but for consequent transitions between them. Given a trajectory θ which consists from N keyframes the state cost can be computed as:

$$q(\theta) = \sum_{i=0}^{N-1} q(\theta_i, \theta_{i+1}) \tag{5.1}$$

Where $q(\theta_i, \theta_{i+1})$ is a cost for a transition from the configuration θ_i to the configuration θ_{i+1}. This policy for cost computation allows to take full control over amount of collision checking and other expensive operations. Given a pair of keyframes, it is possible to determine the exact amount of intermediate configura-

tions which have to be checked in order to cover the transition from θ_i to θ_{i+1} with required precision. This ensures that only necessary amount of computations is made. This model allows to plan for trajectories with substantially smaller amount of keyframes: 10-20. Note, that we keep the second term of the original cost (Equation 4.1) of the STOMP, while replacing the first term, which corresponds to the state cost. The second term penalizes non-smooth trajectories, which is essential for our purposes as well. Thus, an optimization problem now can be formulated as:

$$\min_{\theta} \mathbb{E}\left[\sum_{i=0}^{N-1} q(\theta_i, \theta_{i+1}) + \gamma\theta^\top R\theta\right] \tag{5.2}$$

Where $\gamma \in [0, 1]$ is the importance weight of the control cost.

Introduced policy of a trajectory state cost estimation fits better the continuous nature of the trajectory, as we evaluate not separate configurations, but consequent transitions between them instead. During the evaluation of a transition it is possible to estimate the duration of the transition, and hence, attempt to minimize it. In order to achieve this, we extend the configuration vector by adding one more dimension for joint velocity used during the transition from the previous keyframe to the current one. Given the velocity it is possible to estimate the duration of the transition for each consequent pair of the keyframes. We introduce an additional term for the transition cost: a duration cost, which penalizes long durations. Given this cost component, STOMP now is able to find optimal velocities which allows to obtain trajectories with lower durations. This provides an access to accelerations along the transition as well, which allows to estimate torques and optimize the trajectory taking this feature into consideration as well. In contrast to the original STOMP, where a duration of a trajectory is fixed, our modification is capable of choosing optimal path/velocity values which lead to optimal duration/torque relation.

Furthermore, the runtime may be reduced by optimizing the world representation. We keep a signed EDT as a main tool for the world modeling. Further we refer to the signed EDT as signed distance field, as this term is more appropriate when talking about data structure itself. We make an assumption that while performing a manipulation task, the robot base is stable and does not move. In contrast to the original algorithm, where single distance field is used for the environment representation only, we split up the robot representation into a dynamic and a static part. Geometry of the static part is precomputed and stored into a separate distance field. Meanwhile the distance field which represents the environment must be recomputed for each consequent use of the optimizer, as the

environment may change, the static part of the robot remains the same. Thus, the amount of the calculations required to take place before the optimization process starts, is reduced.

Overall, we introduce the transition cost function which can be formulated as follows:

$$q(\theta_i, \theta_{i+1}) = q_o(\theta_i, \theta_{i+1}) + q_{lim}(\theta_i, \theta_{i+1}) + q_c(\theta_i, \theta_{i+1}) + q_d(\theta_i, \theta_{i+1}) + q_t(\theta_i, \theta_{i+1}) \quad (5.3)$$

Where q_o is an obstacle cost, which penalizes collisions and being close to the obstacles, q_{lim} is a joint limit cost which penalizes violations of joint limits, q_c is a constraint cost which penalizes any custom constraints of the end-effector position/orientation, q_d is a duration cost, which penalizes long durations and q_t is a torque cost which penalizes high torques. Each cost component function $q_j(\theta_i, \theta_{i+1})$ is designed so that:

$$q_j(\theta_i, \theta_{i+1}) = \begin{cases} \lambda_j \cdot q_k(\theta_i, \theta_{i+1}), & \text{if } \theta_i \to \theta_{i+1} \text{ is valid} \\ \gg 1, & \text{otherwise} \end{cases} \quad (5.4)$$

Where $\lambda_j \in [0, 1]$ is an importance weight of the cost component q_j, and $q_k(\theta_i, \theta_{i+1}) \in [0, 1]$ is a cost for a valid transition defined for the cost component q_j. The term $\theta_i \to \theta_{i+1}$ corresponds to a transition from the configuration θ_i to the configuration θ_{i+1}. A transition from θ_i to θ_{i+1} is considered to be valid with respect to the cost component q_j if there are no critical violations of the constraints defined for q_j. For example, a collision with an obstacle is a critical violation with respect to the obstacle cost function, which makes this transition not valid. At the same time, having the closest obstacle in the distance of $x > 0$ meters makes the transition valid. A much larger cost for not valid transitions is designed so that the algorithm can not decrease costs of the other transitions in order to compensate a high cost for invalid transition. This design encourages a removal of any invalid transitions from the trajectory during the optimization process. So, by this cost design we attempt to overcome an issue mentioned by Anca D. Dragan et. al: "We noticed in our experiments that the CHOMP obstacle cost is less correlated to the feasibility of a trajectory than expected. A feasible trajectory that stays close to obstacles will accumulate more cost than an infeasible trajectory that collides slightly at one point, but on average stays further away from obstacles" [19]. At the same time, when the transition is valid, the cost component is scaled within interval $[0, 1]$, which allows to utilize a system of weights $\lambda_j \in [0, 1]$ in order to set a relative importance of the cost components.

This approach gives an opportunity to obtain optimized trajectories according to a specific criteria, depending on a concrete situation and task.

Overall, the modifications described above focus on decreasing the runtime, as well as on adding new features, such as optimizing a duration of a trajectory and controlling the character of the trajectory by the system of flexible cost weights. In the next sections we describe each component of the cost function. We finish this chapter with a brief description of the implementation of the optimizer.

5.2 Cost Function

In this section the proposed transition-based cost function is presented. It consists of the five main components: obstacle, joint limit, constraint, duration and torque costs. In the next subsections we describe each cost component in detail.

5.2.1 Obstacle Cost

One of the most important requirements for any robot motion is being collision free. This ensures that a goal is reached and both robot and its environment experience no harm. We present an obstacle cost which penalizes any collisions and proximity to any obstacles as well. In this section the procedure of the obstacle cost computation is described in details.

Collision checking is a very expensive operation in terms of computational time. During a motion planning process, large amount of configurations must be checked for collisions. That is why, in order to perform the planning in a reasonable amount of time it is necessary to choose a proper model for collision checking, as well as ensure that only necessary checks are performed. In the next subsections we first describe our environment model and then continue with the cost computation itself.

World Representation

In our work we adopt the approach of world representation from the original STOMP. It utilizes the assumption that the environment is static and precomputes all necessary information for the collision checking by means of a signed distance field.

We utilize this assumption even more by dividing the robot body into static and dynamic parts. As static parts do not move during the planned motion, their geometry is precomputed as well and stored into a separate distance field. This allows to reduce the amount of computations needed before the motion planning

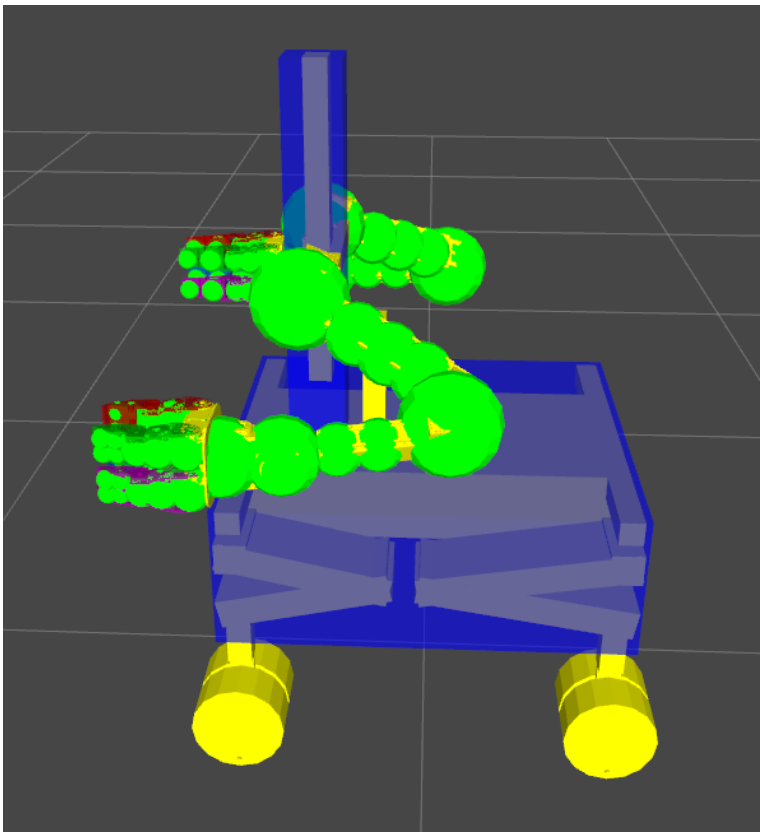


Figure 5.1: The collision approximation of the Momaro robot used in the experiments. Green: dynamic part; Blue: static part; Yellow: real geometry.

process started, as this distance field must be computed only once, meanwhile the environment distance field is computed before each planning task.

The dynamic part of the robot body which takes part in the planned motion is approximated with a set of spheres as in original STOMP. During the collision checking each sphere is checked for collision with environment and with static part of the robot by means of corresponding distance fields. The example approximation of the Momaro centaur-like robot [37] which is used in the experiments is shown in Figure 5.1. The dynamic part of the robot, which is represented by the left and right arm in this case, is approximated with a set of spheres (green). The static part, which is the robot base and trunk, is approximated with two parallelepipeds (blue). Note, that for the static part it is possible to use representation of any complexity, as it is precomputed only once. We use very simple representation in this example.

In order to perform collision checking within dynamic part (i.e., self collisions of moving parts) of the robot, the spheres are divided into collision groups. It is

necessary to exclude collision checks between groups which physically can not collide, so Allowed Collision Matrix (ACM) is introduced. This matrix only contains pairs of the groups which potentially can collide. Each group has a simplified model which is represented as a sphere with radius and center, that covers all other spheres which form this group. By performing the collision check between two groups using their simplified models, it is possible to reduce the computations by excluding full collision checking for collisions which are definitely not present. If the simplified collision check fails, a full check is done by checking each pair of spheres from two groups for a collision.

To conclude, our world model consists of the following components:

- *Environment* - represents a surroundings of a robot with a signed distance field.
- *Robot static part* - represents parts of the robot body which do not take part in manipulation with a signed distance field.
- *Robot dynamic part* - represents parts of a robot body which take part in the manipulation as a set of spheres. This set is divided into collision groups in order to perform self collision checking.

Cost Computation

Given the model of the environment and the robot, as described in the previous subsection, one may perform collision checking and estimate obstacle cost $q_o(\theta_i, \theta_{i+1})$ for the transition from the configuration θ_i to the configuration θ_{i+1}.

In order to estimate obstacle cost, it is necessary to determine a set of intermediate configurations Θ which covers the space between θ_i and θ_{i+1} with a defined precision. To achieve this, first we apply forward kinematics to θ_i and θ_{i+1} and determine the link which moves for the longest Euclidean distance d. In most cases this link is the end-effector, but sometimes it may be some other link, for example elbow. Dividing the distance d by a given precision p gives the amount of the intermediate configurations which must be checked for collisions in order to cover transition from θ_i to θ_{i+1} with required precision. These configurations are equally spaced along the transition. Precision p defines the distance between each pair of intermediate configurations along the transition.

This approach allows to perform only necessary collision checks. It guarantees that the transitions along the trajectory are checked for collisions with required precision independently of the length of the trajectory. This allows to speed up the planning process and to safely plan for the trajectories with lesser keyframes

amount. However, there is no need to check intermediate configurations with high precision for transitions when obstacles are far away, as such transition is guaranteed to be collision free. At the same time, in situations when the transition is performed in the area with high density of the obstacles, it may be necessary to check the transition with much higher precision. Needless to say that if a very high precision is used constantly, it results in a significant growth of the runtime. At the same time lower precision may be insufficient in certain scenarios.

This leads us to the idea of utilizing a variable precision. In situations when the obstacles are far away, a low precision is used, while in situations when the obstacles are close, a high precision is used. In order to estimate the precision necessary for the particular transition, we estimate the distance d_{obst} to the closest obstacle as: $d_{obst} = \min(dist(\theta_i), dist(\theta_{mid}), dist(\theta_{i+1}))$, where $dist(\theta_i)$ is a function which estimates the minimum distance to the obstacle for a given configuration θ_i and θ_{mid} is a middle configuration between θ_i and θ_{i+1}. Given a finest allowed precision p_{max}, it is possible to compute the precision p necessary for the particular transition from θ_i to θ_{i+1} as:

$$p = \max\left(\frac{d_{obst}}{2}, p_{max}\right) \tag{5.5}$$

Distance to the closest obstacle for the configuration θ_i is estimated as follows:

$$dist(\theta_i) = \min_{s_j \in S}(EDT(s_j) - r_{s_j}) \tag{5.6}$$

Where S is a set of spheres used to approximate a robot body. r_{s_j} is a radius of the sphere s_j and $EDT(s_j)$ is the distance from sphere center s_j to the closest obstacle, obtained from the distance field. Position of each sphere s_j is determined by applying forward kinematics using configuration θ_i.

Consequently, given the length d of the transition from θ_i to θ_{i+1}, we obtain the amount of uniformly spaced intermediate configurations $K = \frac{d}{p}$. This allows to decrease the computational time necessary by performing less checks in situations with low obstacle density and at the same time - increase reliability of the planner by performing a high-precision collision checking in situations with high obstacle density.

Note, that the parameter p_{max} have to be chosen carefully, as too low maximum precision p_{max} may lead to undetected collisions in some of the cases when $\frac{d_{obst}}{2} < p_{max}$ and the collision model as well as the environment have small elements with dimensions $\ll p_{max}$. The finest precision p_{max} is introduced in order to prevent unreasonably large number of intermediate configurations $K \to \infty$ which occurs in cases with $d_{obst} \to 0$. Introducing p_{max} in Equation 5.5 prevents this from

happening. It is noticeable, that one can always use constantly high precision p instead of the variable precision.

After that, it is possible to define a set Θ which consists of K uniformly spaced intermediate configurations obtained by interpolation from θ_i to θ_{i+1}. We define the obstacle cost for the transition from θ_i to θ_{i+1} as:

$$q_o(\theta_i, \theta_{i+1}) = \max\left(q_o(\theta_j) | \forall \theta_j \in \Theta, q_o(\theta_{i+1})\right) \tag{5.7}$$

Where $q_o(\theta_i)$ is the obstacle cost which determines how feasible the particular configuration θ_i is. Accordingly, the obstacle cost for the transition is defined as maximum cost detected along this transition. Obstacle cost $q_o(\theta_i)$ is computed as:

$$q_o(\theta_i) = \begin{cases} 0 & \text{if } d_{obst} \geq d_{max} \\ \lambda_o \cdot \left(1 - \frac{d_{obst} - d_{min}}{d_{max} - d_{min}}\right), & \text{if } d_{obst} > d_{min} \wedge d_{obst} < d_{max} \\ C_o \cdot |d_{min} - d_{obst}|, & \text{otherwise} \end{cases} \tag{5.8}$$

Where $\lambda_o \in [0, 1]$ is the importance weight for the obstacle cost, d_{min} is a minimum acceptable distance to the obstacles and d_{max} is a maximum distance to the obstacles which the algorithm should take into consideration. d_{obst} is the distance to the nearest obstacle which is calculated by function $dist(\theta_i)$ using world representation described in the previous subsection. $C_o \gg 1$ is a predefined constant which ensures that unfeasible configurations have a very high cost and thus the algorithm attempts to make them feasible in the first place. The term C_o is multiplied by $|d_{min} - d_{obst}|$ in order to indicate larger violations with larger cost. In case when configuration θ_i is feasible and the distance to the nearest obstacle fall in the interval $(d_{min}, d_{max}]$, the cost has smooth values in the interval $[0, 1]$. The equation used to calculate this value can be found at the second row in Equation 5.8. This allows to optimize trajectories which already do not have critically close configurations to the obstacles with $d_{obst} < d_{min}$. It results in a trajectories which have comfortable distances to the obstacles. If such behaviour is not needed, one can set $d_{max} = d_{min}$.

Distance to the closest obstacle d_{obst} is being computed taking into account both self-collisions and collisions with the environment. By using spheres for the robot approximation and utilizing the signed distance field it is possible to check for collision and obtain contact information very fast. In addition, spheres allow to check collisions against each other faster than more complex geometric primitives as parallelepipeds or prismatoids.

Overall, a presented way of computation of the obstacle cost effectively eliminates unnecessary cost computations and hence, allows to reduce the runtime. At

the same time, the use of variable precision for obstacle cost estimation of the transition between two configurations allows to detect collisions reliably.

5.2.2 Joint Limit Cost

This cost penalizes violations of joint limits. Normally, actuators of a robot have limited range of possible positions. Violation of these limits may cause damage to the motors. Another consequence is that the motion is not executed properly which results in the failed task and potential damage to the robot and/or the environment. Thus, it is essential to account for the joint limits. As any other cost component in our cost function, joint limit cost estimates the cost for a given transition from a start configuration θ_i to a destination configuration θ_{i+1}. The joint limit cost q_{lim} is formulated as:

$$q_{lim}(\theta_i, \theta_{i+1}) = \max\left(q_{lim}(\theta_j) | \forall \theta_j \in \Theta, q_{lim}(\theta_{i+1})\right) \tag{5.9}$$

Where the set Θ is the set of intermediate configurations which describe the transition from θ_i to θ_{i+1} with a given constant precision p. the constant precision is used, as this cost does not depend on the environment geometry. Other than that, the procedure is analogous to Equation 5.7.

Given upper and lower bounds on joint positions θ_{max} and θ_{min} and a configuration θ_k we first find if there exists any violation such that $\exists \theta_k^j \leq \theta_{min}^j \vee \exists \theta_k^j \geq \theta_{max}^j : \forall \theta_k^j \in \theta_k$, where θ_k^j is the position of the joint j in the configuration θ_k. We define the deviation $\Delta\theta_k$ from the limits as follows:

$$\Delta\theta_k = \min_{\theta_k^j \in \theta_k} (\Delta\theta_k^j) \tag{5.10}$$

Where $\Delta\theta_k^j$ is computed as:

$$\Delta\theta_k^j = \begin{cases} \min(\theta_{max}^j - \theta_k^j, \theta_k^j - \theta_{min}^j) & \text{if } \theta_k^j \geq \theta_{max}^j \vee \theta_k^j \leq \theta_{min}^j \\ \min(\theta_k^j - \theta_{max}^j, \theta_{min}^j - \theta_k^j), & \text{otherwise} \end{cases} \tag{5.11}$$

In case when there are no violations of joint limits, we record the magnitude $\Delta\theta_k$ of the smallest deviation from the limits (second row of Equation 5.11). Note, that if there is a violation of a joint limit, $\Delta\theta_k$ has negative value, or is zero. If there are not violations, $\Delta\theta_k$ has a positive value.

Given these values, we define the joint limit cost as:

$$q_{lim}(\theta_k) = \begin{cases} C_c \cdot (|\Delta\theta_k| + 1) & \text{if } \Delta\theta_k \leq 0 \\ \frac{1}{\varepsilon^2}\Delta\theta_k^2 - \frac{2}{\varepsilon}\Delta\theta_k + 1, & \text{if } 0 < \Delta\theta_k < \varepsilon \\ 0, & \text{otherwise} \end{cases} \tag{5.12}$$

Where $C_c \gg 1$ is a predefined constant which ensures that not feasible configurations have a very high cost and thus the algorithm attempts to make them feasible in the first place. The term ε is the magnitude of a considered safety margin. We do not include a corresponding importance weight λ_{lim}, as in our opinion, this cost is of significant importance in any case and can not be assigned a low priority. It is known that even though positions which are closed to the motor limits are valid and reachable, they still may cause harm to the motors. That is why we employ a smooth cost to penalize positions which are close to the joint limits. This cost is based on a quadratic function, such that largest considered deviation of ε leads to a cost value close to 0, meanwhile deviation close to 0 leads to a cost value value close to 1. This approach allows to produce trajectories which do not contain joint positions which are very close to the limits. In our work we use the value $\varepsilon = 0.1$ rad. The visualization of the cost function for the case $0 < \Delta\theta_k < \varepsilon$ is shown in Figure 5.2. One can observe the smooth grows to wards the maximum value of 1.

5.2.3 Custom Constraint Cost

This cost component is similar to the joint limit cost and preserves any other custom constraints on joint positions/orientations which are set specifically for concrete task. Mostly it is used to constraint the orientation of the end-effector when manipulating objects, for which preserved orientation is of high importance.

We apply the same procedure for constraint cost q_c, as for joint limit cost: given a set of configurations Θ, we compute the cost q_c for each of them. Then, the maximum cost is chosen as the cost for the transition. For any custom constraints on joint position or link orientation we check if there are any violations, as described in Equations 5.10 and 5.11, but limits which define custom constraints are used. If violations exist, we record the magnitude of the largest violation $\Delta\theta_k$. and compute a cost for the custom constraints as follows:

$$q_c(\theta_k) = \begin{cases} C_c \cdot (|\Delta\theta_k| + 1) & \text{if } \Delta\theta_k \leq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.13}$$

Where $C_c \gg 1$ is a predefined constant which penalizes any violations of custom constraints. We do not include a corresponding importance weight λ_c, as this cost

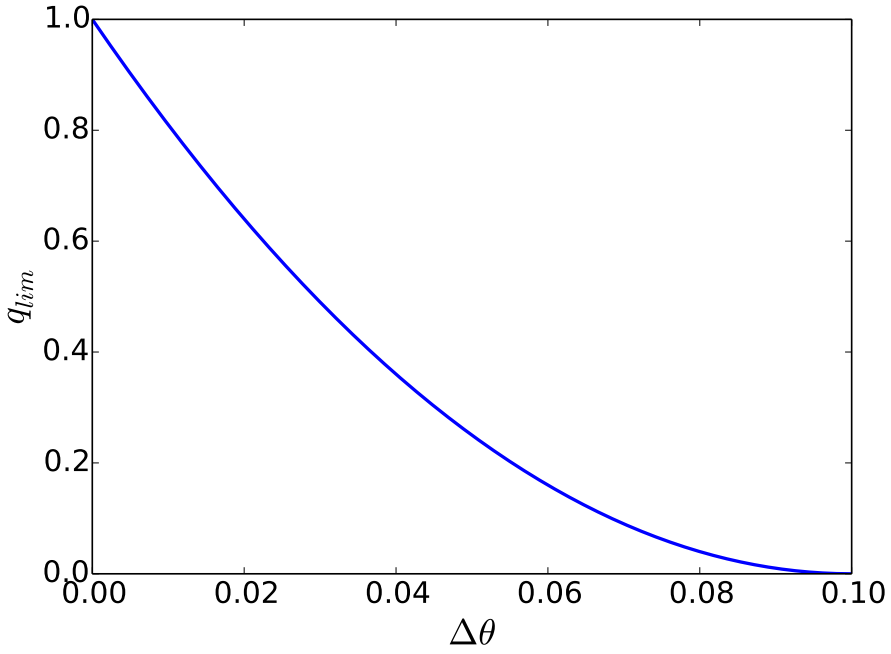


Figure 5.2: Cost q_{lim} for approaching a joint limit, with $\varepsilon = 0.1$.

has either a very large value, which penalizes violations, either is 0. This way of the constraint cost computation allows to optimize trajectories with respect to an arbitrary custom constraints which depend on a specific task.

5.2.4 Duration Cost

While planning of trajectories, using modern algorithms and hardware, often takes less than a second, an execution of the motion may take much larger fraction of time. That is why it is possible to sufficiently speed up the plan-execute pipeline by optimizing the duration of the trajectories during the planning process. In this subsection we present another component of our cost function, which penalizes long durations, and hence - allows to minimize a duration of a trajectory.

In order to have a mechanism to influence the duration of the trajectory, a linear velocity of a joint with longest path is added into a configuration space. We make an assumption that a duration necessary to execute a trajectory is bounded by the duration necessary to execute the trajectory of the joint with longest path. Thus, before evaluating a trajectory, a joint with longest path is determined, and consequent duration estimation is performed with respect to this joint. By doing so we avoid increasing dimensionality of a problem by a factor of two, which would result in large runtime growth. The extended configuration now consists of a joint

vector and a single value for the velocity: $\tilde{\theta}_i = \langle \theta_i, v \rangle$. For the transition from $\tilde{\theta}_i$ to $\tilde{\theta}_{i+1}$ the linear velocity v from the configuration $\tilde{\theta}_{i+1}$ determines the velocity which will be attempted to be reached by the joint with longest path overall.

In case of a continuous trajectory execution, it makes sense to plan trajectories which start with non-zero velocity. This may be very useful if a frequent replanning method is utilized. In this case, if during the movement it is necessary to replan because the environment or the task have changed, it is highly desirable to be able to change the movement "on the fly", without stopping the robot first. Our modification of STOMP algorithm provides this opportunity. Of course, it is possible to plan for the trajectories which end with non-zero velocity as well.

However, the Momaro robot that we use for the experiments currently has other trajectory execution policy. At each keyframe the velocity is 0, so the trajectory is being executed as a series of small movements between the keyframes with a complete stop at the end of each movement. That is why, in this case the velocity for the current transition does not depend on the velocity from previous transitions and overall the velocity along the trajectory does not have to be smooth. In the original STOMP sampling of the exploration noise for each dimension of the configuration is done using covariance $\Sigma = R^{-1}$. This allows to produce trajectories which are smooth and have low control cost. An example of this noise can be seen in Figure 5.3. It is possible to see that all trajectories proceed smoothly from the start to the goal. However, for the described case of "interrupted" trajectory execution, this behaviour is not needed. Thus, we restrict the exploration noise of last dimension, which represents the velocity, to be sampled with identity covariance $\Sigma = I$. Moreover, we exclude this dimension from control cost computation as it does not have to be smooth.

We prevent the velocity from exceeding the limits $0 < v < v_{max}$ on the stage of noisy trajectories generation by clipping it to that limit. Given the desired velocity for the transition from $\tilde{\theta}_i$ to $\tilde{\theta}_{i+1}$ it is possible to estimate the duration t necessary for this transition. The algorithm of the duration estimation depends on trajectory execution control. In Algorithm 2 we present the algorithm for continuous trajectory execution. The velocity is assumed to change uniformly during the transition. Distance d corresponds to the largest displacement between configuration $\tilde{\theta}_i$ and $\tilde{\theta}_{i+1}$.

Before the optimization process started, we estimate a maximum acceptable duration t_{max} for one transition in order to have an ability to scale duration cost from 0 to 1. We define t_{max} as: $t_{max} = \frac{t_{total}}{N-1}$, where N is the amount of keyframes and t_{total} is a duration of the initial trajectory executed with substantially low velocities.

In order to provide an additional level of safety for optimized trajectories, we

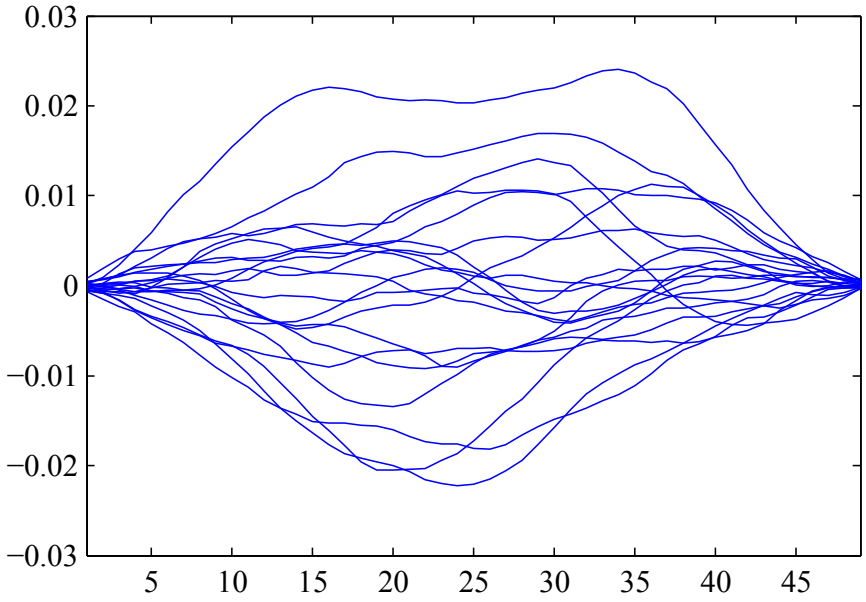


Figure 5.3: Noise samples obtained with 0 mean and covariance $\Sigma = R^{-1}$. The figure is taken from [1].

introduce an additional constraint on a velocity which depends on a distance to obstacles. The closer the robot is to the obstacle, the lower is the velocity. This constraint is represented as a set V of tuples of a form $\langle v, d \rangle$, where v is the maximum allowed velocity when the distance to the nearest obstacle is less than d. For example, a set V may be composed as: $V = (\langle 0.15, 0.2 \rangle, \langle 0.25, 0.3 \rangle, \langle 0.5, 0.5 \rangle)$, where the first tuple $\langle 0.15, 0.2 \rangle$ means that velocity $v > 0.15$ rad/s is not acceptable when distance d to the closes obstacle is < 0.2.

Finally, we determine the duration cost as:

$$q_d(\tilde{\theta}_i, \tilde{\theta}_{i+1}) = \begin{cases} C_v, & \text{if } \exists \langle v, d \rangle \in V : v > v_{\tilde{\theta}_i} \wedge d < d_{\tilde{\theta}_i} \\ \lambda_d \cdot \frac{t}{t_{max}}, & \text{if } t \leq t_{max} \\ C_d \cdot (t+1), & \text{otherwise} \end{cases} \tag{5.14}$$

Where $\lambda_d \in [0, 1]$ is the importance weight for the duration cost, $C_d \gg 1$ and $C_v \gg 1$ are predefined constants, which penalize exceeding of the duration limit and obstacle-velocity constraints respectively. The terms $v_{\tilde{\theta}_i}$ and $d_{\tilde{\theta}_i}$ are the velocity and the distance to the closest obstacle respectively, measured for a set of intermediate configurations Θ between $\tilde{\theta}_i$ and $\tilde{\theta}_{i+1}$, which was defined previously when estimating the obstacle cost. Using this approach, we manage to optimize the execution time of the trajectories, and hence, speed up the overall execution of

Algorithm 2: Duration estimation for continuous trajectory execution

Given:
Initial velocity v_1
Target velocity v_2
Distance to travel d

Estimate duration(v_1, v_2, d)**:**
if $v_1 = v_2$ **then**
  return $\frac{d}{v_1}$
else
  $a = \frac{v_2^2 - v_1^2}{2d}$
  return $\frac{v_2 - v_1}{a}$

the task. Velocity included into configuration space allows to estimate the torques better. We discuss our cost for torque minimization in the next section.

5.2.5 Torque Cost

The last component for our cost function is a torque cost. The purpose of this component is to penalize high torques and ensure that torque limits are not exceeded. It is very important to do so when objects with large masses are manipulated. Such situations happen especially often in disaster scenario, for example when a robot has to clear blockages. In this subsection we describe how the torque cost is computed.

In order to evaluate torque cost of the transition from $\tilde{\theta}_i$ to $\tilde{\theta}_{i+1}$ we find a set $\tilde{\Theta}$ of intermediate configurations which are uniformly distributed along transition with given precision p. We define the torque cost for the transition as:

$$q_t(\tilde{\theta}_i, \tilde{\theta}_{i+1}) = \max\left(q_t(\tilde{\theta_j}) | \forall \tilde{\theta_j} \in \tilde{\Theta}, q_t(\tilde{\theta}_{i+1})\right) \tag{5.15}$$

The torques τ affecting motors are expressed in function of joint positions and their derivatives: $\tau = f(\theta, \dot{\theta}, \ddot{\theta})$. As we have the linear velocity for the transition included into a configuration space, it is possible to estimate the velocity and acceleration for each intermediate configuration as well. As described in previous section, only the velocity of joint with longest path is available. Thus, we use this velocity for all joints with non-zero path during this transition and 0 velocity and acceleration for joints with zero path. By doing so we avoid underestimating the amount of torque. Given dynamic model of the robot it is possible to estimate the torques. We use the RBDL library for this purpose. Recursive Euler-Newton

algorithm [38] is used in order to iteratively estimate the torque for each joint of the kinematic chain. Once the magnitudes τ of the torques are computed, we define the torque cost of the configuration $\tilde{\theta}_i$ as:

$$q_t(\tilde{\theta}_i) = \begin{cases} C_t \cdot (\max_{j \in J}(\tau_j - \tau_{max}) + 1), & \text{if } \tau_j > \tau_{max} \\ \lambda_t \cdot \dfrac{\sum_J \tau_j}{J \cdot \tau_{max}}, & \text{otherwise} \end{cases} \tag{5.16}$$

Where $\lambda_t \in [0, 1]$ is the importance weight for the torque cost, τ_{max} is a maximum allowed torque for a single motor, and $C_t \gg 1$ is a predefined constant. In the first row of the equation above, we penalize any exceeding of the maximum allowed torque by large cost $\gg 1$. In the second row we produce a cost $\in [0, 1]$ which penalizes high torques. By employing this cost function we achieve effective torque minimization. In addition, any exceeding of the limit is removed in a first place as the cost in these cases is $\gg 1$.

5.3 Optimization Process

In the previous subsections the cost function which attempts to optimize different criteria of the trajectories was presented. However, this complex function leads to a complex solution space with many disjoint local minima. In this subsection we describe a strategy which is employed when applying STOMP to optimization tasks in order to find feasible trajectories more effectively.

The most severe barrier on the way to planning a feasible trajectory are obstacles. They may form very sophisticated geometries which make planning for kinematic chains with many DOF a challenging task due to complex configuration spaces. In most cases when there are obstacles present in the working area, the initial trajectory for a planning algorithm is going directly through obstacles. Thus, finding a collision-free trajectory is the first problem which must be solved by the planner. However, our cost function consists of five components, some of which are not relevant for this phase of planning. These components are the duration and torque costs. While the algorithm attempts to leave the region of collisions, the values of these costs are not important, because current solution is not feasible anyway. Accounting for these cost components at this stage would lead to longer convergence, as in some regions components may pull the trajectory into different directions. Moreover, these components slow down the process of optimization, as they introduce additional computations.

In order to solve this issue, the process of optimization is split into two consecutive phases. In the first phase a cost function consists only of three components:

obstacle cost, joint limit cost and constraint cost. We refer to this function as simplified. Optimization with simplified cost function continues until a valid collision-free trajectory is found. After this, the configuration space is being extended with linear velocity, and the second phase of the optimization starts. In this phase the full cost function with five components is used. This phase continues until termination criteria is met. We describe termination criteria below.

The termination criteria are:

- *Maximum iterations* - a total maximum number of iterations which algorithm is allowed to execute.

- *Maximum iterations after valid solution* - a maximum allowed number of iterations to be performed after a valid solution with no critical violations is found.

- *Maximum iterations with no improvement* - a maximum number of iterations to be executed without significant improvement of the cost. The improvement is measured in percents relative to the cost of the current best solution. If after given number of iterations the cost of the best solution improved less than for given percent amount, algorithm terminates.

Due to use of the complex cost function the search space has many disjoint local minima. In certain situations the algorithm can get stuck in these regions. In order to prevent failures, or unnecessary exhaustive runs in these cases, we apply the algorithm in iterative manner. If the algorithm can not improve the solution during given amount of iterations, and current best solution is not valid, the optimization process starts from scratch. However, in this case, the best solution from previous run is used as initialization. Much larger initial noise standard deviation tend to explore previously unseen areas and lead to a found valid solution. The amount of replanning attempts M is predefined and we typically use the value $M = 5$. If a valid solution is still not found after M attempts - the problem is not solved and the algorithm terminates. This procedure is shown in Algorithm 3.

This approach shows better results than attempts to solve the problem in one exhaustive run of the algorithm. The separation of the optimization process into two parts with simplified and full cost function respectively allows to decrease the time of the optimization, meanwhile keeping all the benefits of the full cost, as it will be discussed in Section 6.3.3.

Algorithm 3: STOMP modified

Given:
Initial trajectory θ
Termination criteria t
Number of replanning attempts M

STOMP-MODIFIED(θ, t, M)**:**
$i = 0$
while $isValid(\theta) = false \wedge i < M$ **do**
  Run stomp with simplified cost:
  $\theta = \textbf{STOMP}(\theta, t, \langle q_o, q_{lim}, q_c \rangle)$
  $i = i + 1$
if $isValid(\theta) = false$ **then**
  return θ
else
  Extend the trajectory with velocity dimension:
  v = 0.1 // default velocity
  $\forall \theta_i \in \theta$ **do:** $\theta_i = \langle \theta_i, v \rangle$
  Run stomp with full cost:
  $\theta = \textbf{STOMP}(\theta, t, \langle q_o, q_{lim}, q_c, q_d, q_t \rangle)$
  return θ

5.4 Implementation

In this subsection we briefly overview the implementation of the algorithm described above. We use a C++ as a main programming language and the whole system is designed under the Robot Operating System (ROS) [39] framework. The original implementation of STOMP was updated in order to be compatible with ROS Indigo. We keep the overall architecture of the system but exchange certain components with our implementations. The class diagram of the system is shown in Figure 5.4.

Classes *Stomp*, *PolicyImprovement* and *CovariantMovementPrimitive* were adopted from the original STOMP implementation and only minor changes were made to those classes. The other classes were implemented from scratch. Below we give a brief overview of functionality of each class.

- *StompNode* - is the highest class in the hierarchy. It represents the ros node and implements all necessary ros interfaces. It receives an optimization request and performs a high-level optimization using the *Stomp* object as described in Algorithm 3. It has some useful methods to provide the feedback about optimization process and final result.

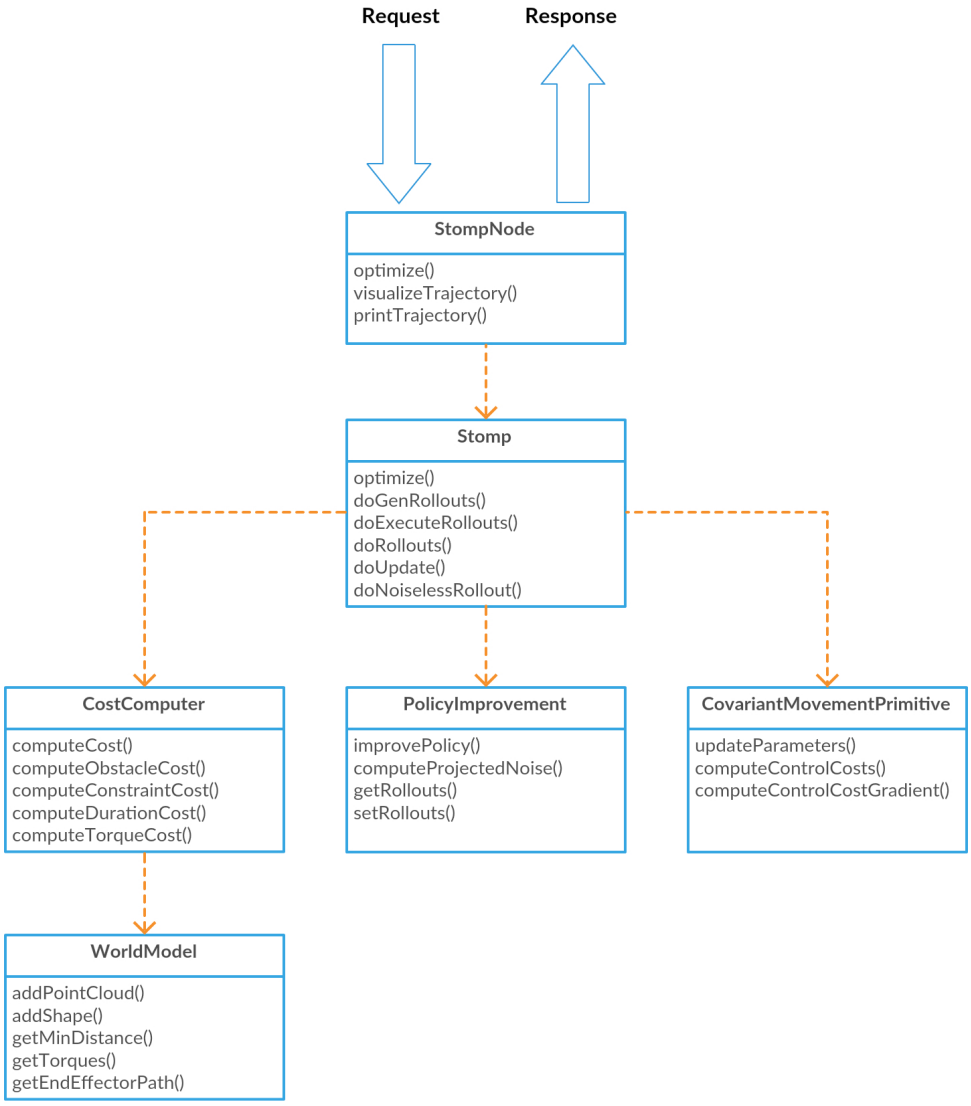


Figure 5.4: Class diagram of the implemented system. Only the main classes are shown.

- *Stomp* - the core class of the STOMP algorithm. It uses the objects of *PolicyImprovement*, *CovariantMovementPrimitive* and *CostComputer* in order to perform STOMP iterations.

- *PolicyImprovement* - implements main operations of the STOMP algorithm. This class handles generation of new trajectories, as well as generation of probability-weighted convex combination from a set of trajectories. Finally, in this class the parameter updates after each iteration are determined.

- *CovariantMovementPrimitive* - computes control costs, handles initialization of a finite difference matrix.
- *CostComputer* - transtion costs which are described in previous sections are computed in this class. It uses *WorldModel* class in order to obtain all data necessary to compute the costs.
- *WorldModel* - represents a world model as described in section 5.2.1. It uses MoveIt [40] library to compute forward/inverse kinematics. It utilizes an implementation of a signed distance field from MoveIt as well. It uses RBDL library to compute the torques.

5.5 Summary

In this chapter a proposed modification of the STOMP algorithm was presented. In contrast to the original algorithm, where a state cost function is used, which is applied independently to each keyframe in a trajectory, we introduce a transition cost function, which is applied to consecutive transitions between the keyframes of the trajectory. This allows to effectively control the amount of computations made as well as to estimate a duration cost.

We describe each component of a new transition cost function in detail: obstacle, joint limit, constraint, torque and duration components. In order to control the duration of the transition, a linear velocity of a joint with longest path is included into a configuration. In addition, this allows to optimize trajectories which start and end with non-zero velocities. This feature can be beneficial for applying the algorithm in a frequent-replanning manner.

Finally, a two-phased optimization is proposed. First phase lasts until collision-free valid trajectory is found, or maximum allowed number of iterations is performed. During this phase a simplified cost function, which consists only from obstacle, joint limit and constraint components, is used. During the second phase, the full cost is used (if duration and torque optimization is required by the task). Optimization continues until termination criteria is met. Finally, we present short description of the implementation of the algorithm.

6 Evaluation and Results

In order to measure the degree of success of our approach it is necessary to perform certain experiments and evaluate obtained results. We compare our method with three other algorithms: RRTConnect [8], Lazy Bi-directional KPIECE (LBKPIECE) [41] from Open Motion Planning Library (OMPL) [42] and with STOMP-Industrial from MoveIt Industrial [43]. In this chapter we first discuss our criteria for the evaluation and the environment for the experiments, than we present the experiments and the obtained results. Finally, we discuss the overall performance of our method and outline its strengths and weaknesses.

6.1 Criteria

Our trajectory optimization algorithm is designed to optimize different features of a trajectory. Thus, evaluation of it includes vast range of criteria. In this section we present a list of criteria which we have chosen:

- *Success rate.* One of the most important characteristic of any planner/optimizer is an ability to solve a planning task correctly. It is an essential requirement for successful use of the algorithm. It must be tested especially scrupulously, as STOMP is a stochastic approach, experiments with low number of trials may not reveal possible failures.

- *Runtime.* Determines how quickly a given problem can be solved. A runtime may limit solvable range of tasks significantly as some of the tasks require a solution to be available in a very short amount of time. Another use case of low-runtime algorithms is performing manipulation tasks in dynamic environments with a help of frequent replanning.

- *Trajectory length.* In some situations planners may produce unreasonably long trajectories, which makes them to be executed longer as well as introduces higher load for the motors.

- *Trajectory duration.* In most cases a duration of a trajectory is much longer than a planning time itself. That is why it is important to optimize the duration as much as possible.

- *Torque minimization.* Manipulation with heavy objects requires a minimization of torques in order to reduce a risk of hardware damage.

This set of criteria allows to evaluate different aspects of a planner and to reveal its advantages and disadvantages. In the next sections we describe a process of evaluation and present the obtained results.

6.2 Experiment Environment

We perform the experiments in simulation on a desktop computer with following specifications:

- CPU: Octa-core 4.00 GHz Intel Core i7-4790K
- RAM: 32 GB DDR3
- System type: 64-bit
- Operating system: Kubuntu 14.04 with 4.2.0-42 kernel
- ROS distribution: Indigo Igloo [44]

All evaluated algorithms run on a single core. Planning tasks are performed on the Momaro centaur-like robot [37]. Momaro has four articulated compliant legs which have a steerable wheels at the tips. This configuration makes the robot very mobile and dexterous. However, an upper body is much more interesting in the context of manipulation tasks. Momaro has two 7 DOF arms each equipped with 8 DOF gripper. In addition, the torso is able to rotate in a yaw plane. This setup creates a large workspace which is important for the manipulation tasks. Momaro robot is depicted in Figure 6.1.

The volume of a workspace covered and represented by a signed distance field is $2.0 \times 1.5 \times 1.5$ m. The distance field has a resolution of 1.5 cm. The highest precision for a collision checking of a transition is 1 check per 1 cm of a path. All start and goal configurations used in the experiments are defined manually by a human. For our modification as well as for STOMP-Industial a straight interpolation between start and goal is used as an initialization for all the experiments.

6.3 Experiments

In this section we describe the experiments performed and present the obtained results. We use the criteria from section 6.1 in order to evaluate our method and to compare it against three chosen approaches.

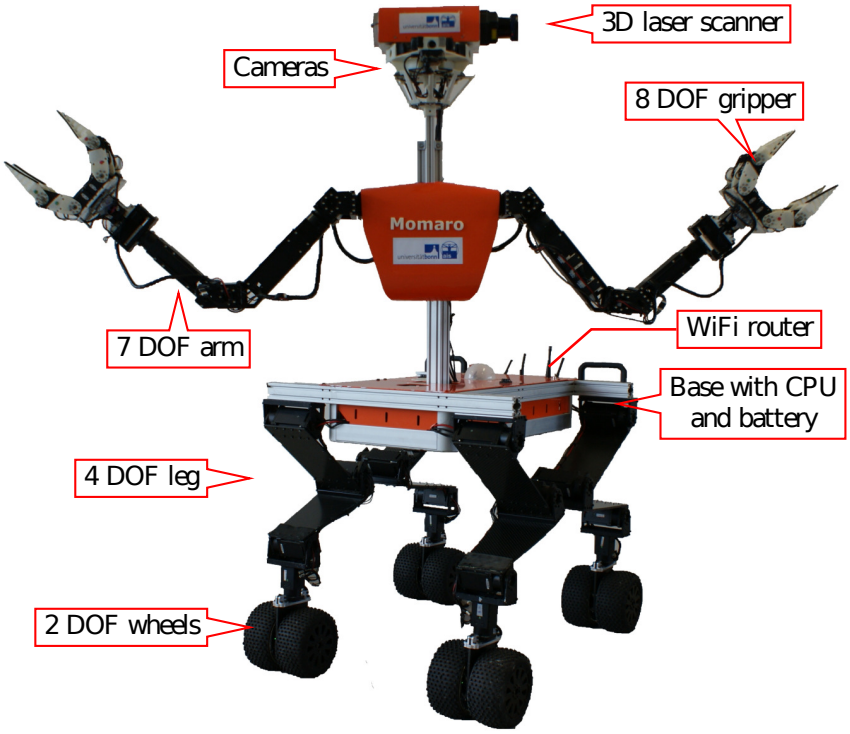
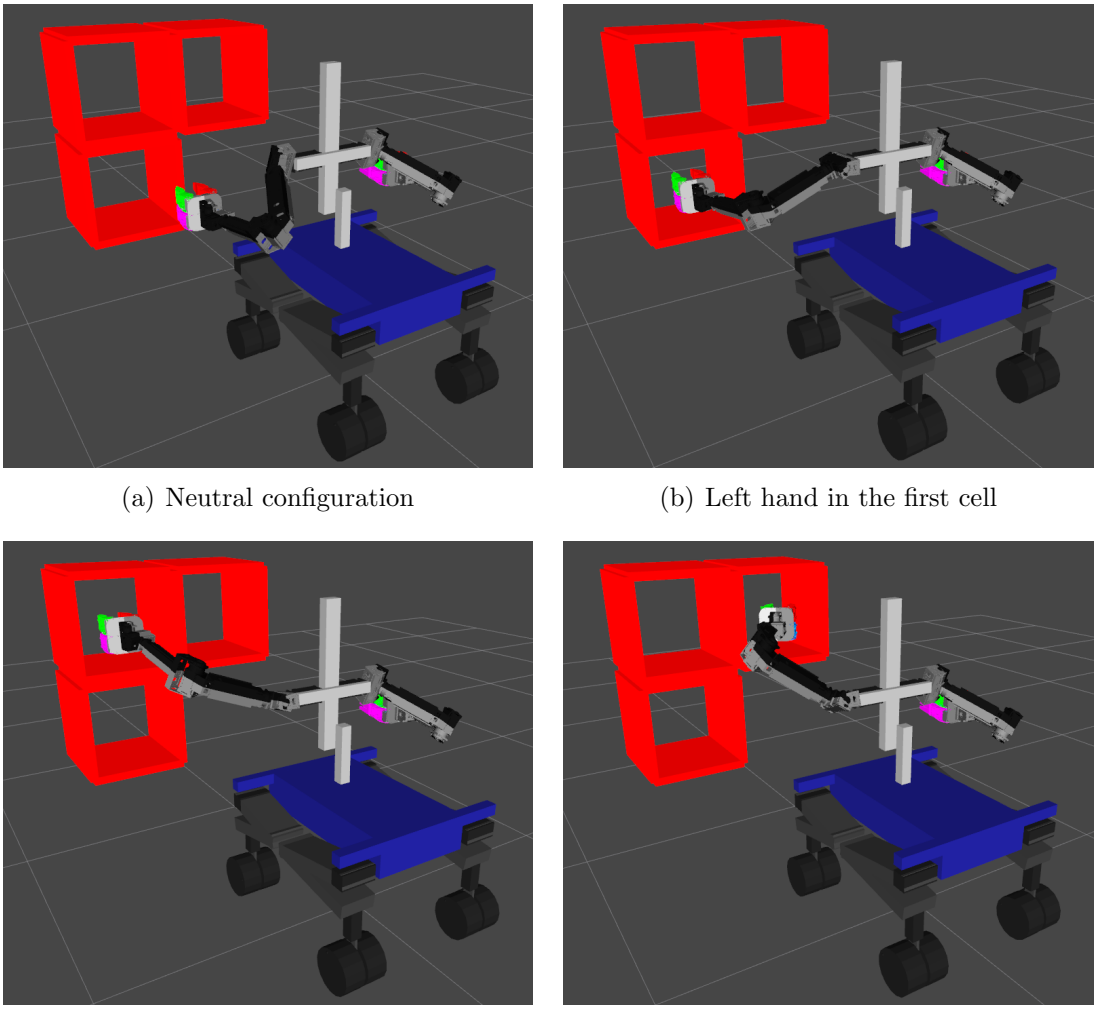


Figure 6.1: The Momaro robot, which is used for the experiments. The figure is taken from [37].

6.3.1 Shelf

The first experiment represents a range of tasks, when there are certain obstacles on the way, but obstacle-free area is quite large. It also represents a typical setup of manipulation tasks when there is an object standing on a surface and the task is to position the arm into a pre-grasp pose. An environment of this experiment consists of a shelf with three cells $35 \times 35 \times 35$ cm each. A thickness of shelf borders is 3 cm. The robot is standing in front of the shelf with an arm in a neutral position (Figure 6.2(a)). Except of the neutral configuration we also introduce three other, where the hand of the robot is located inside the first, the second and the third cell respectively. The scene as well as these configurations are shown in Figure 6.2.

This experiment consists of 12 tasks which are formed by all possible transitions between the four configurations described above. Each task is performed 100 times in order to obtain reliable results. For each execution we record the runtime, success/fail, and a trajectory length in joint space. Trial is considered to be successful if it is collision free and there are no violation of joint limits or any



(a) Neutral configuration

(b) Left hand in the first cell

(c) Left hand in the second cell

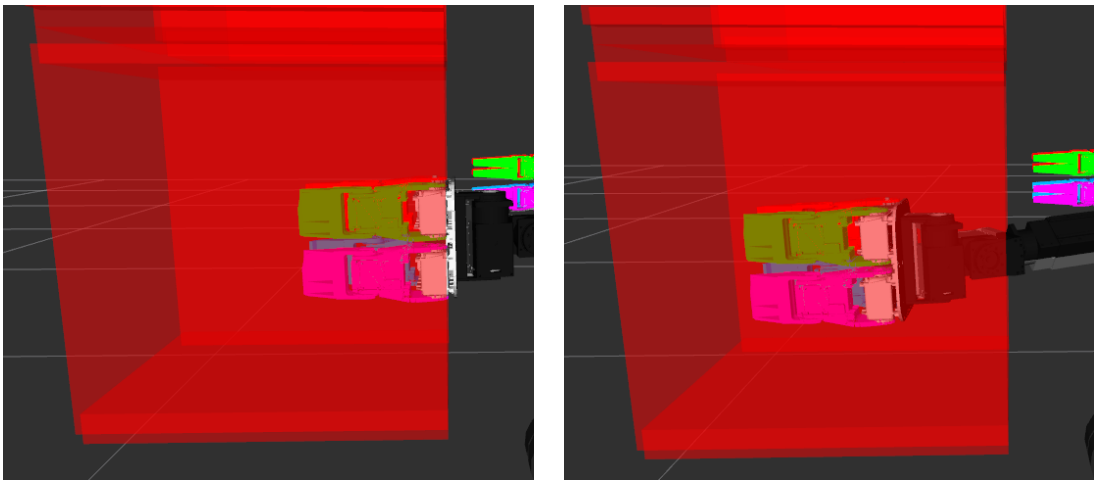
(d) Left hand in the third cell

Figure 6.2: Four configurations which form 12 possible tasks for the shelf experiments.

additional position/orientation constraints. When all trials were finished, average over these values are calculated.

In order to cover the shelf environment as much as possible, two more series of tasks of higher difficulty are performed. The set of configurations from above is referred to as “Easy”. To make the task harder, the gripper is immersed deeper into the cells, which produces a new configuration for each cell. The neutral configuration remains the same. This setup makes the task harder as the gripper must travel larger distance in a tight space of the cells. As in “Easy” experiment, there are 12 tasks available. We call this test as “Hard”. The difference between “Easy” and “Hard” configurations is shown in Figure 6.3. In “Hard” case the

gripper is immersed 11 cm deeper into the cell.



(a) Gripper immersion in "Easy" case. (b) Gripper immersion in "Hard" case.

Figure 6.3: Difference between "Easy" and "Hard" configuration sets. In "Hard" configurations set the gripper is immersed 11 cm deeper, which makes a planning problem harder to solve.

Finally, an orientation constraint for the gripper is introduced. We keep the same configurations as in "Hard" test, but now the gripper must hold the orientation along whole trajectory, as if it was holding a glass with liquid which is unwanted to be spilled. These motions are typically required for pick and place tasks when an orientation of an object in space matters. Pitch and roll are constrained to deviate no more than for ± 0.2 rad from an orientation of an initial orientation. This requirement makes the problem more challenging. We refer to this task as "Hard constrained".

We performed the tests described above using four algorithms: LBKPIECE, RRTConnect, STOMP-Industrial and our method which is refereed to as STOMP-New. The tests were preformed in simulation. We set the time limit for LBKPIECE and RRTConnect to be 5 seconds. If there is no valid solution found after this time elapsed, the trial is considered to be failed. We set the maximum iterations number for STOMP-Industrial and STOMP-New to be 100. At each iteration 10 trajectories are sampled. An initial noise standard deviation is set to 0.6 for both methods for each dimension. Trajectories optimized by our method have 10 keyframes between start and goal configurations, which results in 12 keyframes in total. We made our best in order to tune the algorithms so, that they demonstrate the best performance. While tuning of LBKPIECE and RRTConnect is fairly simple, we can not guarantee that we managed to achieve the best tuning

Table 6.1: Comparison of the success rate and average runtime of trajectory optimization in the shelf experiment. The algorithms which realizations did not support the features needed for the experiment are marked with "-" sign.

	Difficulty level					
	Easy		Hard		Hard constrained	
Algorithm	success rate	runtime [s]	success rate	runtime [s]	success rate	runtime [s]
LBKPIECE	0.94	2.47 ± 1.08	0.93	2.46 ± 0.85	-	-
STOMP-Industrial	0.87	0.87 ± 0.86	0.76	1.47 ± 1,01	-	-
RRTConnect	0.97	0.29 ± 0.18	0.96	0.85 ± 0.58	0.97	1.22 ± 1.04
STOMP-New	1.0	0.09 ± 0.02	1.0	0.18 ± 0.11	0.99	0.28 ± 0.21

of STOMP-Industrial, as it is not straightforward to do so. In this experiment we use simplified cost function in STOMP-New as other methods do not optimize the duration or motor torques. Thus, we do not use these costs in order to obtain a fair comparison. Both robot arm and torso yaw joint are involved in this task, which results in a configuration with 8 DOF, where 7 DOF are coming from the arm and 1 DOF - from the torso yaw joint respectively.

Overall, each test consists of 12 tasks, each task is attempted to be solved 100 times, which results in 1200 runs per test for each of the algorithms. The measurements obtained are shown in Table 6.1 and 6.2. There are no results for LBKPIECE and STOMP-Industrial for "Hard constrained" test as orientation constraints were not realized in these implementations.

In Table 6.1 the average runtime and success rate of each algorithm are shown. It is possible to see that algorithms except STOMP-Industrial and LBKPIECE succeeded in almost all trials achieving success rate close to 1.0. However, the average runtime differs significantly. The slowest method is LBKPIECE. It is noticeable that for both "Easy" and "Hard" difficulties LBKPIECE spend approximately the same amount of time. Meanwhile, other algorithms experienced growth of runtime with increasing difficulty. The second slowest algorithm in this experiment is STOMP-Industrial, which has a noticeable improve in runtime in comparison with LBKPIECE, having the lowest success rate at the same time. Our approach together with RRTConnect have shown the best performance, keeping high success rates and low runtimes. Our method achieved three or four times lower runtime than RRTConnect in average. The average time for distance field, utilized in STOMP-New computation was 0.033 s.

In Table 6.2 the average trajectory length in joint space for the "Easy" experiment is shown. LBKPIECE and RRTConnect do not perform an optimization of the trajectory during the planning itself. Instead, they pick the first valid so-

Table 6.2: Comparison of the lengths of the trajectories in joint space, obtained in the shelf experiment.

	LBKPIECE	RRTConnect	STOMP-Industrial	STOMP-New
Torso yaw	1.16	1.14	1.71	1.17
Shoulder yaw	2.37	2.18	1.87	0.96
Shoulder pitch	1.80	1.68	1.46	1.17
Elbow yaw	1.40	1.39	1.42	0.84
Elbow pitch	1.31	1.21	1.25	0.86
Wrist roll	4.41	4.53	2.03	0.83
Wrist pitch	1.18	1.10	0.83	0.44
Wrist yaw	1.06	1.00	1.03	0.62
$\sum$	14.69	14.23	11.63	6.89

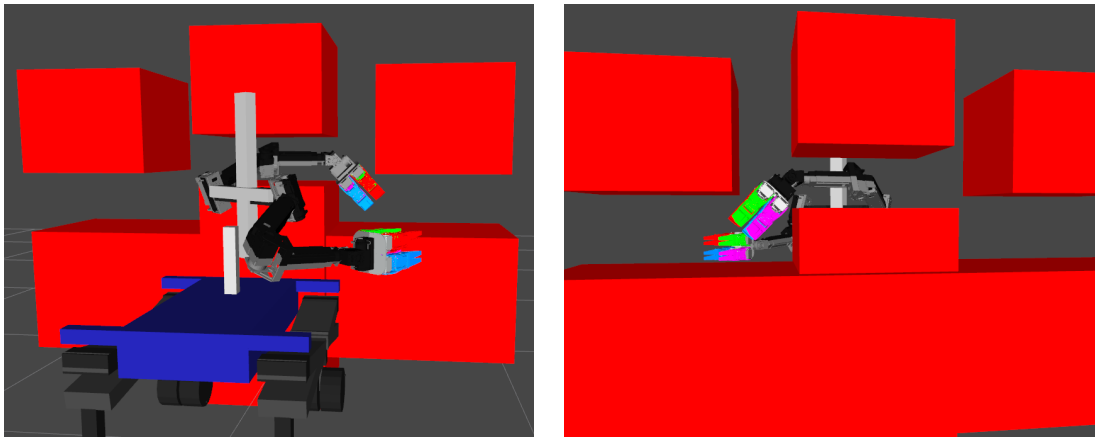
lution found. Trajectory smoothing and removal of redundant waypoints takes place in the postprocessing step. Thus, these algorithms produce relatively long trajectories which sometimes have unnecessary manoeuvres. At the same time, both STOMP implementations are optimization methods, hence their trajectories are shorter. Due to use of different obstacle cost functions in STOMP-Industrial and STOMP-New, the average trajectory length differs significantly. We observed that trajectories obtained by STOMP-Industrial tend to avoid the obstacles with large margins. There were no opportunity to tune the obstacle cost function of STOMP-Industrial as there are no tunable parameters of this function available in configuration files.

Overall, our approach demonstrated sufficient performance with respect to the success rate, as well as RRTConnect and LBKPIECE. STOMP-Industrial was outperformed by all other methods with respect to the average success rate. Our method shown shorter runtime, which is achieved by effective world representation as well as variable precision of collision checking.

6.3.2 Corridor

In this experiment we design a much harder task than previous one. While the previous experiment represented tasks of moderate complexity, the purpose of this experiment is to model a harder planning problem. With this task we are able to analyse the performance of compared algorithms in a context of close to worst-case difficulties.

An environment designed for this experiment consists from a narrow corridor between two large obstacles. An initial configuration is positioned outside the corridor, and a goal configuration is positioned directly inside the corridor (Figure



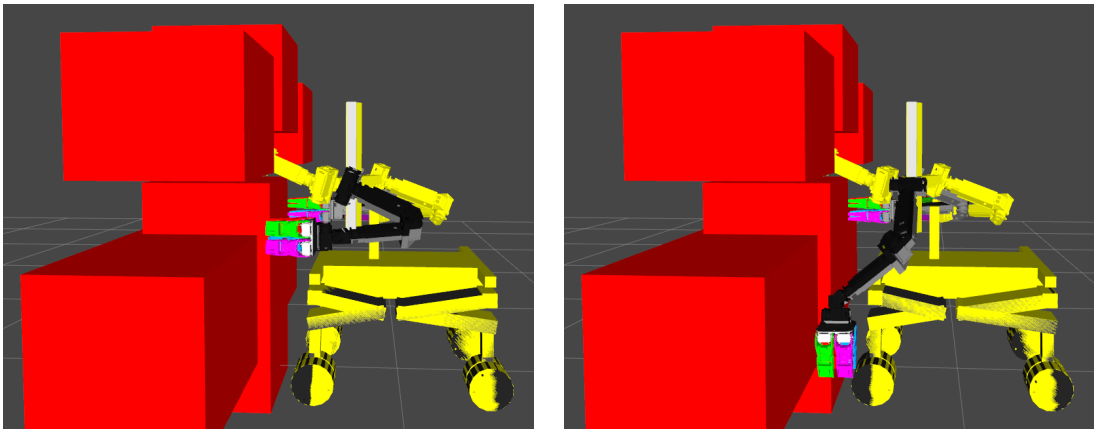
(a) Viewed from behind. (b) Viewed from the front.

Figure 6.4: Goal configuration for the corridor experiment viewed from different sides of the scene.

6.4). The obstacles are configured in a way that it is only possible to reach the goal configuration by following the corridor during the whole trajectory. It is possible to see that the corridor is narrow and the arm has very limited space in it. In the middle of the corridor there is a ledge, so that it is not possible to just rotate the torso, while keeping the arm in a static position. In order to reach the goal, the robot must overcome the ledge by performing a manoeuvre inside the corridor.

As in the previous experiment, several difficulty levels are designed. In this case they differ by the start configuration. We present two difficulty levels for this experiment, which are "Easy" and "Hard". The difference between them is demonstrated in Figure 6.5. In the "Easy" variation, the start configuration is chosen in a way that the arm is situated in front of the entrance to the corridor, which makes it easier to find a path for methods which start planning with some initial trajectory. In contrast, in the "Hard" variant, the arm is positioned very close to the bottom obstacle, in addition it is extended downwards and the gripper is rotated. In this scenario, a planning problem is much harder, as there is a larger distance to travel and an additional manoeuvre is required in order to reach the entrance into the corridor.

The parameters of the algorithms remain the same as in the previous experiment. The only difference is the time limit for LBKPIECE and RRTConnect. For this experiment we set it to be 10 seconds as the task is much harder. Iteration limit for STOMP-Industrial and STOMP-New is set to 200. The same 8 DOF are involved in this planning task. The experiment consists of 100 planning attempts for each difficulty for each algorithm. Obtained average runtime and success rate are shown



(a) "Easy" - the arm is in front of the entrance to the corridor.

(b) "Hard" - the arm is extended downwards and is pulled away from the entrance to the corridor.

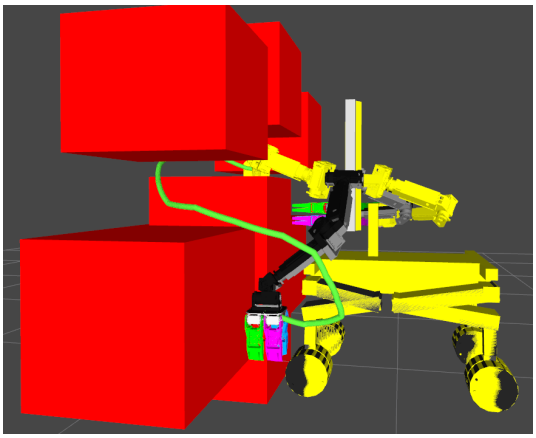
Figure 6.5: Two variants of the start configuration for the corridor experiment. Black: start pose; Yellow: goal pose.

Table 6.3: Comparison of the success rate and the average runtime of a trajectory optimization in the corridor experiment.

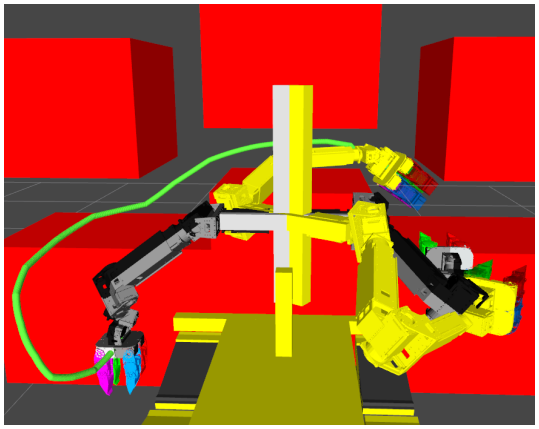
	Difficulty level			
	Easy		Hard	
Algorithm	success rate	runtime [s]	success rate	runtime [s]
LBKPIECE	0.65	6.97 ± 2.58	0.50	7.82 ± 2.58
RRTConnect	0.08	9.64 ± 1.27	0.06	9.71 ± 1.56
STOMP-Industrial	0.00	2.82 ± 0.07	0.00	2.85 ± 0.08
STOMP-New	0.78	1.89 ± 1.44	0.18	3.64 ± 1.29

in Table 6.3.

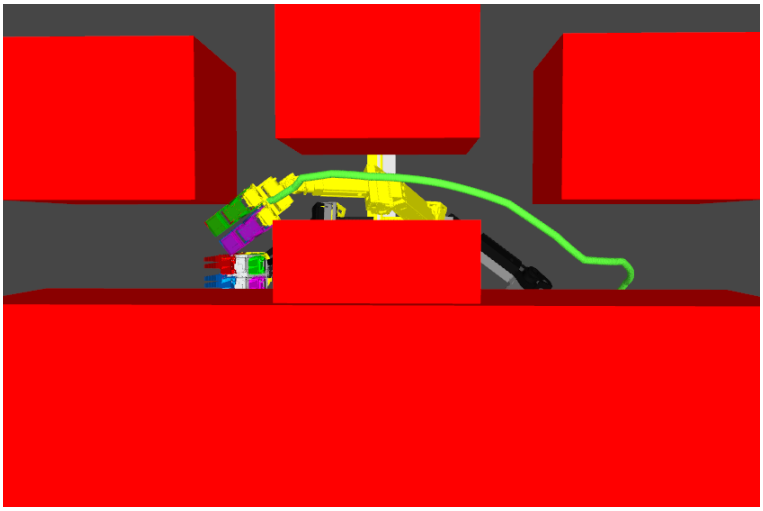
RRTConnect and STOMP-Industrial performed equally bad in both variations of the experiment with a success rate close to 0. For the "Easy" experiment, STOM-New shown the best success rate and average runtime. This is because in this scenario a linear interpolation which is used as initial trajectory is quite close to the desired path, so that STOMP-New is able to find a valid solution. LBKPIECE shown similar success rate, but the runtime was much larger. However, in the "Hard" scenario, relative performance of these two algorithms changed drastically. With a bad initialization STOMP-New was not able to find a valid solution reliably resulting in a low success rate of 18%. At the same time the performance of LBKPIECE decreased in much lower proportion, staying at a fair 50% level. The



(a) Viewed from the side.



(b) Viewed from behind.

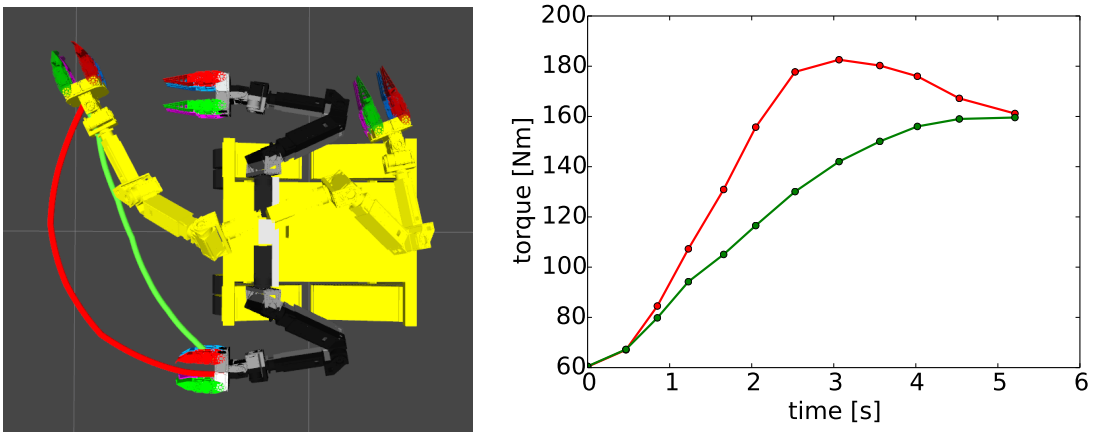


(c) Viewed from the front.

Figure 6.6: A feasible trajectory (green) for the corridor experiment in "Hard" variation, viewed from different sides of the scene. Black: start pose; Yellow: goal pose. Green line: trajectory of the end-effector.

example of a successful trajectory, obtained by STOMP-New for the "Hard" case is shown in Figure 6.6.

This experiment demonstrates that the performance of STOMP-New depends on the initial trajectory. Thus, in case of a hard tasks it does not provide a valid solution reliably, without being provided with an initial trajectory which is close to a feasible trajectory.. However, STOP-New shown a significant improvement in comparison to the STOMP-Industrial version which was not able to solve the problem at all. Overall, this experiment shows that our method is capable of solving hard tasks given an initialization close to a feasible solution.



(a) (b)

Figure 6.7: Comparison of trajectories obtained with/without torque optimization. The robot is assumed to hold 5 kilograms. Red: without torque optimization; Green: with torque optimization. **(a)** Trajectories of the end-effector. Black: start pose; Yellow: goal pose. With torque optimization the robot first rotates the torso and only than extends the arm, which results in lower torque. **(b)** Magnitude of the total torque. Without optimization (upper line) the torque grows faster and reaches unnecessary high values. While with optimization (lower line) total torque grows slower.

6.3.3 Duration and Torque Optimization

In this subsection we analyze the performance of a torque minimization as well as a new feature of our modification: trajectory duration optimization. In addition, we also demonstrate the capabilities of the system of flexible weights, which is utilized in our cost function.

Torque Optimization

In order to demonstrate how toque minimization influences the resulting trajectory, an optimization is performed. The initial configuration is a default position of the robot with bended elbow. A final configuration in contrast has fully extended arm and the torso is rotated in a direction of the arm extension. There are no obstacles in the scene, which ensures that nothing restricts the movement of the robot. The weight of the end-effector is being increased by 5 kilograms which imitates the heavy object being held by the robot. The optimization is performed two times: without torque minimization and with torque minimization, in order to emphasize the difference. The obtained trajectories are depicted in Figure 6.7.

It is possible to see that there is a significant difference between two obtained tra-

jectories. Without torque minimization the resulting trajectory moves uniformly all the joints towards the goal, as this makes the trajectory smooth and quick to execute. However, with torque minimization turned on, the trajectory follows other policy. The arm in extended forward state experiences higher torques due to the gravity force. That is why the optimizer avoids this effect by rotating the torso first and keeping the arm bended. When this movement is finished, the arm is extended by following the shortest path, which allows to overcome the problematic region quickly. For instance, such movement is natural for a human moving a heavy barbell.

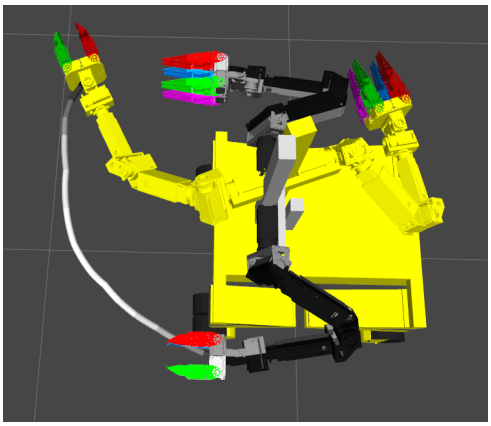
In Figure 6.7(b) one can observe that the trajectory (red) which is not optimized with respect to torques has total torque growing faster (upper line) than total torque of optimized trajectory (lower line), which happens due to completely extended arm. At the same time with torque minimization turned on, the total torque is growing slower.

Duration Optimization

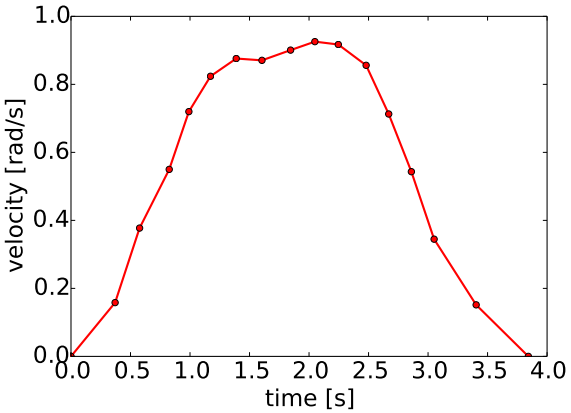
In order to demonstrate an effect of the duration cost component, an optimization for a simple obstacle-free task is performed firstly. The optimization is done two times: in the first case both initial and final velocities are set to 0 rad/s; in the second case, start velocity is set to 0.7 rad/s and final velocity is 0 rad/s. This simple experiment demonstrates an ability of our algorithm to be used not only for usual cases, when the robot is still in the beginning of the motion, but also for cases when the robot is already moving. This feature allows to use it in a frequent-replanning manner in order to solve tasks in dynamic environments. In this and the following demonstrations we consider continuous trajectory execution. The resulting trajectories as well as planned velocities are shown in Figure 6.8. The darker a segment of a trajectory is, the lower the velocity is.

One can observe that in both cases the velocity smoothly grows towards its maximum allowed value, which is 1.0 rad/s in this case. After that, it stays on this level for some time and than starts to go down until the target value is reached. The trajectory which starts with 0.7 rad/s velocity has smaller duration, than the trajectory which starts with 0 rad/s, which is an expected result. The trajectory with 0 velocity has duration of 3.81 seconds, meanwhile the trajectory with non-zero initial velocity has duration of 2.78 seconds, which results in a 1.03 seconds surplus.

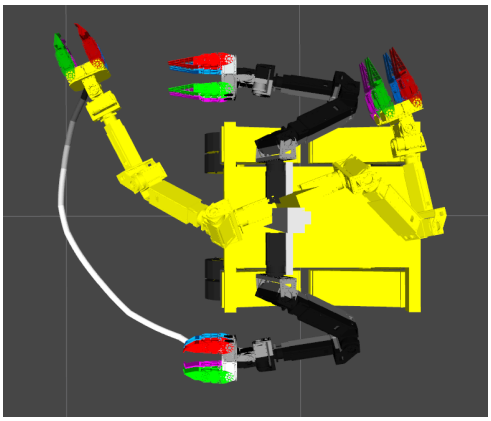
In order to show another interesting property of our cost function, we pick a task, similar to one from the shelf experiment. We consider a continuous control in this example as well. The velocity in the start, as well as in the end is 0 rad/s.



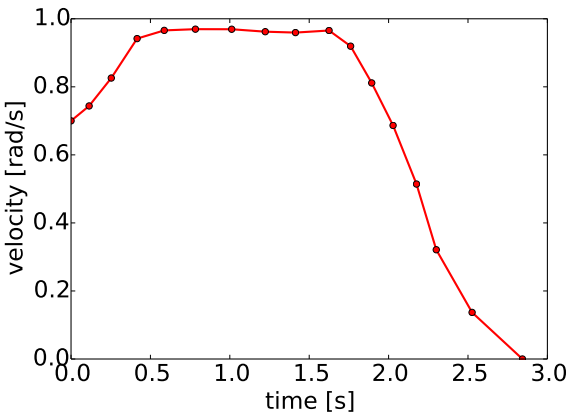
(a) Trajectory with both start and goal velocities being 0 rad/s.



(b) Velocity vs time for (a).



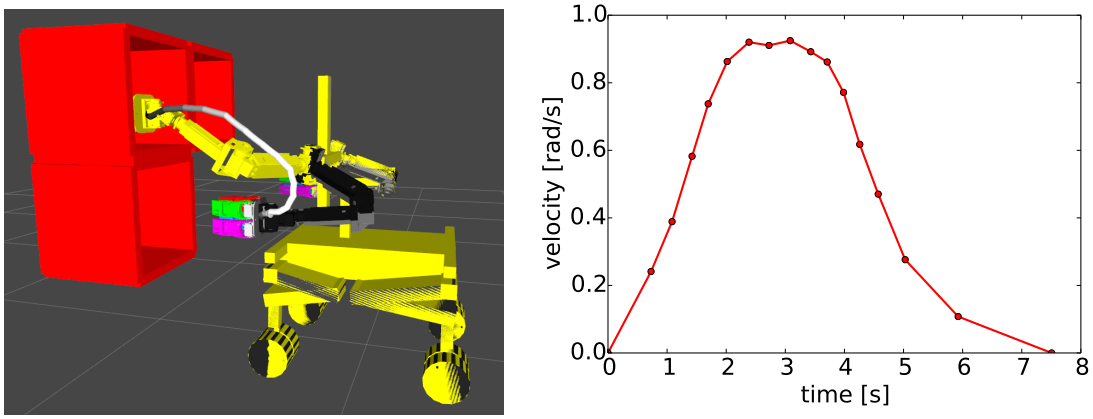
(c) Trajectory with start velocity being 0.7 rad/s and goal velocity being 0 rad/s.



(d) Velocity vs time for (c).

Figure 6.8: Example of a duration optimization for continuous trajectory execution. Black: start pose; Yellow: goal pose. The grey-scale lines represent the trajectories of the end-effector. The brighter the segment is, the larger is the velocity during that segment. In the first case (a) the initial and the goal velocity are 0 rad/s. In the second case (c) the initial velocity is 0.7 rad/s. In both cases the optimizer attempts to reach the maximum velocity and keep it as long as possible, before starting deceleration in order to minimize the overall duration of the trajectory. However, as in (c) the velocity is high initially, there is less time spent for acceleration, and hence, the overall duration is smaller.

The obtained trajectory is shown in Figure 6.9. It is possible to see that in the final part of the trajectory the robot must move close to the obstacle. In order to provide an additional level of safety, the optimizer decreases the velocity near obstacles, which can be seen as a dark region in the end of the trajectory.



(a) (b)

Figure 6.9: Duration optimization in a presence of obstacles. In the final part of the trajectory the robot is forced to move close to the obstacle. In order to make the movement safer, the optimizer chooses lower velocities in that region. This makes an earlier deceleration necessary, which results in a trajectory with longer duration, but with a higher safety level. **(a)** The grey-scale line represents the trajectory of the end-effector. The brighter the segment is, the larger is the velocity during that segment. It is possible to see darker segments in the end, which correspond to deceleration near the obstacle. Black: start pose; Yellow: goal pose. **(b)** Velocity vs time. It is possible to see a longer deceleration part in the end, which is the outcome of a required lower velocities in the end of the trajectory.

Cost Importance Weights

Finally, we demonstrate how all cost components work together orchestrated by cost importance weights. In the first example we take typical task from the shelf experiment described in Subsection 6.3.1. We change the weight of the obstacle cost component in order to obtain trajectories with different levels of safety. The obtained trajectories are shown in Figure 6.10. While the trajectory with the value of obstacle cost weight 1.0 is the safest, as it moves the arm very far from the obstacles, this trajectory involves the highest amount of movement and is potentially the longest to execute. At the same time the trajectory with lowest obstacle weight is the fastest to execute, but includes movements close to the obstacles. Note that this experiment is performed with a simple cost to demonstrate the behavior of the obstacle cost component. If done with full cost and non-zero duration importance weight, the algorithm would avoid the obstacle with decent margin, as movements close to the obstacles must be performed with low velocity, which affects the duration negatively.

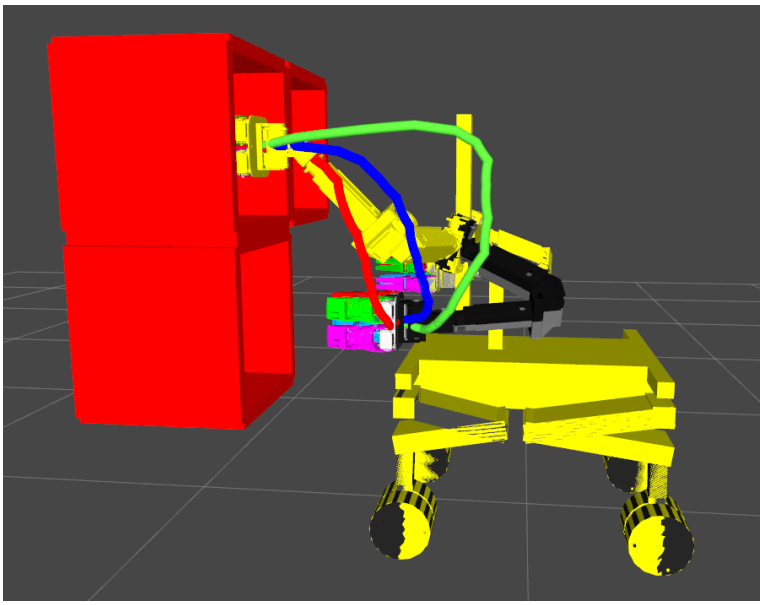


Figure 6.10: Trajectories obtained with different obstacle cost importance weight. Red: 0.0; Blue: 0.5; Green: 1.0. The larger the weight is, the larger distance to the obstacles is being kept by the robot. Black: start pose; Yellow: goal pose.

In the next example, we take the task described earlier in this subsection. This is the task for torque minimization, where the robot is holding a heavy object. We change the weights for the torque and duration cost components and observe different trajectories, which are shown in Figure 6.11.

With high torque importance cost weight and low duration cost weight, the trajectory is the safest for the robot. However it is also the slowest, as torque minimization avoids high accelerations, and as a consequence, high velocities. The trajectory with both high duration and torque weights sacrifices torque minimization in order to deliver the robot arm to its destination as fast as possible, but still preserves torque limits.

Overall, the duration and torque cost components shown that they influence trajectories in an expected way, allowing to achieve desired results. A system of flexible cost weights provides even more freedom to the algorithms of higher level, or to human operator in a sense that it is possible to choose different combination of weights depending on a particular situation in order to obtain trajectories with different features.

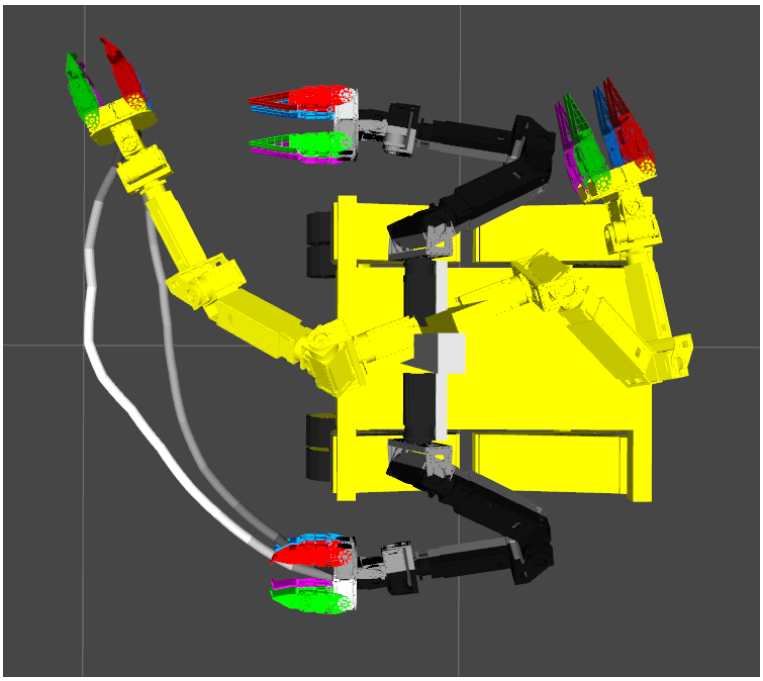


Figure 6.11: Trajectories obtained with different torque/duration cost importance weights. Left: 1.0 torque, 1.0 duration; Right: 1.0 torque, 0.0 duration. The grey-scale lines represent the trajectories of the end-effector. Black: start pose; Yellow: goal pose. The darker a segment of a trajectory is, the lower is the velocity during this segment.

Performance of Simplified and Full Cost

In order to estimate a runtime slow-down as well as possible success rate degradation when using full cost instead of simplified cost, we perform the shelf experiment one more time using full cost. To remind, the simplified cost only includes obstacle, joint limit and constraint components only and it is used in both shelf and corridor experiments. We perform the experiment in the same fashion as it was done first time. The comparison of the obtained runtimes is shown in Table 6.4.

A success rate remains the same as in the experiment with simplified cost, thus we do not show it in the table. However, there is a difference in runtime, caused by additional computations necessary to calculate additional cost components. For both "Easy" and "Hard" unconstrained tests, the runtime grew for around 30%. At the same time, in case of the test with orientation constraints, the runtime growth reached 71% which is explained by many disjoint local minima caused by the constraints, which requires more iterations to overcome. Overall, the runtime growth is not critical and do not limit the cases in which our modification may

Table 6.4: Comparison of average runtime for the shelf experiment using simplified and full costs.

	Difficulty level		
	Easy	Hard	Hard constrained
Simplified cost	0.09 ± 0.02	0.18 ± 0.11	0.28 ± 0.21
Full cost	0.12 ± 0.04	0.23 ± 0.19	0.48 ± 0.32
Mean runtime growth	33%	28%	71%

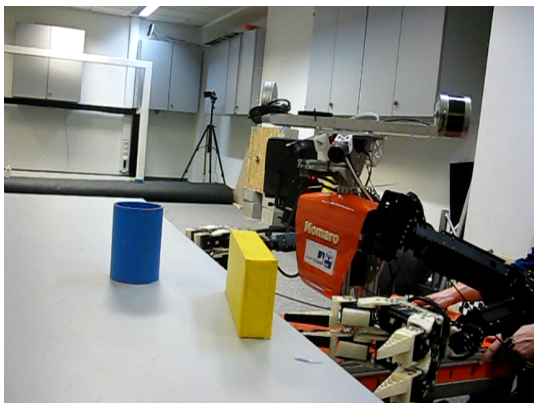
be applied. The runtime with full cost is still lower than runtime of compared methods which is shown in Table 6.1.

6.3.4 Real Robot

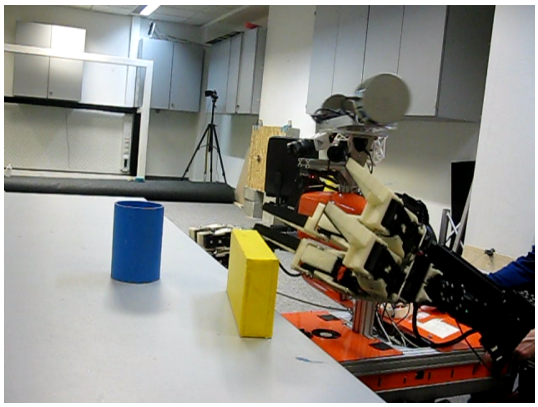
In order to prove that proposed method can be applied in reality, an experiment with a real robot was performed. As well as in simulation, the Momaro robot was used. An optimization with simplified cost only was performed, as fluid continuous trajectory execution was not implemented at the time when the experiment was performed. A distance field which represents the environment is computed from data which is obtained from a laser scanner which is situated on the top of the robot body.

The setup is shown in Figure 6.12(a). The robot is situated in front of a table, with opened left hand. The left arm of the robot is in a default configuration, which was used in previous experiments. The task is to reach a pre-grasp pose in order to consequently grasp the blue object. However, the arm is positioned below the surface of the table. Moreover, the occlusion of a direct way to the blue object is increased even more by the yellow object. Thus, the optimizer has to construct a manoeuvre in order to safely bypass the obstacles and successfully reach the blue object. The execution of a constructed trajectory is shown in Figure 6.12.

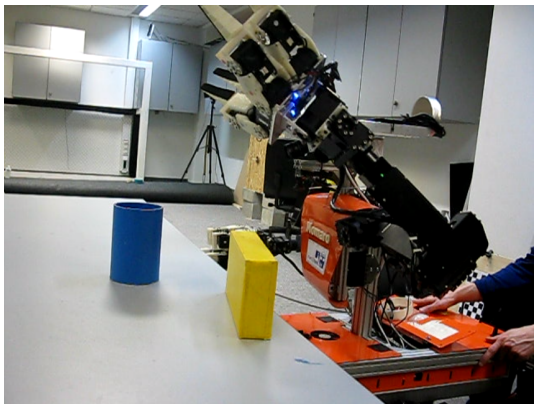
An interesting feature is how the robot uses the torso yaw in order to obtain a safer motion. It is possible to see in Figure 6.12(d) that the torso is rotated in a direction of the arm extension. This allows to occupy a safer position before reaching the pre-grasp pose and to minimize the risk of fingers colliding with the blue object while approaching the pre-grasp pose. In Figure 6.12(e) it is possible to see that the torso is rotated back to the neutral state, as it was defined by the goal position. Finally, in Figure 6.12(f) it is depicted how the object is being grasped and lifted from the table. Note, that the grasp motion and lifting motion were executed manually and were not part of the planned trajectory. Our method was used to plan the trajectory to reach the pre-grasp pose. Pre-grasp pose was



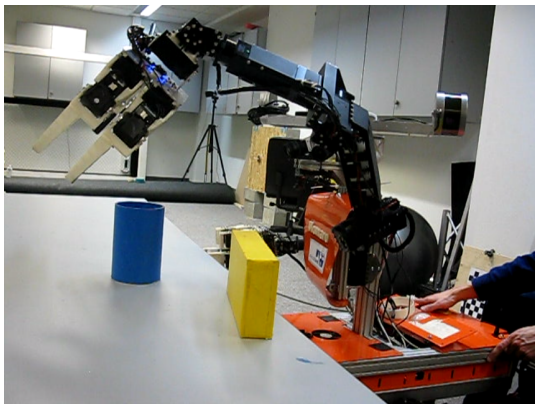
(a) Initial position.



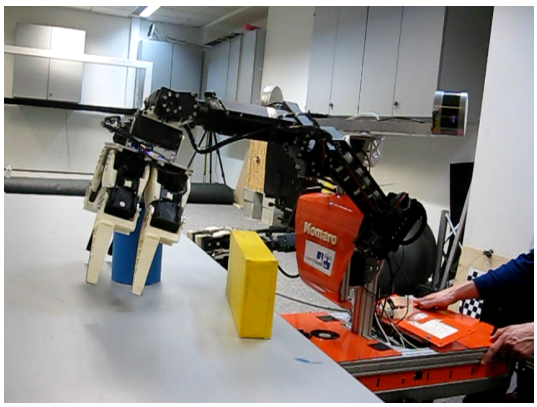
(b) Trajectory execution.



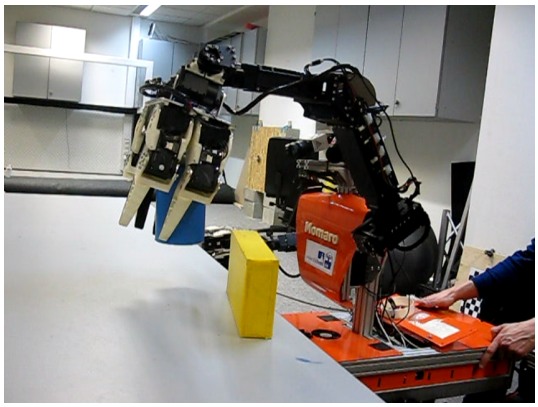
(c) Trajectory execution.



(d) Trajectory execution.



(e) Reached pre-grasp pose.



(f) Successfully grasped and lifted object.

Figure 6.12: The trajectory of the left arm executed on the real robot.

defined manually by a human.

This experiment shown that our method can be used on the real robot and is capable of planning a feasible trajectories which allow to successfully perform objective tasks.

6.4 Discussion

Overall, the results of the experiments shown that modifications which we introduced in this work improve the STOMP performance and capabilities. A decreased runtime gives an opportunity to utilize frequent replanning technique in order to act in dynamic environments. At the same time, an improved cost function allows to obtain more sophisticated solutions.

The first experiment with a shelf has shown that our modification of STOMP is capable of holding the same high success rate as RRTConnect and LBKPIECE have. At the same time, STOMP-New has a smaller runtime. Moreover, as STOMP is optimization-based method, final trajectories are smoother and do not include any redundant movements, which makes them to be ready for immediate execution on a real robot. This experiment demonstrated that for tasks with moderate difficulty STOMP-New is able to find reliably a feasible solution even when it is initialized with a naive straight interpolation between start and goal configurations.

Second experiment with a tight corridor demonstrates the importance of an initialization for the STOMP algorithm. For tasks with a high difficulty, initialization close to a feasible solution is needed for STOMP-New in order to solve a problem reliably. Our modification shown good performance with good initialization. Moreover, even with not feasible initialization it was able to find a solution in 18% cases. With both good and bad initialization, our modification shown better results than STOMP-Industrial. These two experiments demonstrate both strong and weak sides of our modification.

Finally, duration and torque cost components together provide additional possibilities for the planning. While they do not affect the runtime significantly, trajectories obtained with account for these components can be very helpful. In particular, these components are required to effectively perform tasks which require manipulations with large weights. Moreover, duration optimization allows to execute tasks faster, which is essential for any autonomous robot with limited battery power. However, one should be aware of the fact that such a complex cost which includes many components may lead to several different solutions of the same problem with equal cost. In order to reliably obtain solutions which have

desired properties, the costs should be prioritized distinctly with a help of cost weights, as it was shown in the last series of the experiments.

The experiments show that our modification of STOMP is a feasible algorithm, which can be used not only to optimize collision-free trajectories, but also to plan trajectories from scratch effectively. Additional cost components introduce higher flexibility and extend the range of tasks which can be solved by our method.

7 Conclusion

To conclude the results of our work, first we the summarise contributions of this thesis, then we discuss known limitations, and, finally, we present possibilities for the future work.

7.1 Summary

In this thesis a manipulation trajectory planning problem has been investigated. A method for efficient multicriteria arm trajectory optimization based on the STOMP [1] algorithm is proposed. The proposed method has been evaluated during series of experiments and its performance was compared against several planning algorithms.

In order to speed up a cost computation and introduce additional criteria for trajectory cost evaluation, the method of cost computation has been changed. In original STOMP a cost of a trajectory is a sum of costs of all keyframes which form the trajectory. This requires substantially large amount of keyframes in order to reliably cover whole trajectory. In our work we propose to compute the cost of the trajectory as a sum of costs of consequent transitions between keyframes. This allows to avoid unnecessary computations, increase reliability and incorporate additional criteria into the cost function. We introduce a duration cost which allows to optimize a duration of a trajectory. We add an additional dimension into configuration space, which represents a velocity of a joint with largest path within a trajectory. This velocity acts as an additional instrument for influencing the duration. As a result, the algorithm is capable of choosing optimal path and velocity in order to minimize the duration. The algorithm is capable of minimizing torque along the trajectory as well. This is of high importance for any task which involves manipulation with substantial masses.

We design our own cost function, which consists from following components: obstacle cost, constraint cost, torque cost and duration cost. In case when there is no severe violations, such as collisions or violations of joint limits, etc., cost component has a value in range $[0, 1]$, which allows to scale the relative importance of cost components using weights. This gives an opportunity to change proper-

ties of optimized trajectories based on a current situation and characteristics of performed task.

Proposed method was evaluated in simulation and its performance was compared against the following algorithms: RRTConnect, LBKPIECE and STOMP-Industrial. We evaluated several criteria, including: success rate, runtime, trajectory length and overall quality of a solution. Our method shown sufficient speed-up in comparison to other evaluated algorithms, meanwhile maintaining high success rate. The runtime was 4 to 5 times lower than the runtime of the fastest compared method. At the same time, the trajectories obtained by our approach have better quality, as it is optimization-based: trajectories obtained by our approach are smooth and are shorter than the ones obtained by compared methods which do not involve an optimization. In addition, we analyzed how the duration and torque costs influence optimized trajectories, and observed the expected effects. We measured the runtime growth caused by the additional costs and expansion of the configuration space by the velocity dimension. For unconstrained tasks, the observed growth is about 30% only, however in presence of orientation constraints, the optimization was performed up to 71% slower. Nevertheless, the average runtime of our approach is still smaller than average runtime of compared algorithms, and it should be possible to utilize frequent-replanning approach. In addition, we performed an experiment with a real robot, which shown that our method can be successfully involved in real-world missions.

Overall, the presented approach demonstrated good performance and shown several interesting effects in trajectory optimization. However, there are certain limitations, some of which can be mitigated in future work.

7.2 Limitations

As STOMP [1] is an optimization-based algorithm, our modification inherits this property. That is why the probability of finding a feasible solution as well as computational time necessary, heavily depends on the initialization. In our experiments we have shown that our modification may be used to plan from scratch in case of tasks of moderate difficulty, using naive straight interpolation between start and goal configurations as initialization. However, it was shown that for tasks of higher complexity, a success depends on a quality of the initialization.

Another inherited feature of the original STOMP is a way of approximating a robot. In order to speed up a collision checking, a set of spheres is used to represent a geometry of the robot. Usually it is not possible to represent all the details of robot geometry with a reasonable amount of spheres. Meanwhile in most

situations it is acceptable, in cases when high precision of a robot model is needed, this model may be insufficient.

In this work we assume that an environment around a robot, as well as the base of the robot itself remain static during a trajectory optimization and its consequent execution. This is insufficient in dynamic environments. Currently a possible way to overcome this problem is to utilize a frequent-replanning technique, which can be done according to estimated average runtime of our method.

In our modification a velocity for a single joint is optimized. Absence of velocities and accelerations for each joint makes the torque estimation less precise than when full velocities and accelerations are available.

7.3 Future Work

In order to mitigate some of the limitations discussed in the previous section, and to improve the overall performance of presented method, several possibilities for future work are proposed.

It is possible to increase a precision of a robot approximation by using more complex geometric primitives than spheres. This would allow to extend a range of tasks which can be solved reliably. Use of more complex shapes will increase a computational complexity of a collision checking, and, hence, the overall runtime. However, if the collision checking is performed using modern GPUs, the runtime growth can be minimized.

Another aspect of world representation which may be improved is a distance field, which represents a static part of the environment. Currently a fixed resolution is used for the whole distance field. However, same as in case of a robot approximation, in certain situations higher resolution is required. Unfortunately, if applied to the whole distance field, such an operation will increase exponentially the runtime needed to compute the distance field. In order to avoid this effect, it is possible to utilize a multiresolutional distance field, where regions close to obstacles have high resolution, meanwhile obstacle-free regions have lower resolution.

In order to make torque estimation more realistic, it is possible to estimate velocities and accelerations for all the joints. Given the optimized velocity for the joint x with longest path, one can use the duration necessary to execute a transition for joint x as a bound for execution of other joints which travel for the shorter path. This approach would allow to obtain more realistic velocities and accelerations for the torque estimation.

Current implementation of our method uses single core. However, a cost computation process may be parallelized, as cost of a trajectory consists of costs for

transitions between keyframes, which can be computed independently. Potentially, parallelization can decrease the average runtime of our method.

Overall, we consider several directions for possible improvement of our method which may allow to mitigate some of the limitations.

Bibliography

[1] M. Kalakrishnan et al. "STOMP: Stochastic trajectory optimization for motion planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 9-13, 2011.

[2] *CENTAURO – Robust Mobility and Dexterous Manipulation in Disaster Response by Fullbody Telepresence in a Centaur-like Robot*. `https://www.centauro-project.eu`.

[3] Dmitry Berenson et al. "Manipulation Planning with Workspace Goal Regions". In: *IEEE International Conference on Robotics and Automation (ICRA '09)*. May 2009.

[4] Rosen Diankov et al. "BiSpace Planning: Concurrent Multi-Space Exploration". In: *Robotics: Science and Systems*. June 2008.

[5] James Kuffner et al. "Motion Planning for Humanoid Robots". In: *Proc. 11th Int? Symp. of Robotics Research (ISRR 2003)*. Nov. 2003.

[6] Radu Bogdan Rusu et al. "Real-time Perception-Guided Motion Planning for a Personal Robot". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. St. Louis, MO, USA, Oct. 2009, pp. 4245–4252.

[7] *Combining Planning Techniques for Manipulation Using Realtime Perception*. Anchorage, Alaska, 2010.

[8] James J. Kuffner Jr. and Steven M. Lavalle. "RRT-Connect: An efficient approach to single-query path planning". In: *Proc. IEEE Int'l Conf. on Robotics and Automation*. 2000, pp. 995–1001.

[9] Steven M. Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. 1998.

[10] Dmitry Berenson et al. "Manipulation planning on constraint manifolds." In: *ICRA*. IEEE, Jan. 28, 2010, pp. 625–632.

[11] Ioan A. Şucan and Lydia E. Kavraki. "Kinodynamic Motion Planning by Interior-Exterior Cell Exploration". In: *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*. Ed. by Gregory S. Chirikjian et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 449–464.

[12] Lydia Kavraki et al. *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces*. Tech. rep. Stanford, CA, USA, 1994.

[13] Yuandong Yang and Oliver Brock. "Elastic roadmaps—motion generation for autonomous mobile manipulation". In: *Autonomous Robots* 28.1 (2009), p. 113.

[14] Pang C. Chen and Yong K. Hwang. "SANDROS: a dynamic graph search algorithm for motion planning". In: *IEEE Trans. Robotics and Automation* 14 (1998), pp. 390–403.

[15] O Khatib. "Real-time Obstacle Avoidance for Manipulators and Mobile Robots". In: *Int. J. Rob. Res.* 5.1 (Apr. 1986), pp. 90–98.

[16] Nathan Ratliff et al. "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2009.

[17] Sean Quinlan and Oussama Khatib. "Elastic Bands: Connecting Path Planning and Control". In: *In Proceedings of the International Conference on Robotics and Automation*. 1993, pp. 802–807.

[18] Oliver Brock and Oussama Khatib. "Elastic Strips: a Framework for Motion Generation in Human Environments Key Words—please Provide". In: 2003.

[19] Anca Dragan, Nathan Ratliff, and Siddhartha Srinivasa. "Manipulation Planning with Goal Sets Using Constrained Trajectory Optimization". In: *2011 IEEE International Conference on Robotics and Automation*. May 2011.

[20] Arunkumar Byravan et al. "Space-Time functional gradient optimization for motion planning". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., Sept. 2014, pp. 6499–6506.

[21] Ricarda Steffens, Matthias Nieuwenhuisen, and Sven Behnke. "Continuous Motion Planning for Service Robots with Multiresolution in Time". In: *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*. Ed. by Emanuele Menegatti et al. Cham: Springer International Publishing, 2016, pp. 203–215.

[22] Matthias Nieuwenhuisen et al. "Mobile Bin Picking with an Anthropomorphic Service Robot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2013.

[23] Chonhyon Park, Jia Pan, and Dinesh Manocha. "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments". In: *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*. 2012, pp. 207–215.

[24] Q.-Z Ye. "The signed Euclidean distance transform and its applications". In: *9th International Conference on Pattern Recognition* (Dec. 1988).

[25] John Schulman et al. "Motion Planning with Sequential Convex Optimization and Convex Collision Checking". In: *International Journal of Robotics Research (IJRR)* (2014).

[26] Frank E. Curtis and Michael L. Overton. "A sequential quadratic programming algorithm for nonconvex, nonsmooth constrained optimization". In: *SIAM Journal on Optimization* 22.2 (2012), pp. 474–500.

[27] Nikita Kitaev et al. "Physics-Based Trajectory Optimization for Grasping in Cluttered Environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[28] F. Reuleaux and A.B.W. Kennedy. *The Kinematics of Machinery: Outlines of a Theory of Machines*. Macmillan, 1876.

[29] J.J. Uicker, G.R. Pennock, and J.E. Shigley. *Theory of Machines and Mechanisms*. McGraw-Hill series in mechanical engineering. Oxford University Press, 2003.

[30] J. Denavit and R. S. Hartenberg. "A kinematic notation for lower-pair mechanisms based on matrices." In: *Trans. of the ASME. Journal of Applied Mechanics* 22 (1955), pp. 215–221.

[31] *IKFast: The Robot Kinematics Compiler*. `http://openrave.org/docs/latest_stable/openravepy/ikfast/#ikfast-the-robot-kinematics-compiler`.

[32] Dinesh Manocha Jia Pan. "Efficient Configuration Space Construction and Optimization for Motion Planning". In: *Engineering* 1.1, 46 (2015), p. 46.

[33] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society, series B* 39.1 (1977), pp. 1–38.

[34] Bengt Fornberg. "Generation of finite difference formulas on arbitrarily spaced grids". In: *Mathematics of Computation* 51.184 (1988), pp. 699–699.

[35] Peter Dayan and Geoffrey E. Hinton. "Using Expectation-maximization for Reinforcement Learning". In: *Neural Comput.* 9.2 (Feb. 1997), pp. 271–278.

[36] E. Theodorou, J. Buchli, and S. Schaal. "Reinforcement learning of motor skills in high dimensions: A path integral approach". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. May 2010, pp. 2397–2403.

[37] Max Schwarz et al. "NimbRo Rescue: Solving Disaster-Response Tasks through Mobile Manipulation Robot Momaro". In: *Journal of Field Robotics* (2016).

[38] Wisama Khalil. "Dynamic Modeling of Robots Using Newton-Euler Formulation". In: *Informatics in Control, Automation and Robotics: Revised and Selected Papers from the International Conference on Informatics in Control, Automation and Robotics 2010*. Ed. by Juan Andrade Cetto, Jean-Louis Ferrier, and Joaquim Filipe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–20.

[39] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5.

[40] *MoveIt: software for mobile manipulation*. `http://moveit.ros.org/`.

[41] *Lazy Bi-directional KPIECE with one level of discretization*. `http://ompl.kavrakilab.org/classompl_1_1geometric_1_1LBKPIECE1.html`.

[42] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). `http://ompl.kavrakilab.org`, pp. 72–82.

[43] *STOMP-Industrial*. `http://rosindustrial.org/news/2015/9/25/stomp-for-indigo-presentation-from-the-moveit-community-meeting-3-sept-2015`.

[44] *ROS Indigo Igloo*. `http://wiki.ros.org/indigo`.