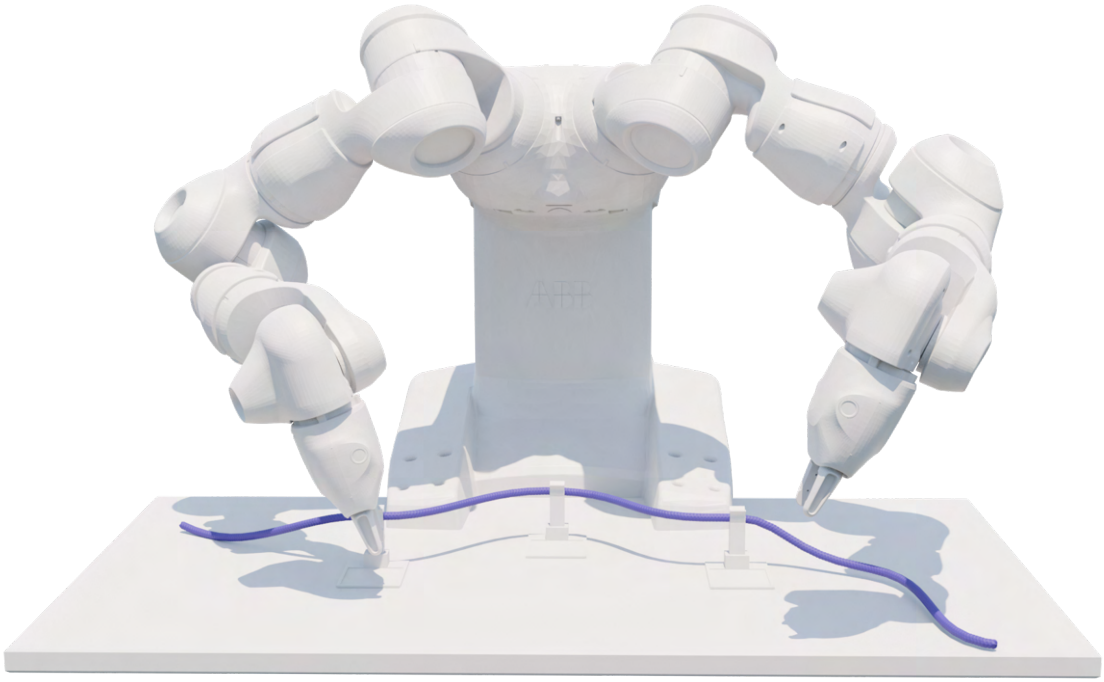


Planning and Control for Cable-routing with Dual-arm Robot

Autonomous cable routing with YuMi

Master's thesis in Systems, Control and Mechatronics

GABRIEL ARSLAN WALTERSSON

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Planning and Control for Cable-routing with Dual-arm Robot

Autonomous cable routing with YuMi

GABRIEL ARSLAN WALTERSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems, Control and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2021

Planning and Control for Cable-routing with Dual-arm Robot
Autonomous cable routing with YuMi
GABRIEL ARSLAN WALTERSSON

Supervisors:
Yiannis Karayiannidis, Electrical Engineering, Chalmers University
Rita Laezza, Electrical Engineering, Chalmers University
Examiner: Yiannis Karayiannidis, Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Division of Systems, Control and Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Rendition of ABB YuMi with the DLO and fixtures.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Planning and Control for Cable-routing with Dual-arm Robot
Autonomous cable routing with YuMi
GABRIEL ARSLAN WALTERSSON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Deformable object representation and manipulation is an active research area in robotics. Though there are many types of deformable objects, all with different properties, this work considers deformable linear objects (DLO), such as ropes or cables. A framework for solving cable routing or wire-harness problems with a dual-arm robot is proposed, with the objective of clipping a rope into several fixtures.
Dual-arm robots have two manipulators that are in close proximity and can collaborate on a single task. Building on inverse differential kinematics, two control methods are implemented. The manipulators can either be controlled individually as two separate entities or together as a single system. Hierarchical Quadratic Programming (HQP) is used to solve the inverse kinematics problem with feasibility control objectives. A computer vision system is implemented for both tracking the DLO in real-time with structure preserved registration and to estimate the poses of the fixtures. A path planner is developed, that can generate trajectory parameters to solve the cable routing problem. The path planner builds a roadmap from predefined tasks and the trajectory parameters are evaluated for any issues. If any issues arise, a genetic optimization algorithm is used to find a solution.
The system is tested with real-world experiments on an ABB YuMi robot. The results demonstrate successful cable routing through several fixtures and problem-solving capabilities.

Keywords: Deformable linear objects, Dual-arm robotics, coordinated manipulation, structure preserved registration, Cable routing, Stochastic optimization, Genetic algorithms.

Acknowledgements

I would like to thank both my supervisors Yiannis Karayiannidis and Rita Laezza for their continuous support and for giving me the opportunity to explore robotics. They have shown great interest throughout the project and always been curious about what I'm up to. I would also like to thank them for letting me steer the project in whatever direction I felt was interesting. It has been a truly enriching experience and an incredible journey.

Gabriel Arslan Waltersson, Gothenburg, June 2021

Contents

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Deformable Objects . . . 1
1.2 Related Works on Dual-arm Control . . . 2
1.3 Related Works on Deformable Object Tracking . . . 3
1.4 Related Works on Path-planning and Manipulation . . . 3
1.5 Problem Description . . . 4

2 Theoretical Background **7**
2.1 Control . . . 7
2.1.1 Quaternion . . . 7
2.1.2 Transforms and Rotation Matrices . . . 7
2.1.3 Joint-space and Operational-space . . . 7
2.1.4 Jacobian . . . 8
2.1.5 Inverse Kinematics . . . 8
2.1.6 Trajectories . . . 8
2.1.7 Convex Optimization . . . 9
2.1.8 Quadratic Programming . . . 9
2.2 Stochastic optimization (Sample based) . . . 10
2.2.1 Genetic algorithms . . . 10
2.3 Computer vision . . . 11
2.3.1 Image encoding . . . 11
2.3.2 Camera calibration matrix . . . 11
2.3.3 Apriltags/markers . . . 12

3 System Overview **13**
3.1 Hardware system overview . . . 13
3.1.1 Camera and environment . . . 14
3.1.2 Robot . . . 14
3.1.3 Gripper modification . . . 16
3.2 Software overview . . . 16
3.2.1 ROS Nodes . . . 17

4 Controller **19**

4.1 Kinematics . . . 20
4.2 Trajectory . . . 20
4.3 Hierarchical Quadratic Programming . . . 22
4.4 Control Objective . . . 24
4.4.1 Individual Manipulation . . . 25
4.4.2 Coordinated Manipulation . . . 26
4.5 Feasibility control objective . . . 29
4.5.1 Joint Velocity Limit . . . 30
4.5.2 Joint Position Limits . . . 30
4.5.3 Elbow Proximity Limit . . . 30
4.5.4 Joint Positions Potential . . . 31
4.6 Safety Checks . . . 31
4.7 Grippers . . . 32

5 Computer vision **33**
5.1 AprilTags . . . 33
5.1.1 Camera Pose . . . 33
5.1.2 Fixture pose . . . 33
5.2 Deformable Object Tracking . . . 34
5.2.1 Color Mask . . . 34
5.2.2 Point Cloud . . . 35
5.2.3 Structure Preserved Registration . . . 35

6 Path planner **37**
6.1 Initialization . . . 38
6.2 Trajectory Generation . . . 38
6.2.1 Evaluation . . . 39
6.2.1.1 `WithinReach` test . . . 40
6.2.1.2 `InsideFixtureSpace` test . . . 40
6.2.1.3 `TrajectoriesPassTooClose` test . . . 40
6.2.1.4 `GrippersCross` test . . . 40
6.2.1.5 `OverRotation` test . . . 40
6.3 Reactive Loop . . . 41
6.4 Tasks . . . 41
6.4.1 `GrabDLO` Task . . . 42
6.4.2 `ClipIntoFixture` Task . . . 43
6.4.3 `ResetOrientation` Task . . . 44
6.4.4 `SolverRerouting` Task . . . 44
6.4.5 `HoldPose` Task . . . 45
6.5 Stochastic Solver . . . 45
6.5.1 Individuals . . . 46
6.5.2 Generate initial population . . . 47
6.5.3 Evaluation fitness score . . . 48
6.5.4 Mutation . . . 51

7 Results **53**
7.1 Apriltags . . . 53

7.2 DLO tracking 53
7.3 Controller 54
7.4 Cable routing 58
7.5 Failure states and limitations 62

8 Conclusion **65**

Bibliography **67**

List of Figures

2.1 Cross over 10

3.1 A figure of the hardware and how it is connected 13
3.2 A figure of the fixture, camera and DLO 14
3.3 Rendition of YuMi with some key frames 15
3.4 Rendition of YuMi with joint numbering 15
3.5 Smart gripper and grip pads 16
3.6 Simplified overview of the software system 17
3.7 Overview over the relevant ROS nodes. 17

4.1 Block diagram for controller. $\mathbf{q}$ is the joint position, $\dot{\mathbf{q}}$ the joint velocities, $\mathbf{J}(\mathbf{q})$ the Jacobians, $\mathbf{x}(\mathbf{q})$ the forward kinematics, $\mathbf{x}_d(t)$ and $\dot{\mathbf{x}}_d(t)$ the desired position and velocity from the trajectory and $\mathbf{A}, \mathbf{b}, \mathbf{G}, \mathbf{h}$ are the constraints for the solver. 19

5.1 Tracking DLO overview 34

6.1 Path planner states, overall structure 37
6.2 Rendition of how the grip points are calculated 42
6.3 Turning around DLO with intermediate step 46
6.4 Rendition of the range for the solver grip points 47
6.5 Zones for optimization, `individual manipulation` 48

7.1 DLO estimation with SPR 54
7.2 `Individual manipulation` position and trajectory 55
7.3 `Individual manipulation` position error 55
7.4 `Individual manipulation` orientation error 56
7.5 `Coordinated manipulation` position and trajectory 57
7.6 `Coordinated manipulation` position error 57
7.7 `Coordinated manipulation` orientation error 58
7.8 Clip into first fixture 59
7.9 Clip into first fixture internal state 59
7.10 Clip into second fixture 60
7.11 Clip into third fixture 61
7.12 Fail to clip into second fixture 63
7.13 DLO occluded by robot arm 63

List of Tables

3.1 PC specification 13
3.2 Joint limits 16

4.1 HQP hierarchy 24

5.1 HSV values 34

6.1 Evaluation of trajectory parameters 39
6.2 Motion primitives 43
6.3 Motion primitives 44
6.4 Motion primitives 44
6.5 Motion primitives 45
6.6 Motion primitives 45
6.7 Parameters 46
6.8 Individuals 47
6.9 Evaluation for *individual manipulation*, fitness score 49
6.10 Evaluation for *coordinated manipulation*, fitness score 50
6.11 Mutation STD 51

7.1 Fixture position estimation 53
7.2 Error position $[m]$ 58
7.3 Error angular $[rad]$ 58
7.4 Pathplanner states during initialization 59
7.5 Pathplanner states for the first fixture 60
7.6 Pathplanner states for the second fixture 61
7.7 Pathplanner states for the third fixture 62

1
Introduction

Robotics has a wide variety of use cases, with applications in a large number of places and sectors. Today robotics is widely used to automate repetitive assembly tasks and process manufacturing. But the field of robotics is predicted to keep growing and become more integrated into our society. For the companies, it is believed that robotics will have a major impact on the future economy and competitiveness for manufacturing [1]. But robotics is also moving outside of the manufacturing sector, into other sectors such as health care, consumer products and agriculture. As robotics move to other sectors, they are expected to perform new types of tasks with a new set of requirements. This thesis explores the area of deformable objects. While it is trivial for humans to manipulate deformable objects, it is challenging for robotics. This work deals with deformable linear objects, such as ropes or cables. A framework for solving cable routing or wire harness problems with a dual-arm robot is proposed.

1.1 Deformable Objects

We encounter deformable objects daily, everything from clothes to food items. In the industry and service sector, deformable objects can be found in packaging and folding clothes, cable routing and wire harness assembly, repair work and medical/surgery treatments etc. Deformable objects can broadly be classified into 4 groups [2], linear, planar, cloth-like and solid/volumetric. Linear objects are uniparametric, objects of this type have one dimension significantly larger than the other two. Objects that belong to this category are ropes, strings, cables and beams etc. Planar objects are biparametric, they have one dimension significantly smaller than the other two. Objects in this category are paper, metal sheets and thin-walled objects. Solid/volumetric objects are triparametric and have no single dimension significantly larger or smaller. Objects of this type are food items, sponges and any deformable solid object. Though cloth-like objects can have the same dimensional properties as planar or volumetric objects, the distinction is that cloth-like objects have very little strain or compression strength. Objects of this type are shirts, pants and other objects made from fabric like materials.
Deformable objects have some unique challenges related to robotics compared with rigid bodied objects. Rigid body objects are objects that have a fixed shape. The pose (position and orientation) of a rigid body can be described with a finite number of dimensions, namely 6. Deformable objects, in contrast often requires an infinite state space to represent the exact pose. Realistically the representation has to be

approximated, there are many proposed approximations with various advantages and disadvantages. Usually, the trade-off is between accuracy versus computational complexity. Beyond the representation complexity, tracking deformable objects from computer vision is non-trivial as the shape is not fixed and self-occlusion/ occlusion from the robotic arms introduce further uncertainties. Manipulation of deformable objects is challenging, as they are usually heavily under-actuated where very few states can be directly controlled. For example, a dual-arm robot may only have direct control over two points of the object and to achieve certain configurations may be impossible or require a sequence of manipulations. The complexity in accurately modelling and predicting deformable objects add further challenges to path planning.

1.2 Related Works on Dual-arm Control

A dual-arm robot can be said to be a robot with two robotic manipulators or two robots in close proximity controlled by a single control system. The advantage of a dual-arm robot is that it can perform tasks where separate manipulators are collaborating. For deformable object manipulation, which is already an underactuated system, this is a significant advantage. There are two main approaches for utilizing a dual-arm robot, either each arm is seen as a separate entity or they can be controlled as a single system. A dual-arm robot also has the advantage of being similar to a human in structure, lending the possibility of utilizing already existing workspaces. Though some challenges with dual-arm robots exist, such as the increased number of joints leads to increased degrees of freedom (DOF) and designing control algorithms that efficiently can utilize both manipulators for a single task.

Lewis *et al.* [3] formulates on-line trajectory generation for two cooperating robots performing a single task. A relative Jacobian is formulated that relates the relative motions between the robots. The relative Jacobian allows both manipulators to be solved as a single redundant system. The paper also formulates constraints for obstacle avoidance, joint limits and absolute motion for the end effectors. Using a similar approach [4] formulates the inverse kinematics with a relative and absolute Jacobian. Where the absolute Jacobian maps the joint velocities to the average of the two end effectors and the relative Jacobian as the difference between the end effectors. Allowing tasks to be performed by defining absolute and relative variables. For using the relative Jacobian approach for tasks with force control and high angular velocities, [5] included a wrench transformation matrix and showed improved results. Asymmetric coordinated motion [6] can be achieved by extending the absolute and relative Jacobian. Allowing for a smooth transition between symmetric and asymmetric (master-slave) execution of tasks. For solving the inverse kinematics problem, [7] showed how hierarchical quadratic programming (HQP) could be used for a humanoid robot to achieve complex tasks. Both equality constraints and inequality constraints can be solved in a strict hierarchy, where the lower priority constraints can be relaxed if necessary.

1.3 Related Works on Deformable Object Tracking

One of the most challenging parts of deformable object manipulation is the estimation of the object. Besl *et al.* [8] showed how an iterative closest point method could be used for matching two rigid body point clouds without known correspondence. Chui and Rangarajan [9] showed how Gaussian mixture models (GMM) could be used for non-rigid registration for point cloud matching. Building on Gaussian mixture models the coherent point drift (CPD) method [10] forces the GMM to move coherently to preserve the topological structure. For non-rigid objects, the coherence constraint was achieved by global regularization of the object. Building upon CPD and other work on global and local regularization, Tang *et al.* [11] showed a real-time tracking algorithm for deformable objects called structure preserved registration (SPR), that could deal with occlusions. SPR uses GMM in combination with local and global regularization together with a physics simulator to estimate deformable objects. The purpose of physics simulator is to ensure that the estimation is physically feasible. The output from the simulator is used as the initial state for the next SPR estimation. The paper demonstrated successful tracking of both linear and cloth-like objects. Including an experiment on cable routing using the planner from [12]. [13] showed some of the earliest working results of tracking highly deformable objects such as ropes and cloth-like objects. It is based on an expectation-maximization algorithm, modified to make use of a physics simulator for core parts of the computation. Showing a real-time tracking algorithm for linear, planar and volumetric objects. In [14] Tan *et al.* used a different approach and used a learning-based approach for predictive modelling of deformable objects. Using contrastive learning their method only required random data samples for training, making the transfer from simulation to real-world easier.

1.4 Related Works on Path-planning and Manipulation

Manipulation of deformable objects has shown to be challenging. A common scenario for research is to solve knotting problems or shape control. In [15], Zhu *et al.* have focused on achieving a desired shape of a flexible cable between two grippers. The method used was model-free and uses Fourier series to represent the shape. In [12], Tang *et al.* proposes a framework for solving knotting problems. The framework used visual feedback with an RGB-D camera. RGB-D cameras contain both colour and depth information. Their task planner used a library of pre-recorded states and trajectories. From the state estimation, they could identify which of the pre-recorded states was closest to the observation. The transformation between the observed state and the pre-recorded state was calculated and applied to the trajectories before execution. They successfully demonstrated the framework experimentally. Also using prerecorded trajectories, Kudoh *et al.* [16] showed a framework for manipulating deformable objects using a single human demonstration of the task.

The recorded motion/force trajectories were used as the reference for a compliant controller. In [17], Kudoh *et al.* showed in air knotting of a rope using reusable hand motion primitives. The dual-arm robot had three fingers on each hand. In [18] Saha *et al.* describes a motion planner that for manipulating DLOs and solving knot problems. The planner uses a topological state representation in contrast to a geometric one. The planner constructs a probabilistic roadmap by sampling nodes towards the goal and checks if each node is a valid move or not. A sampling method in configuration space with narrow passages was proposed in [19]. Where the points for the road map was uniformly sampled at the surface of the obstacles in configuration space, improving the probability of finding paths through narrow passages. In [20], Duenser *et al.* showed carving shapes from solid foam blocks with a deformable hot wire using a dual-arm robot. The deformable properties of the cutting wire allowed for fewer cuts and concave shapes.
A method for solving for the entire shape space when planning was proposed in [21]. The DLO was represented as minimal-energy curves. The paper proposed a local planner that can move from one minimal energy state to another while all intermediate states are also minimal energy curves. That makes it possible to remain in a minimal-energy state while following for example a sample-based global roadmap. As the object always is in a minimal-energy state, the entire shape space is known at all times. In [22] Rousse *et al.* showed a sampling-based method based on Rapidly-exploring Random Trees can be used to solve planning problems for deformable linear objects with environmental contacts. The planner was validated in a realistic simulator with a solve time limit of 30 minutes.
In [23] Pires *et al.* show how genetic algorithms can be used to solve both the inverse kinematic problem and simple path planning. They discretized a subset of all the possible paths and the individuals were encoded with each gene representing a "move". Genetic algorithms have also been used for path planning directly in configuration space [24] and [25] with usual criteria to minimize energy and collision-free trajectories. Genetic algorithms are common in mobile robots for path planning and optimization [26], [27], [28] and [29].

1.5 Problem Description

The objective of this thesis is to solve a cable routing problem with a dual-arm robot. A DLO is placed arbitrarily on the workspace with several fixtures. The DLO should be picked up by the robot and then clipped into the fixture. The problem can be divided into three parts, control, computer vision and planning.
There are several challenges related to the controller. To clip the DLO into a fixture, both grippers have to collaborate and manipulate the cable simultaneously. The robot also has physical constraints, there are joint position and velocity limits and self collision should be avoided. The system is over redundant as there are more joints than DOF controlled, allowing multiple solutions for the same end-effector pose.
For computer vision, a single camera with both color and depth sensors is used. The vision system should be able to estimate the pose of the fixtures relative to the robot and track the DLO in real-time. Tracking the DLO has multiple challenges

as it is a deformable object and parts of the DLO can be occluded.
The planner generates instructions to solve the cable routing problem. Some of the challenges are related to the DLO as it is deformable, which makes it difficult to accurately predict. When the DLO is clipped into a fixture, then the DLO acts as a physical constraint for all sequent manipulations, limiting the allowed motions. There are challenges in deciding where to pick up the DLO and if those are within reach and out of collisions. Many potential unforeseen situations can occur during manipulation and the planner should be able to solve the most common ones.

2
Theoretical Background

In this chapter, the theoretical background for the project is presented. Robotics is multidisciplinary and blends theory and methods from a wide variety of fields such as computer science, control theory, electrical- and mechanical engineering.

2.1 Control

In this section, the theory used in the controller is described.

2.1.1 Quaternion

Orientation can be represented in multiple ways. Two common representations are Euler angles and quaternions. This thesis uses quaternions [30] on the form $Q = \begin{bmatrix} ix & iy & iz & w \end{bmatrix}$. Quaternions have the advantage that each quaternion represents a unique orientation, while multiple Euler angles can represent the same orientation. Adding the effect of two quaternions are done by quaternion multiplication. The inverse of a quaternion is done by taking the conjugate i.e. the imaginary parts are negated.

2.1.2 Transforms and Rotation Matrices

It is useful in robotics to describe an object pose with a coordinate frame. The transformation to a frame expressed in another base frame can be written with a transformation matrix, see equation (2.1). The transformation matrix consists of a rotation matrix $\mathbf{R}^{\text{base}}_{\text{frame}}$ and the translation to the frame $\mathbf{o}^{\text{base}}_{\text{frame}}$.

$$\mathbf{T}^{\text{base}}_{\text{frame}} = \begin{bmatrix} \mathbf{R}^{\text{base}}_{\text{frame}} & \mathbf{o}^{\text{base}}_{\text{frame}} \\ \mathbf{0}^{\text{T}} & 1 \end{bmatrix} \tag{2.1}$$

2.1.3 Joint-space and Operational-space

Joint space also known as configuration space is defined by the joint variables [31]. A manipulator can have q_{n} joints, the space is then defined by the vector:

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ . \\ q_{\text{n}} \end{bmatrix} \tag{2.2}$$

Tasks are usually defined for the end effector in operational space, also called work/-task space. For a manipulator the end effectors pose can be described in operational space, with a position p_e and an orientation ϕ_e. The orientation can be represented as a quaternion. The end effectors pose $\mathbf{x}_e$ is then described in operational space by the vector:

$$\mathbf{x}_e = \begin{bmatrix} p_e \\ \phi_e \end{bmatrix} \tag{2.3}$$

The direct kinematics equation maps joint space to operational space.

$$\mathbf{x}_e = k(\mathbf{q}) \tag{2.4}$$

From the joint positions, it is then possible to calculate the pose of the end effector.

2.1.4 Jacobian

The differential kinematics relates the joint velocities in joint space to the end effectors velocities in operational space. The analytical Jacobian $\mathbf{J}$ can be calculated by:

$$\mathbf{J}_{ij}(\mathbf{q}) = \frac{\partial k_i(\mathbf{q})}{\partial q_j} \tag{2.5}$$

The Jacobian is a $m \times n$ matrix where m is number of variable in operational space and n is the number of variables in joint space. The relation between end-effector velocities and joint velocities can then be expressed as:

$$\dot{\mathbf{x}}_e = \mathbf{J}(\mathbf{q}) \cdot \dot{\mathbf{q}} \tag{2.6}$$

2.1.5 Inverse Kinematics

Inverse kinematics maps the end-effectors pose to joints position. For simple manipulators, this is possible to calculate analytically, while more complex manipulators can have more than one and sometimes an infinite number of solutions to the problem. To simplify the calculations it is common to instead only control the velocities. That leads to inverse differential kinematics which maps end-effector velocities to joint velocities, given by:

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1} \cdot \dot{\mathbf{x}}_e \tag{2.7}$$

If the Jacobian is square and full rank, then the inverse differential kinematics can be calculated by the inverse of the Jacobian. If the Jacobian is not square the inverse kinematic can instead be solved as an optimization problem. For example, the pseudo inverse [32] gives the least squares solution to the problem where all parameters are equally weighted.

2.1.6 Trajectories

The motion for the end-effectors can be defined in operational space with trajectory parameters. The trajectory parameters define the initial state, final pose and also

if there are any intermediate poses [31]. One way to determine the trajectory for one dimension between two points is to fit a cubic polynomial, the polynomial then describes the intermediate points. Using notation for joint motion, though it applies for any one-dimensional trajectory. The position over time is written as a cubic function:

$$q(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \tag{2.8}$$

and the derivative:

$$\dot{q}(t) = 3a_3t^2 + 2a_2t^1 + a_1 \tag{2.9}$$

If the initial position q_i and velocity $\dot{q}_i$ as well as the final position q_f and velocity $\dot{q}_f$ is determined, then all the parameters can be solved by solving the equation system:

$$\begin{aligned} a_0 &= q_\mathrm{i} \\ a_1 &= \dot{q}_\mathrm{i} \\ a_3t_\mathrm{f}^3 + a_2t_\mathrm{f}^2 + a_1t_\mathrm{f} + a_0 &= q_\mathrm{f} \\ 3a_3t^2 + 2a_2t^1 + a_1 &= \dot{q}_\mathrm{f} \end{aligned} \tag{2.10}$$

2.1.7 Convex Optimization

Convex optimization is a optimization medthod that minimizes a convex function over a convex set. A set $\mathbf{S} \subseteq \mathbf{R}^n$ is convex if for any two points $x_1, x_2 \subseteq \mathbf{S}$ the entire line segment $l(x_1, x_2)$ connecting them also is within the set, i.e. $l(x_1, x_2) \subseteq \mathbf{S}$ [33]. Likewise a function f is convex if for any $x_1, x_2 \subseteq \mathbf{S}$ and for any $0 \leq a \leq 1$.

$$f(ax_1 + (1-a)x_2) \leq af(x_1) + (1-a)f(x_2) \tag{2.11}$$

A property of convex functions is that the local optimum is also the global optimum. It the function is strictly convex.

$$f(ax_1 + (1-a)x_2) < af(x_1) + (1-a)f(x_2) \tag{2.12}$$

Then the global optimum is also unique.

2.1.8 Quadratic Programming

Quadratic programming is a special case of nonlinear programming where the objective function is nonlinear but all the constraints are linear.

$$\begin{aligned} min\ & \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x} \\ s.t.\ & \mathbf{G}\mathbf{x} \leq \mathbf{h} \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned} \tag{2.13}$$

where $\mathbf{x}$ is a vector containing n variables that is optimized for, $\mathbf{P}$ is a $n \times n$ weight matrix and $\mathbf{q}$ is a n long weight vector. The inequality constraints are defined by the $m \times n$ matrix $\mathbf{G}$ and the n long vector $\mathbf{h}$. The equality constraints are defined by the $m \times n$ matrix $\mathbf{A}$ and the n long vector $\mathbf{b}$. As all the constraints are linear, as long as they do not conflict with each other, the problem is convex.

2.2 Stochastic optimization (Sample based)

Stochastic optimization is an optimization method well suited for nonlinear and non-convex problems where many local minima may exist. The idea is to efficiently sample the space in a probabilistic manner, to quickly find a solution while avoiding getting stuck in a local minimum. There are many types of stochastic optimization algorithms, a subset of those are biologically inspired optimization algorithms. Within biologically inspired optimization methods, some of the common methods are evolutionary algorithms, ant colony optimization and particle swarm optimization [33]. This thesis uses genetic algorithms, which is a subset of evolutionary algorithms. The basic idea is to find the optimum for a function by mimicking evolution.

2.2.1 Genetic algorithms

A version of the genetic algorithm can be seen in algorithm 1. The optimization works by setting up an initial population of individuals, each individual represents a solution to the problem. The parameters that are optimized for in an individual are referred to as genes. The initial population is usually randomly scattered in the optimization space. Then each individual is evaluated and given a fitness score, the fitness score represents how "good" an individual is. New individuals are generated from the previous generation. Each new individual in the population is generated by the following. If cross over happens, then two individuals are sampled based on their score. Cross over is done by picking a random cross over point and combining the two "parent" individuals into a new individual, see figure 2.1. The new individual then get parts of the genes from one parent and the other parts from the other parent.

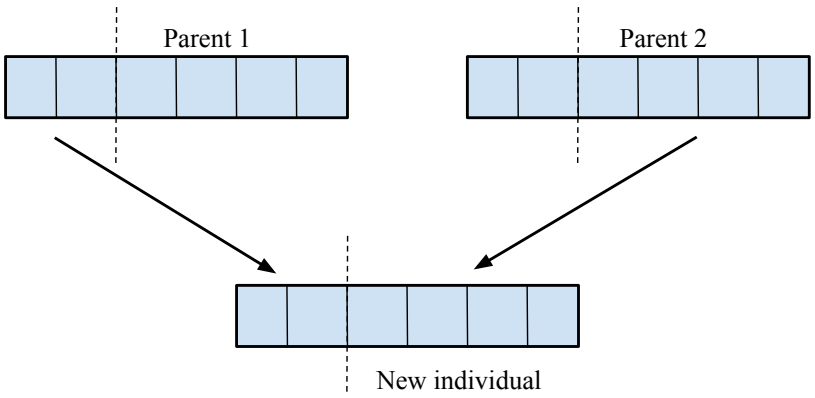


Figure 2.1: Cross over

If cross over does not happen, then an individual is sampled from the scores without cross over. Each gene in a new individual has a probability of being mutated. The new individuals replace the previous and a new generation has been created. The

best individual from the previous generation is usually passed on unchanged.

Algorithm 1: Evolution based stochastic optimization

```
Result: Solution for intermediate step
Generate initial population;
for Number of generations do
    Evaluate population;
    for Number of individuals do
        if Probability crossing then
            Sample parent one based on fitness score;
            Sample parent two based on fitness score;
            individual = cross(parent one, parent two);
        else
            Sample individual based on fitness score;
        end
        for number of genes in individual do
            if Probability gene mutation then
                gene = mutate(gene);
            end
        end
        Append individual to temporary population;
    end
    Copy the best individual to temporary population;
    Replace population with temporary population;
end
return best individual;
```

2.3 Computer vision

Computer vision is a subfield in computer science for processing and analysing images or videos to gain a higher-level perception of the environment.

2.3.1 Image encoding

A color image can be encoded in different ways [34]. A common encoding from cameras is RGB, the color is achieved by adding red, green and blue together. Another color space that will be used in this thesis is HSV. HSV stands for hue, saturation and value. The color space is then cylindrical, where the hue is periodical.

2.3.2 Camera calibration matrix

Homogeneous coordinates for a two dimensional vector $\begin{bmatrix} x & y \end{bmatrix}$ would be expressed as $\begin{bmatrix} kx & ky & k \end{bmatrix}$[35], where k is a arbitrarily scalar. For any value of k the homogeneous coordinates represents the same point.
The camera matrix $\mathbf{P} = \mathbf{K}\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$ [36], describes the intrinsic $\mathbf{K}$ and the extrinsic

$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$ parameters. The extrinsic parameters describes the pose of the camera and the intrinsic parameters also known as the calibration matrix describes the internal model of the camera. The calibration matrix can be seen in equation (2.14).

$$\mathbf{K} = \begin{bmatrix} f_{\mathrm{x}} & s & c_{\mathrm{x}} \\ 0 & f_{\mathrm{y}} & c_{\mathrm{y}} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

Where f_{x} and f_{y} represents focal lengths. s represents any skew. c_{x} and c_{y} repents the optical center. These values are in pixel coordinates.

2.3.3 Apriltags/markers

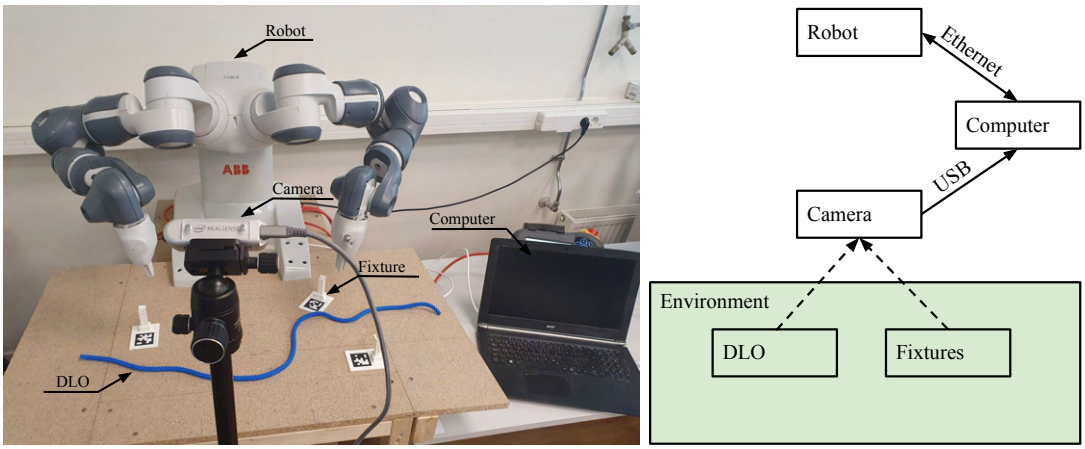
It is possible to identify the pose of a marker from a single camera if the size of the marker is known and the camera is calibrated. Apriltags used artificial features (fiducials) in the form of a 2D bar code (tag) for pose estimation [37]. The tag can be localized in 6 DOF from a single image. The algorithm works by detecting value gradients in the image from the tag and with an least squares method fits lines to the features in the tag. The transformation is then calculated by a direct linear transform procedure. The tags can also store a few bits of information, usually the ID of the tag.

3
System Overview

In this chapter, an overview of the system is presented. This thesis uses a dual-arm robot for cable routing with camera tracking. The system can be split into two parts, the hardware and the software.

3.1 Hardware system overview

An image over the hardware can be seen in figure 3.1a. The hardware consists of a robot, camera and computer. The robot interacts with the environment which consists of several fixtures and the DLO. The camera is used for estimating the DLO and fixtures and is connected to the computer with USB, see figure 3.1b. The robot is controlled by the computer and connected with Ethernet. The specifications of the computer can be seen in table 3.1.



(a) Image of the hardware **(b)** Overview of the hardware

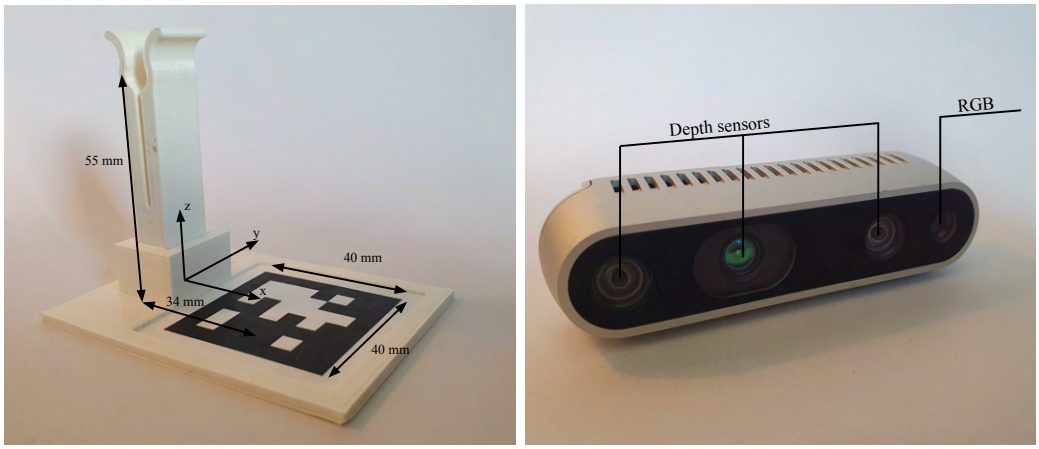
Figure 3.1: A figure of the hardware and how it is connected

Table 3.1: PC specification

CPU	Intel i5-6300HQ
GPU	Nvidia GTX 960M
RAM	8 GB
Operating system	Ubuntu 18

3.1.1 Camera and environment

For visual tracking, an Intel realsense d435 camera is used. The camera can be seen in figure 3.2b, it has both an RGB sensor and depth sensors. The depth sensor uses stereo vision together with an IR emitter. The DLO used is a rope with a diameter of 8 mm, see figure 3.2c. The rope is 1 meter in length. The fixtures used to clip the DLO are 3D printed in polylactic acid (PLA) and can be seen in figure 3.2a. Each fixture has a fiducial marker with a unique ID for pose estimation.



(a) Fixture **(b)** Camera Realsense D435i

(c) The DLO used

Figure 3.2: A figure of the fixture, camera and DLO

3.1.2 Robot

The robot used in this thesis is an ABB IRB14000, also called YuMi. A rendition of the YuMi can be seen in figure 3.3. YuMi has two identical arms with 7 DOF each. Each arm has a max load capacity of 500 grams and a reach of 0.559 meters without any attachments. Both arms have been attached with smart grippers. Each gripper has two fingers that can open and close. The grippers have a maximal grip force of 20 N. The robot is controlled externally with a computer using ABBs externally guided motion (EGM) through a ROS interface. Throughout the report some important frames will be referenced, these frames can be seen in figure 3.3. The *base frame* refers to the coordinate system at the base of the robot. This frame will be used as the base frame for the entire system and if nothing else is explicitly written then this frame will be the basis of all transforms. The *world frame* refers to a known position in the workspace and is mainly used for the camera pose estimation. The

camera has multiple internal frames, the *camera_link* frame refers to the body of the camera. For control, the end-effectors pose is defined by the *gripper_r* and *gripper_l frames*. These frames are located at the tip of the smart grippers in between the two fingers. Two frames for the elbows are also defined and used for simple self-collision avoidance.

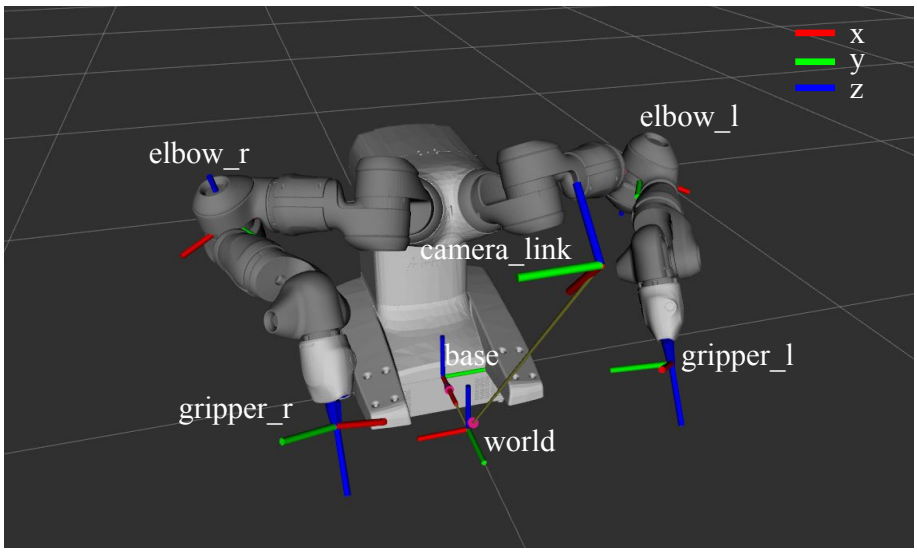


Figure 3.3: Rendition of YuMi with some key frames

Each arm of YuMi is identical and have 7 joints each. The joint numbering following ABBs convention can be seen in figure 3.4. The position and velocity limits of each joint can be seen in table 3.2. This thesis does not consider high-velocity control and the joint velocities will be far below the limits.

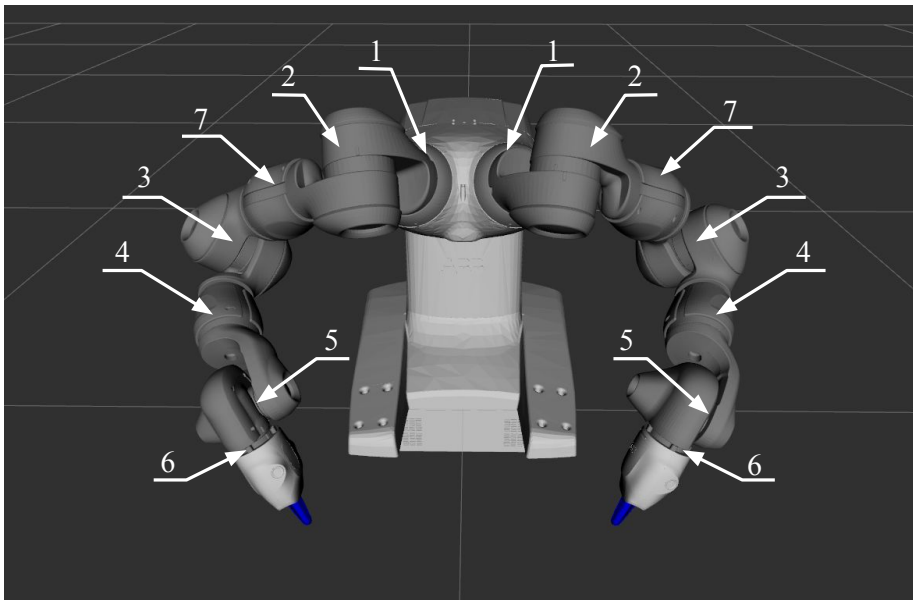


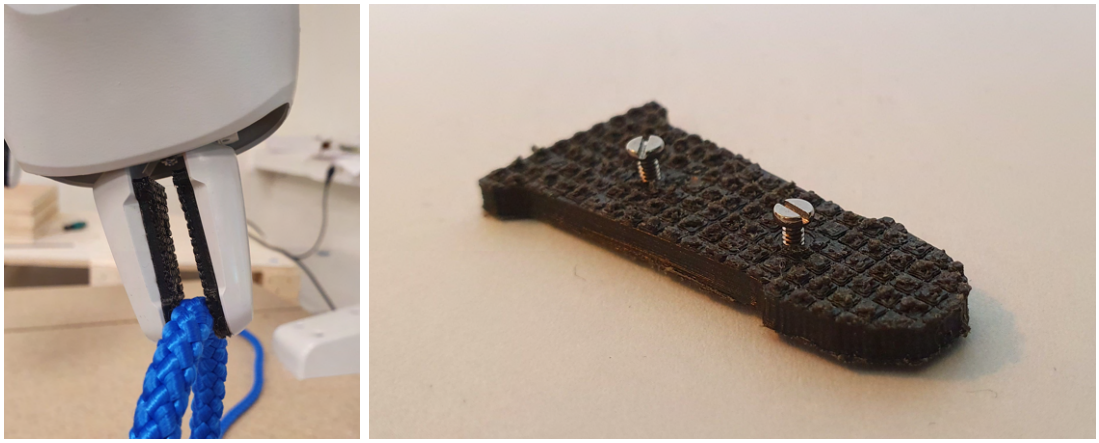
Figure 3.4: Rendition of YuMi with joint numbering

Table 3.2: Joint limits

Joint	Joint Position Range $[deg]$	Velocity Limit $[deg/s]$
1	-165.5 to 165.5	180
2	-143.5 to 43.5	180
7	-168.5 to 168.5	180
3	-123.5 to 60	180
4	-290 to 290	400
5	-88 to 138	400
6	-229 to 229	400

3.1.3 Gripper modification

Each smart gripper has a maximal grip force of 20 N. The inside of the gripper fingers has a smooth metal surface, resulting in a holding force too weak to be able to clip the DLO into the fixtures. To increase the friction, some grip pads with a rough surface were designed and printed in thermoplastic polyurethane (TPU), see figure 3.5b. Each finger in the gripper has two threaded holes and the grip pads are attached with screws to the inside of the fingers, see figure 3.5a. With the grip pads, the holding force of the DLO exceeded the capabilities of the robot (max lift capability of 500g) and the DLO can be clipped into the fixtures.



(a) Gripping the DLO **(b)** Grip pads

Figure 3.5: Smart gripper and grip pads

3.2 Software overview

A simplified overview of the software can be seen in 3.6. The software can be divided into three different parts, the controller, computer vision and path planner. The computer vision part is responsible for estimating the poses of the fixtures and track the DLO. The camera driver outputs two images, one for RGB color and one depth map. Using both the color and depth map images the DLO can be estimated in 3D. The fixtures are estimated from fiducial markers. The path planner uses the

DLO estimation, fixture poses and controller state to generate trajectory parameters. The trajectory parameters contain a list of poses that the controller should follow. The controller generates control commands for the robot based on the trajectory parameters. The software system is implemented in ROS.

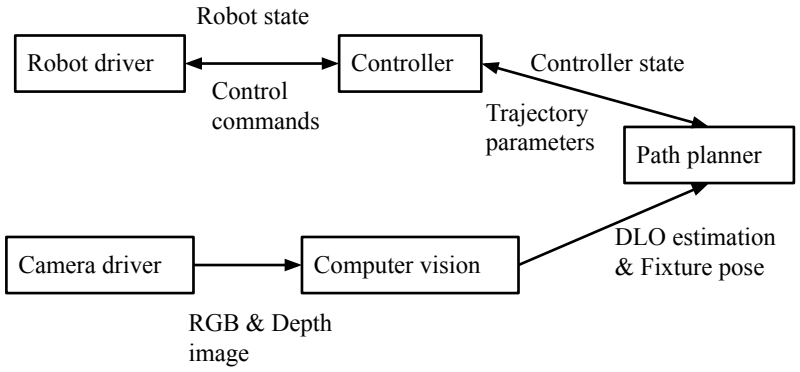


Figure 3.6: Simplified overview of the software system

3.2.1 ROS Nodes

ROS is a framework making it easier to write software for robotics [38] and provides a communication layer between nodes. The project uses ROS melodic [39] and ubuntu 18. ROS allows for simple integration with hardware, such as cameras and the robot but also provides useful tools. The tf library [40] allows for tracking multiple coordinate systems in a decentralized system. The tf library builds trees where each frame can have one parent frame but multiple child frames. Within a connected tree, any transformation between any frame can be obtained. The implementation is divided into several nodes. An overview of the relevant ROS nodes can be seen in figure 3.7, the arrows illustrate the flow of information. Note some of the background nodes have been omitted or simplified for clarity. The specific implementation will only be discussed briefly.

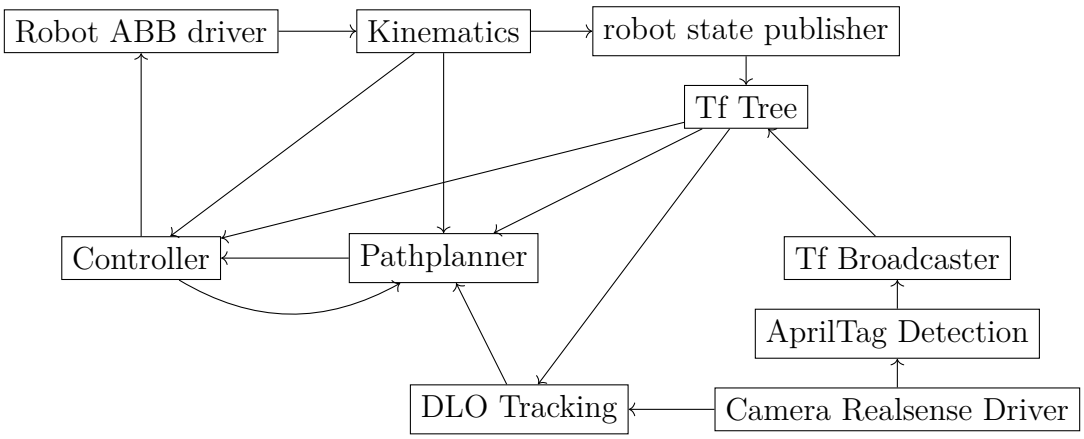


Figure 3.7: Overview over the relevant ROS nodes.

The ABB driver interfaces with the physical hardware, for control purposes the

driver sends velocity commands and receives a position for each joint. The kinematics node uses the joint positions and calculates the forward kinematics and Jacobians. To note is that there are two parallel ways of receiving the forward kinematics. The controller mainly uses the forward kinematics from the kinematics node as all the information is from the same set of joint values. The robot state publisher, a ROS library, is used for visualization and adding the robot to the tf tree. The tf tree [1], though technically not a node but instead a decentralized system, track all frames in the system that are broadcasted. The tf system is used for both visualization (in rviz) and calculating transforms between different frames. The Tf broadcaster connects the camera, fixture and gripper frames to the tree. The realsense driver interfaces with the camera hardware and publishes color and depth images to ROS. The apriltag detection node uses the color image and estimates both the camera pose and the fixtures. The Tf broadcaster will keep publishing the last known pose of the camera and fixtures, as they are assumed to be stationary the apriltag system only needs to be initialized. The DLO tracking node estimates the DLO from color and depth information and publishes a point cloud representing the DLO. The path planner uses the available information and generates trajectory parameters, the trajectory parameters describe a series of motions for executing a task. The controller calculates the joint velocity commands from the trajectory parameters and joint positions. Note that the kinematics node together with the controller node will be referred to as the controller.

[1] For more information visit, `http://wiki.ros.org/tf`

4
Controller

In this chapter, the algorithms for the controller are presented. The controller is responsible for generating joint velocity commands for the robot from trajectory parameters. An overview of the components in the controller can be seen in figure 4.1. Note that the graph only covers control over the joints, control for the grippers is discussed in section 4.7.

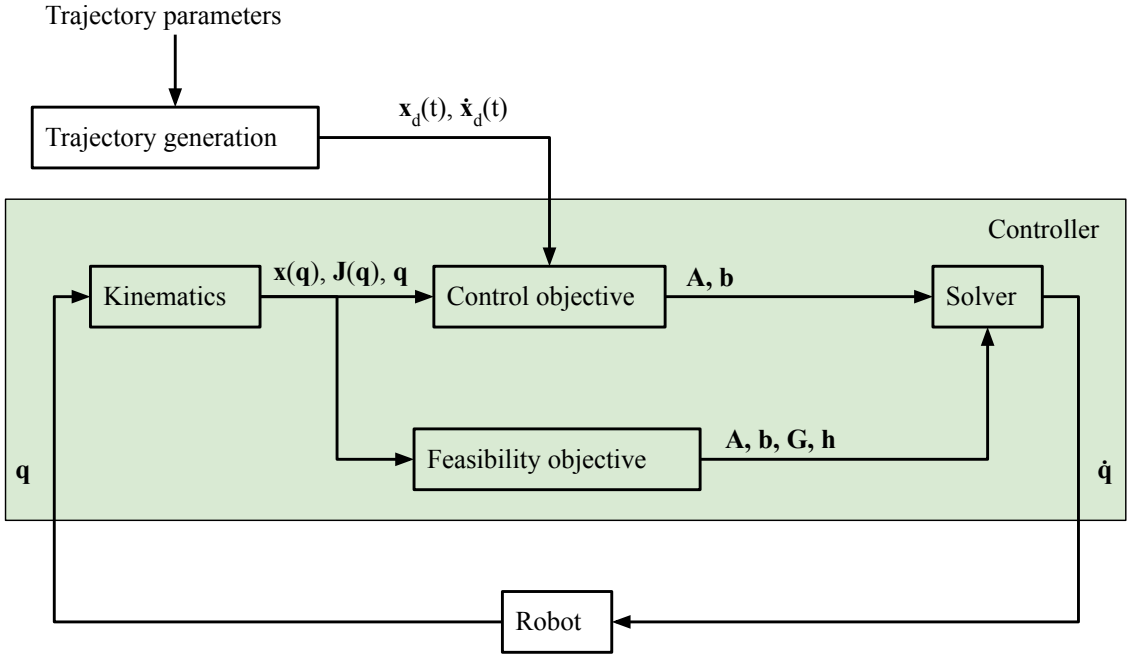


Figure 4.1: Block diagram for controller. $\mathbf{q}$ is the joint position, $\dot{\mathbf{q}}$ the joint velocities, $\mathbf{J}(\mathbf{q})$ the Jacobians, $\mathbf{x}(\mathbf{q})$ the forward kinematics, $\mathbf{x}_d(t)$ and $\dot{\mathbf{x}}_d(t)$ the desired position and velocity from the trajectory and $\mathbf{A}, \mathbf{b}, \mathbf{G}, \mathbf{h}$ are the constraints for the solver.

There are two modes the robot can be controlled. The arms can be seen as separate entities and controlled individually, this mode will be referred to as `individual manipulation`. The arms of the robot can also be seen as a single system and be controlled together, this mode is referred to as `coordinated manipulation`. The controller receives the joint positions for the robot. From the joint positions, the forward kinematics and Jacobians can be calculated. The trajectory parameters are received from the path planner, from the trajectory parameters a smooth trajectory can be calculated. The control objective generates the constraints responsible for

following the trajectory. The feasibility objectives generates the constraints that are responsible for keeping the solution within limits or avoiding collisions but are not directly controlled. The joint velocity commands are solved with an HQP solver.

4.1 Kinematics

The calculation of forward kinematics and Jacobians will be very brief. They are calculated with the Orocos kdl library [1] and a YuMi model from Orebro university [2]. Through the forward kinematics can be calculated with the tf library, with kdl the forward kinematics and Jacboians can be ensured to be generated from the same joint position measurement. From the kdl library and YuMi model the forward kinematics and Jacobians used will be for joint 6 and joint 3 expressed in the *base frame*, see figures 3.3 and 3.4.

4.2 Trajectory

From the trajectory parameters, a smooth trajectory can be calculated. The trajectory parameters define the poses (position and orientation) that the trajectory should pass through. Each trajectory parameter has a time T_i attached, which describes how long it should take to reach the pose.
For position trajectories, the trajectory can be divided into 3 components. Each component representing a translation along an axis (x, y, z) in the *base frame* to a gripper, each can be solved separately. For a given axis the trajectory is a function of time, that returns a scalar value for the position. The trajectory between two points (p_i and $p_{\mathrm{i}+1}$) can be generated by fitting a cubic function, see equation (2.8). If the initial and final velocity ($\dot{p}_\mathrm{i}$ and $\dot{p}_{\mathrm{i}+1}$) is known then that parameters can be solved by the equations (4.1). Then using the time $0 \leq t_\mathrm{i} \leq T_\mathrm{i}$, the desired position and velocity can be expressed as a function of time, $p_\mathrm{i}^\mathrm{d}(t_\mathrm{i})$ and $\dot{p}_\mathrm{i}^\mathrm{d}(t_\mathrm{i})$.

$$
\begin{aligned}
a_0 &= p_\mathrm{i} \\
a_1 &= \dot{p}_\mathrm{i} \\
a_2 &= \frac{3p_{\mathrm{i}+1} - \dot{p}_{\mathrm{i}+1}T_{\mathrm{i}+1} - 2a_1T_{\mathrm{i}+1} - a_0}{T_{\mathrm{i}+1}^2} \\
a_3 &= \frac{\dot{p}_{\mathrm{i}+1} - 2a_2T_{\mathrm{i}+1} - a_{\mathrm{i}+1}}{3T_{\mathrm{i}+1}^2}
\end{aligned}
\tag{4.1}
$$

What has been described so far generates a trajectory between two points. The trajectory parameters can contain *i = 1, 2, ... n* poses, the trajectory segments can then be linked together. For a smooth trajectory, the initial velocity must match the final velocity of the previous segment. The algorithm for calculating the velocities can be seen in algorithm 2. For the initial velocity for the first segment, the velocity is set to the current velocity of the robot. The final velocity of the final segment

[1] For more information visit, `https://www.orocos.org/kdl.html`
[2] For more information visit, `https://github.com/OrebroUniversity/yumi`

is always set to 0. The intermediate velocities between the segments depend on if there is a velocity change or not.

Algorithm 2: Calculates velocity $\dot{p}_i$ from trajectory parameters

Result: $\dot{p}_i$
i ranges from 1 to n;
if $i \geq 2$ *and* $i \leq n-1$ **then**
  $AvgV_i = \frac{p_i - p_{i-1}}{T_i}$;
  $AvgV_{i+1} = \frac{p_{i+1} - p_i}{T_{i+1}}$;
  if $sign(AvgV_i) == sign(AvgV_{i+1})$ **then**
    $\dot{p}_i = \frac{AvgV_i + AvgV_{i+1}}{2}$
  else
    $\dot{p}_i = 0$
  end
else
  if $i == 0$ **then**
    $\dot{p}_i$ = current velocity of robot
  else
    $\dot{p}_i = 0$
  end
end
return $\dot{p}_i$;

By calculating the trajectory for each axis, a vector:

$$\mathbf{x}_{\mathrm{d}}(t) = \begin{bmatrix} x_{\mathrm{d}}(t) & y_{\mathrm{d}}(t) & z_{\mathrm{d}}(t) \end{bmatrix}^{\mathrm{T}} \tag{4.2}$$

that contains the desired position over time can be obtained. From equation (2.9) we have the desired velocities:

$$\dot{\mathbf{x}}_{\mathrm{d}}(t) = \begin{bmatrix} \dot{x}_{\mathrm{d}}(t) & \dot{y}_{\mathrm{d}}(t) & \dot{z}_{\mathrm{d}}(t) \end{bmatrix}^{\mathrm{T}} \tag{4.3}$$

The orientation is represented as quaternions and generating trajectories for orientation is not equally trivial as for position. The trajectories for orientation are simplified, the angular velocity reaches zero between each trajectory parameter. While it is possible to create a continuous motion for all trajectory parameters, it is considered outside the scope of this thesis. The following method is described in [31]. The quaternions are converted to rotational matrices $\mathbf{R}$. The rotation matrix that describes the relative rotation between the initial $\mathbf{R}_{\mathrm{i}}$ and final $\mathbf{R}_{\mathrm{i}}^{\mathrm{i+1}}$ orientation.

$$\mathbf{R}_{\mathrm{i+1}}^{\mathrm{i}} = \mathbf{R}_{\mathrm{i}}^{\mathrm{T}} \mathbf{R}_{\mathrm{i+1}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.4}$$

The rotation $\mathbf{R}_{\mathrm{i+1}}^{\mathrm{i}}$ can be expressed as a rotation around a single axis with a new ration matrix $\mathbf{R}^{\mathrm{i}}(t_{\mathrm{i}})$. Where $\mathbf{R}^{\mathrm{i}}(0) = \mathbf{I}$ and $\mathbf{R}^{\mathrm{i}}(T_{\mathrm{i}}) = \mathbf{R}_{\mathrm{i+1}}^{\mathrm{i}}$. The unit vector describing the axis $\mathbf{r}$ of rotation and the rotation angle around the axis φ_{f} can be calculated by:

$$\varphi_{\mathrm{f}} = \arccos \frac{r_{11} + r_{22} + r_{33} - 1}{2}$$
$$\mathbf{r} = \frac{1}{2 \sin \varphi_{\mathrm{f}}} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \tag{4.5}$$

The idea is to smoothly vary the angle $\varphi(t)$ with respect to time around the axis $\mathbf{r}$. The function $\varphi(t)$ can be generated in the same manner as for position, by fitting a cubic function (2.8). With initial angle $\varphi(0) = 0$, the final angle $\varphi(T_{\mathrm{i}}) = \varphi_{\mathrm{f}}$ and both of the derivatives $\dot{\varphi}(0) = \dot{\varphi}(T_{\mathrm{f}}) = 0$. The velocity components $\boldsymbol{\omega}^{\mathrm{i}}$ in $\mathbf{R}_{\mathrm{i}}$ can be calculated as:

$$\boldsymbol{\omega}^{\mathrm{i}}(T) = \dot{\varphi}(T) r \tag{4.6}$$

The angular velocity is then rotated back to the robots *base frame* by:

$$\boldsymbol{\omega}_{\mathrm{d}}(T) = \mathbf{R}_{\mathrm{i}-1} \boldsymbol{\omega}^{\mathrm{i}}(T) \tag{4.7}$$

Resulting in the desired angular velocities. For the desired orientation, the rotation matrix $\mathbf{R}^{\mathrm{i}}(T)$ can be calculated from $\mathbf{r}$ and $\varphi(T)$ by:

$$\begin{aligned} c_\varphi &= \cos\left(\varphi(T)\right) \\ s_\varphi &= \sin\left(\varphi(T)\right) \\ \mathbf{R}^{\mathrm{i}}(T) &= \begin{bmatrix} \mathbf{r}_{\mathrm{x}}^2(1 - c_\varphi) + c_\varphi & \mathbf{r}_{\mathrm{x}}\mathbf{r}_{\mathrm{y}}(1 - c_\varphi) + \mathbf{r}_{\mathrm{z}} s_\varphi & \mathbf{r}_{\mathrm{x}}\mathbf{r}_{\mathrm{z}}(1 - c_\varphi) + \mathbf{r}_{\mathrm{y}} s_\varphi \\ \mathbf{r}_{\mathrm{x}}\mathbf{r}_{\mathrm{y}}(1 - c_\varphi) + \mathbf{r}_{\mathrm{z}} s_\varphi & \mathbf{r}_{\mathrm{y}}^2(1 - c_\varphi) + c_\varphi & \mathbf{r}_{\mathrm{y}}\mathbf{r}_{\mathrm{z}}(1 - c_\varphi) + \mathbf{r}_{\mathrm{x}} s_\varphi \\ \mathbf{r}_{\mathrm{x}}\mathbf{r}_{\mathrm{z}}(1 - c_\varphi) + \mathbf{r}_{\mathrm{y}} s_\varphi & \mathbf{r}_{\mathrm{y}}\mathbf{r}_{\mathrm{z}}(1 - c_\varphi) + \mathbf{r}_{\mathrm{x}} s_\varphi & \mathbf{r}_{\mathrm{z}}^2(1 - c_\varphi) + c_\varphi \end{bmatrix} \end{aligned} \tag{4.8}$$

Then transformed back to the *base frame*, resulting in the desired orientation is then given by:

$$\mathbf{R}_{\mathrm{d}} = \mathbf{R}_{\mathrm{i}-1} \mathbf{R}^{\mathrm{i}}(T) \tag{4.9}$$

The desired orientation can be converted back to a quaternion representation $\mathbf{Q}_{\mathrm{d}}$.

4.3 Hierarchical Quadratic Programming

Hierarchical Quadratic Programming (HQP) solves the quadratic programming with a hierarchy on tasks [7]. The standard quadratic programming formulation is written in the form (2.13). The idea with HQP is to add relaxation variables that can relax a constraint if no possible solution exists. The implementation of the algorithm is

based on previous students work. The overall algorithm can be seen in algorithm 3.

Algorithm 3: Solves quadratic programming with hierarchy on constraints

Result: x
for *number of tasks* **do**
    Extend task with relaxation variables;
    for *number of previous tasks* **do**
        Apply previously calculated relaxation variable to previous task;
        Append previous task;
    end
    Extend cost matrix;
    Solve with normal quadratic solver;
    Save relaxation variable;
end
x equals the last solution, without the relaxation variables;
return x;

The HQP solver is used to calculate the joint velocities $\dot{\mathbf{q}}$ for the robot, where $\dot{\mathbf{q}} = \begin{bmatrix} \dot{\mathbf{q}}_\mathrm{r} & \dot{\mathbf{q}}_\mathrm{l} \end{bmatrix}^\mathrm{T}$ i.e. a vector with dimension 14 as each arm has 7 joints. In the HQP algorithm a task is a set of constraints, for example this can be the control objective or collision avoidance. The tasks are ordered with descending hierarchy. HQP solves a QP problem for each task, solving both the parameters and the relaxation variables $\mathbf{x} = \begin{bmatrix} \dot{\mathbf{q}} & \mathbf{V}^\mathrm{i}_\mathrm{relax} \end{bmatrix}^\mathrm{T}$. The relaxation variables $\mathbf{V}^\mathrm{i}_\mathrm{relax}$ are added for each task. The tasks are solved in order and all previous tasks are appended. The relaxation variables are calculated by extending the current task.

$$\mathbf{A}_\mathrm{relax} = \begin{bmatrix} \mathbf{A} & -\mathbf{I} \end{bmatrix} \tag{4.10}$$

$$\mathbf{G}_\mathrm{relax} = \begin{bmatrix} \mathbf{G} & -\mathbf{I} \end{bmatrix} \tag{4.11}$$

Then for any previous task with higher priority their constraints are relaxed with their respective relaxation variables. For equality constraints the constraints are relaxed by:

$$\begin{aligned} \mathbf{A}_\mathrm{relaxed} &= \begin{bmatrix} \mathbf{A} & \mathbf{0} \end{bmatrix} \\ \mathbf{b}_\mathrm{relaxed} &= \mathbf{b} + \mathbf{V}_\mathrm{relax} \end{aligned} \tag{4.12}$$

For inequality constraints:

$$\begin{aligned} \mathbf{G}_\mathrm{relaxed} &= \begin{bmatrix} \mathbf{G} & \mathbf{0} \end{bmatrix} \\ \mathbf{h}_\mathrm{relaxed} &= \mathbf{h} + max(\begin{bmatrix} \mathbf{V}_\mathrm{relax} & \mathbf{0} \end{bmatrix}) \end{aligned} \tag{4.13}$$

The next step is to stack the constraints from the current task and all the previous tasks. Resulting in the constraint matrices for task i is then:

$$\mathbf{A}^\mathrm{i}\mathbf{x} = \begin{bmatrix} \mathbf{A}^\mathrm{i}_\mathrm{relax} \\ \mathbf{A}^{\mathrm{i}-1}_\mathrm{relaxed} \\ . \\ \mathbf{A}^0_\mathrm{relaxed} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{V}^\mathrm{i}_\mathrm{relax} \end{bmatrix} = \mathbf{b} = \begin{bmatrix} \mathbf{b}^\mathrm{i} \\ \mathbf{b}^{\mathrm{i}-1}_\mathrm{relaxed} \\ . \\ \mathbf{b}^0_\mathrm{relaxed} \end{bmatrix} \tag{4.14}$$

Similarly for inequality constraints:

$$\mathbf{G}^{\mathrm{i}}\mathbf{x} = \begin{bmatrix} \mathbf{G}^{\mathrm{i}}_{\mathrm{relax}} \\ \mathbf{G}^{\mathrm{i-1}}_{\mathrm{relaxed}} \\ . \\ \mathbf{G}^{0}_{\mathrm{relaxed}} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{V}^{\mathrm{i}}_{\mathrm{relax}} \end{bmatrix} \leq \mathbf{b} = \begin{bmatrix} \mathbf{h}^{\mathrm{i}} \\ \mathbf{h}^{\mathrm{i-1}}_{\mathrm{relaxed}} \\ . \\ \mathbf{h}^{0}_{\mathrm{relaxed}} \end{bmatrix} \tag{4.15}$$

From equation (2.13) the matrices $\mathbf{P}$ and $\mathbf{q}^{\mathrm{T}}$ need to be chosen. $\mathbf{q}^{\mathrm{T}}$ is chosen to be a vector containing zeros with the same dimensions as $\mathbf{x}$. $\mathbf{P}$ is a square matrix with the dimensions $\mathbf{x} \times \mathbf{x}$ and is chosen to be:

$$\mathbf{P} = \begin{bmatrix} \mathbf{I}_{\dot{\mathrm{q}}} & \mathbf{0} \\ \mathbf{0} & \alpha_i \mathbf{I}_{\mathrm{V}} \end{bmatrix} \tag{4.16}$$

Where α_i is a constant and can be chosen to be unique for each task, see table 4.1. α is chosen to be a large number, as the objective is to use the relaxation variables as little as possible. When the last task has been solved, the $\dot{\mathbf{q}}$ for that solution is extracted and used as joint velocity commands for the robot. The hierarchy of the tasks is presented in table 4.1. How the constraints are calculated is described in subsequent chapters.

Table 4.1: HQP hierarchy

Hierarchy	Task	Type	α
1	Joint Velocity limit	Inequality	10^3
2	Joint Position limit	Inequality	10^3
3	Elbow Proximity Limit	Inequality	10^3
4	Control Objective	Equality	10^4
5	Joint Position Potential	Equality	2×10^5

4.4 Control Objective

In this section it is discussed how the constraints for trajectory following is calculated. The Jacobian $\mathbf{J}_{\mathrm{w}}$ from the kdl library maps the joint velocities to the wrist (joint 6), see equation (4.17). $\dot{\mathbf{x}}_{\mathrm{w}}$ is a vector defining $\begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^{\mathrm{T}}$ velocities for the wrist and $\boldsymbol{\omega}_{\mathrm{w}} = \begin{bmatrix} \omega_{\mathrm{x}} & \omega_{\mathrm{y}} & \omega_{\mathrm{z}} \end{bmatrix}^{\mathrm{T}}$ defines angular velocity in the *base frame*.

$$\begin{bmatrix} \dot{\mathbf{x}}_{\mathrm{w}} \\ \boldsymbol{\omega}_{\mathrm{w}} \end{bmatrix} = \mathbf{J}_{\mathrm{w}}\dot{\mathbf{q}} \tag{4.17}$$

Though it is more convenient to control a point that is at the tip of the grippers i.e. the frames *gripper_r* and *gripper_l*, see figure 3.3. This can be done by mapping $\dot{\mathbf{x}}_{\mathrm{w}}$ and $\boldsymbol{\omega}_{\mathrm{w}}$ to $\dot{\mathbf{x}}_{\mathrm{g}}$ and $\boldsymbol{\omega}_{\mathrm{g}}$ where the subscript g stands for gripper.

$$\begin{bmatrix} \dot{\mathbf{x}}_{\mathrm{g}} \\ \boldsymbol{\omega}_{\mathrm{g}} \end{bmatrix} = \mathbf{L}^{\mathrm{w}}_{\mathrm{g}}\mathbf{J}_{\mathrm{w}}\dot{\mathbf{q}} = \mathbf{J}_{\mathrm{g}}\dot{\mathbf{q}} \tag{4.18}$$

where $\mathbf{L}_{\text{wg}}$ is a linking matrix that maps.

$$\begin{bmatrix} \dot{\mathbf{x}}_{\text{g}} \\ \boldsymbol{\omega}_{\text{g}} \end{bmatrix} = \mathbf{L}_{\text{wg}} \begin{bmatrix} \dot{\mathbf{x}}_{\text{w}} \\ \boldsymbol{\omega}_{\text{w}} \end{bmatrix} \tag{4.19}$$

The angular velocity is unchanged, through the positional velocity is affected by the angular velocity as the grippers can be seen as an extension. The components of the linking matrix then become:

$$\mathbf{L}_{\text{wg}} = \begin{bmatrix} \mathbf{I} & \mathbf{L}_{\text{r}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{4.20}$$

where $\mathbf{L}_{\text{r}}$ is a skew symmetric matrix describing the contribution of angular velocity to positional velocity:

$$\mathbf{L}_{\text{r}} = \begin{bmatrix} 0 & v_{\text{z}} & -v_{\text{y}} \\ -v_{\text{z}} & 0 & v_{\text{x}} \\ v_{\text{y}} & -v_{\text{x}} & 0 \end{bmatrix} \tag{4.21}$$

where:

$$\begin{bmatrix} v_{\text{x}} \\ v_{\text{y}} \\ v_{\text{z}} \end{bmatrix} = \mathbf{R}_{\text{w}} \mathbf{P}_{\text{g}}^{\text{w}} \tag{4.22}$$

$\mathbf{R}_{\text{w}}$ is the rotation matrix from *base frame* to the wrist and $\mathbf{P}_{\text{g}}^{\text{w}}$ is the position transform from the wrist to the gripper point. Which gives:

$$\mathbf{J}_{\text{g}} = \mathbf{L}_{\text{wg}} \mathbf{J}_{\text{w}} \tag{4.23}$$

The Jacobian now describes the differential kinematics between the base and the gripper point.

4.4.1 Individual Manipulation

For `individual manipulation`, the arms are treated as separate entities. Each arm has 7 DOF and the end-effectors is controlled with 6 DOF, the system is redundant as the arms have high DOF than whats controlled. This means that there are multiple solutions for the inverse kinematics and the inverse differential kinematics. This problem can be solved as an optimization problem. In section 4.3 the solver is discussed in more detail. The optimization is solved with quadratic programming, see equation (2.13), where the motion is expressed as equality constraints on the form:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.24}$$

where $\mathbf{A}$ represents the differential kinematics, $\mathbf{b}$ the target velocities for the end effector and $\mathbf{x}$ solves for the joint velocities. The Jacobians can be combined, which results in:

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{J}_{\text{r}} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_{\text{l}} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_{\text{r}} \\ \dot{\mathbf{q}}_{\text{l}} \end{bmatrix} = \mathbf{b} = \begin{bmatrix} \dot{\mathbf{x}}_{\text{r}} \\ \boldsymbol{\omega}_{\text{r}} \\ \dot{\mathbf{x}}_{\text{l}} \\ \boldsymbol{\omega}_{\text{l}} \end{bmatrix} \tag{4.25}$$

$\mathbf{J}_\mathrm{r}$ and $\mathbf{J}_\mathrm{l}$ is the Jacobian calculated in equation (4.23) for each arm, $\dot{\mathbf{q}}_\mathrm{r}$ and $\dot{\mathbf{q}}_\mathrm{l}$ is the joint velocities for each arm and $\begin{bmatrix} \dot{\mathbf{x}}_\mathrm{r} & \boldsymbol{\omega}_\mathrm{r} & \dot{\mathbf{x}}_\mathrm{l} & \boldsymbol{\omega}_\mathrm{l} \end{bmatrix}^\mathrm{T}$ is the target velocity for each end effector. The desired velocities ($\dot{\mathbf{x}}_\mathrm{d}$ and $\boldsymbol{\omega}_\mathrm{d}$) and pose ($\mathbf{x}_d$ and $\mathbf{Q}_d$) are calculated from the trajectory parameters as described in section 4.2. For one arm the target velocity is calculated as:

$$\begin{bmatrix} \dot{\mathbf{x}}_\mathrm{t} \\ \boldsymbol{\omega}_\mathrm{t} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{x}}_\mathrm{d} \\ \boldsymbol{\omega}_\mathrm{d} \end{bmatrix} + \begin{bmatrix} k_\mathrm{p}\mathbf{e}_\mathrm{p} \\ k_\mathrm{o}\mathbf{e}_\mathrm{o} \end{bmatrix} \tag{4.26}$$

The target velocities equal the desired velocities added a correction term. The correction term consists of the position gain k_p multiplied with the position error $\mathbf{e}_\mathrm{p}$ and orientation gain k_o multiplied orientation error $\mathbf{e}_\mathrm{o}$. The position error $\mathbf{e}_\mathrm{p}$ is calculated by taking the difference between the desired and current position $\mathbf{x}_\mathrm{c}$.

$$\mathbf{e}_\mathrm{p} = \mathbf{x}_\mathrm{t} - \mathbf{x}_\mathrm{c} \tag{4.27}$$

The orientation error $\mathbf{e}_\mathrm{o}$ can be calculated [31] by equation (4.28), it represents angular error expressed in the *base frame*. The desired quaternion $Q_\mathrm{d} = \begin{bmatrix} \boldsymbol{\epsilon}_\mathrm{d} & \eta_\mathrm{d} \end{bmatrix}^\mathrm{T}$ and the current orientation of the end effector $Q_\mathrm{e} = \begin{bmatrix} \boldsymbol{\epsilon}_\mathrm{e} & \eta_\mathrm{e} \end{bmatrix}^\mathrm{T}$.

$$\mathbf{e}_0 = \eta_\mathrm{e}\boldsymbol{\epsilon}_\mathrm{t} - \eta_\mathrm{t}\boldsymbol{\epsilon}_\mathrm{e} - S(\boldsymbol{\epsilon}_\mathrm{d})\boldsymbol{\epsilon}_\mathrm{e} \tag{4.28}$$

Where $S(\mathbf{v})$ is the skew-symmetric matrix.

$$S(\mathbf{v}) = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \tag{4.29}$$

The calculations are identical for both arms and with equation (4.26) the constraints defined in equation (4.25) have been calculated. The gain k_p and k_o has been arbitrary selected to 1. These will be used by the solver in section 4.3 to solve for the joint velocities. `Individual manipulation` is added as a single task containing both arms to the HQP solver.

4.4.2 Coordinated Manipulation

For `coordinated manipulation`, the arms are controlled as one system. The trajectory parameters no longer define a set of poses for each gripper but instead a set of poses for absolute and `relative motion` [6]. Instead of controlling each gripper individually, the `absolute motion` controls a new *absolute frame* that is defined as the average of the grippers. If the position for the right gripper (*gripper_r*) is denoted $\mathbf{x}_\mathrm{r}$ and the position for the left gripper (*gripper_l*) is denoted $\mathbf{x}_\mathrm{l}$ then the average position is.

$$\mathbf{x}_\mathrm{abs} = \frac{\mathbf{x}_\mathrm{r} + \mathbf{x}_\mathrm{l}}{2} \tag{4.30}$$

Each end effector orientation is represented with a quaternion, $\mathbf{Q}_\mathrm{r}$ for the right and $\mathbf{Q}_l$ for the left gripper. The average orientation $\mathbf{Q}_{abs}$ can be calculated with:

$$\mathbf{Q}_{abs} = avg(\ \mathbf{Q}_\mathrm{r} \quad \mathbf{Q}_\mathrm{l}\ ) \tag{4.31}$$

The `relative motion` controls the difference between the poses of the grippers. In this thesis, it has been chosen that the difference is defined in the *absolute frame*. This makes some types of manipulations easier, for example, when both grippers are holding the same object. From $\mathbf{Q}_{\text{abs}}$ and $\mathbf{x}_{\text{abs}}$ the transformation matrix $\mathbf{T}_{\text{abs}}$ to the *absolute frame* can be obtained. The pose of the grippers can be expressed as transformation matrices $\mathbf{T}_{\text{r}}$ and $\mathbf{T}_{\text{l}}$. They can then be expressed in the *absolute frame*:

$$\begin{aligned} \mathbf{T}_{\text{r}}^{abs} &= \mathbf{T}_{abs}^{-1}\mathbf{T}_{\text{r}} \\ \mathbf{T}_{\text{l}}^{\text{abs}} &= \mathbf{T}_{\text{abs}}^{-1}\mathbf{T}_{\text{l}} \end{aligned} \tag{4.32}$$

The translation for each gripper $\mathbf{x}_{\text{r}}^{\text{abs}}$ and $\mathbf{x}_{\text{l}}^{\text{abs}}$ in the *absolute frame* can be obtained from the transforms in equation (4.32). The relative position is the difference between the grippers in the *absolute frame*:

$$\mathbf{x}_{\text{rel}}^{\text{abs}} = \mathbf{x}_{\text{r}}^{\text{abs}} - \mathbf{x}_{\text{l}}^{\text{abs}} \tag{4.33}$$

Similarly, the orientations for the grippers $\mathbf{Q}_{\text{r}}^{\text{abs}}$ and $\mathbf{Q}_{\text{l}}^{\text{abs}}$ in the *absolute frame* can be obtained. The relative orientation is the difference between the two quaternions, which is calculated with quaternion multiplication and conjugate one of the quaternions [41].

$$\mathbf{Q}_{\text{rel}}^{\text{abs}} = \mathbf{Q}_{\text{r}}^{\text{abs}} \mathit{conj}(\mathbf{Q}_{\text{l}}^{abs}) \tag{4.34}$$

Now the absolute and relative frames have been defined. The Jacobian calculated in equation (4.23) has to be modified. As is the Jacobian in (4.23) relates joint velocities $\dot{\mathbf{q}}$ to gripper velocities $\begin{bmatrix} \dot{\mathbf{x}}_{\text{r}} & \boldsymbol{\omega}_{\text{r}} & \dot{\mathbf{x}}_{\text{l}} & \boldsymbol{\omega}_{\text{l}} \end{bmatrix}^{\text{T}}$, these needs to be mapped to $\begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}} & \boldsymbol{\omega}_{\text{abs}} & \dot{\mathbf{x}}_{\text{rel}}^{\text{abs}} & \boldsymbol{\omega}_{\text{rel}}^{\text{abs}} \end{bmatrix}^{\text{T}}$. The differential kinematics can be split into two Jacobians, one for absolute and one for `relative motion`.

$$\begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}} \\ \boldsymbol{\omega}_{\text{abs}} \end{bmatrix} = \mathbf{L}_{\text{abs}}\mathbf{J}_{\text{g}}\dot{\mathbf{q}} = \mathbf{J}_{\text{abs}}\dot{\mathbf{q}} \tag{4.35}$$

and

$$\begin{bmatrix} \dot{\mathbf{x}}_{\text{rel}}^{\text{abs}} \\ \boldsymbol{\omega}_{\text{rel}}^{\text{abs}} \end{bmatrix} = \mathbf{L}_{\text{rel}}\mathbf{J}_{\text{g}}\dot{\mathbf{q}} = \mathbf{J}_{\text{rel}}\dot{\mathbf{q}} \tag{4.36}$$

Where $\mathbf{L}_{\text{abs}}$ and $\mathbf{L}_{\text{rel}}^{\text{abs}}$ is linking matrices mapping:

$$\begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}} \\ \boldsymbol{\omega}_{\text{abs}} \end{bmatrix} = \mathbf{L}_{\text{abs}} \begin{bmatrix} \dot{\mathbf{x}}_{\text{r}} \\ \boldsymbol{\omega}_{\text{r}} \\ \dot{\mathbf{x}}_{\text{l}} \\ \boldsymbol{\omega}_{\text{l}} \end{bmatrix} \tag{4.37}$$

$$\begin{bmatrix} \dot{\mathbf{x}}_{\text{rel}}^{\text{abs}} \\ \boldsymbol{\omega}_{\text{rel}}^{\text{abs}} \end{bmatrix} = \mathbf{L}_{\text{rel}} \begin{bmatrix} \dot{\mathbf{x}}_{\text{r}} \\ \boldsymbol{\omega}_{\text{r}} \\ \dot{\mathbf{x}}_{\text{l}} \\ \boldsymbol{\omega}_{\text{l}} \end{bmatrix} \tag{4.38}$$

The absolute velocity is defined as the average velocity of the grippers, which gives the linking matrix (4.39). Where $\mathbf{I}_6$ is a 6 by 6 identity matrix.

$$\mathbf{L}_{\text{abs}} = \begin{bmatrix} 0.5\mathbf{I}_6 & 0.5\mathbf{I}_6 \end{bmatrix} \tag{4.39}$$

The relative linking matrix, which describes the difference in right and left gripper velocity, can be calculated by:

$$\mathbf{L}_{\text{rel}}^{\text{abs}} = \begin{bmatrix} \mathbf{R}_{\text{abs}}^{-1} & \mathbf{RS} & -\mathbf{R}_{\text{abs}}^{-1} & \mathbf{RS} \\ \mathbf{0}_3 & \mathbf{R}_{\text{abs}}^{-1} & \mathbf{0}_3 & -\mathbf{R}_{\text{abs}}^{-1} \end{bmatrix} \tag{4.40}$$

The lower part of equation (4.40) describes the angular velocities. The angular velocities are first transformed to the *absolute frame* with $\boldsymbol{\omega}_{\text{r}}^{\text{abs}} = \mathbf{R}_{\text{abs}}^{-1}\boldsymbol{\omega}_{\text{r}}$ and $\boldsymbol{\omega}_{\text{l}}^{\text{abs}} = \mathbf{R}_{\text{abs}}^{-1}\boldsymbol{\omega}_{\text{l}}$. Then the difference is $\boldsymbol{\omega}_{\text{r}}^{\text{abs}} - \boldsymbol{\omega}_{\text{l}}^{\text{abs}}$ which results in the lower part of equation (4.40). The upper part describes the mapping for the positional velocities, it is less trivial as it also depends on absolute angular velocities and the distance from the *absolute frame*. For the trivial case where there is no rotational velocity i.e. $\boldsymbol{\omega}_{\text{r}} = \boldsymbol{\omega}_{\text{l}} = \mathbf{0}$, the mapping becomes:

$$\dot{\mathbf{x}}_{\text{rel}_{\text{p}}}^{\text{abs}} = \dot{\mathbf{x}}_{\text{r}}^{\text{abs}} - \dot{\mathbf{x}}_{\text{l}}^{\text{abs}} = \mathbf{R}_{\text{abs}}^{\text{-1}}\dot{\mathbf{x}}_{\text{r}} - \mathbf{R}_{\text{abs}}^{-1}\dot{\mathbf{x}}_{\text{l}} \tag{4.41}$$

For the case where there is only rotational velocity and no translation velocity i.e. $\dot{\mathbf{x}}_{\text{r}} = \dot{\mathbf{x}}_{\text{l}} = \mathbf{0}$.

$$\dot{\mathbf{x}}_{\text{rel}_{\text{o}}}^{\text{abs}} = \mathbf{RS}\boldsymbol{\omega}_{\text{r}} - (-\mathbf{RS})\boldsymbol{\omega}_{\text{l}} \tag{4.42}$$

Where RS is a rotated skew symmetric matrix for the difference in position between the grippers. The difference between the grippers position in *base frame*:

$$\mathbf{x}_{\text{rel}} = \mathbf{x}_{\text{r}} - \mathbf{x}_{\text{l}} \tag{4.43}$$

A skew symmetric matrix that relates angular velocity to positional velocity:

$$S = \begin{bmatrix} 0 & -\mathbf{x}_{\text{rel}}^{\text{z}} & \mathbf{x}_{\text{rel}}^{\text{y}} \\ \mathbf{x}_{\text{rel}}^{\text{z}} & 0 & -\mathbf{x}_{\text{rel}}^{\text{x}} \\ -\mathbf{x}_{\text{rel}}^{\text{y}} & \mathbf{x}_{\text{rel}}^{\text{x}} & 0 \end{bmatrix} \tag{4.44}$$

This is rotated to the *absolute frame* and divided by half as the distance to each gripper from the *absolute frame* is half the relative distance between them.

$$\mathbf{RS} = \frac{1}{2}\mathbf{R}_{\text{abs}}^{-1}S \tag{4.45}$$

Combining the two components, equations (4.41) and (4.42).

$$\dot{\mathbf{x}}_{\text{rel}}^{\text{abs}} = \mathbf{R}_{\text{abs}}^{-1}\dot{\mathbf{x}}_{\text{r}} + \mathbf{RS}\boldsymbol{\omega}_{\text{r}} - \mathbf{R}_{\text{abs}}^{-1}\dot{\mathbf{x}}_{\text{l}} + \mathbf{RS}\boldsymbol{\omega}_{\text{l}} \tag{4.46}$$

Which is the upper part of equation (4.40). The constraints for the quadratic optimization problem then become:

$$\mathbf{Ax} = \mathbf{J}_{\text{abs}} \begin{bmatrix} \dot{\mathbf{q}}_{\text{r}} \\ \dot{\mathbf{q}}_{\text{l}} \end{bmatrix} = \mathbf{b} = \begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}} \\ \boldsymbol{\omega}_{\text{abs}} \end{bmatrix} \tag{4.47}$$

and

$$\mathbf{Ax} = \mathbf{J}_{\text{rel}} \begin{bmatrix} \dot{\mathbf{q}}_{\text{r}} \\ \dot{\mathbf{q}}_{\text{l}} \end{bmatrix} = \mathbf{b} = \begin{bmatrix} \dot{\mathbf{x}}_{\text{rel}}^{\text{abs}} \\ \boldsymbol{\omega}_{\text{rel}}^{\text{abs}} \end{bmatrix} \tag{4.48}$$

Similarly to `individual manipulation`, see equation (4.26), the target velocities are calculated by:

$$\begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}_\text{t}} \\ \boldsymbol{\omega}_{\text{abs}_\text{t}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}_\text{d}} \\ \boldsymbol{\omega}_{\text{abs}_\text{d}} \end{bmatrix} + \begin{bmatrix} k_{\text{abs}_\text{p}} \mathbf{e}_{\text{abs}_\text{p}} \\ k_{\text{abs}_\text{o}} \mathbf{e}_{\text{abs}_\text{o}} \end{bmatrix} \tag{4.49}$$

and

$$\begin{bmatrix} \dot{\mathbf{x}}_{\text{rel}_\text{t}}^{\text{abs}} \\ \boldsymbol{\omega}_{\text{rel}_\text{t}}^{\text{abs}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{x}}_{\text{rel}_\text{d}}^{\text{abs}} \\ \boldsymbol{\omega}_{\text{rel}_\text{d}}^{\text{abs}} \end{bmatrix} + \begin{bmatrix} k_{\text{rel}_\text{p}} \mathbf{e}_{\text{rel}_\text{p}}^{\text{abs}} \\ k_{rel_o} \mathbf{e}_{\text{rel}_\text{o}}^{\text{abs}} \end{bmatrix} \tag{4.50}$$

The desired velocities $\begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}_\text{d}} & \boldsymbol{\omega}_{\text{abs}_\text{d}} \end{bmatrix}^{\text{T}}$ and $\begin{bmatrix} \dot{\mathbf{x}}_{\text{rel}_\text{d}}^{\text{abs}} & \boldsymbol{\omega}_{\text{rel}_\text{d}}^{\text{abs}} \end{bmatrix}^{\text{T}}$ as well as the target poses $\begin{bmatrix} \dot{\mathbf{x}}_{\text{abs}_\text{t}} & \boldsymbol{\omega}_{\text{abs}_\text{t}} \end{bmatrix}^{\text{T}}$ and $\begin{bmatrix} \dot{\mathbf{x}}_{\text{rel}_\text{t}}^{\text{abs}} & \boldsymbol{\omega}_{\text{rel}_\text{t}}^{\text{abs}} \end{bmatrix}^{\text{T}}$ are obtained with the method described in section 4.2. But instead of using gripper poses as trajectory parameters, the absolute and relative poses are used. The current absolute and relative positions, $\mathbf{x}_{\text{abs}_\text{c}}$ and $\mathbf{x}_{\text{rel}_\text{c}}^{\text{abs}}$, can be calculated with the equations (4.30) and (4.33). The position error can then be calculated with:

$$\mathbf{e}_{\text{abs}_\text{p}} = \mathbf{x}_{\text{abs}_\text{t}} - \mathbf{x}_{\text{abs}_\text{c}} \tag{4.51}$$

$$\mathbf{e}_{\text{rel}_\text{p}}^{\text{abs}} = \mathbf{x}_{\text{rel}_\text{t}}^{\text{abs}} - \mathbf{x}_{\text{rel}_\text{c}}^{\text{abs}} \tag{4.52}$$

For the orientation the current absolute and relative quaternions, $\mathbf{Q}_{\text{abs}_\text{c}}$ and $\mathbf{Q}_{\text{rel}_\text{c}}^{\text{abs}}$, can be calculated with equations (4.34) and (4.31). The difference between the target and the current quaternions is then calculated with equation (4.28). The gain values k_{abs_p}, k_{abs_o}, k_{rel_p} and k_{rel_o} are arbitrary chosen to 1. Now all calculations have been done for the $\mathbf{A}$ matrix and $\mathbf{b}$ vector in for both absolute and relative control, see equations equation (4.47) and (4.48).
For `coordinated manipulation` the control objective is added as two tasks, see table 4.1. The relative and `absolute motion` are added separately, where `relative motion` has a higher priority than `absolute motion`. This is done because the object acts as a physical constraint between the grippers, and if some constraint is violated, it will priorities the `relative motion` over the `absolute motion`.

4.5 Feasibility control objective

Solving the HQP with only the main control objective would result in a controller that ignores the physical limitations of the robot. The limitations that are dealt with in this thesis are joint velocity limitations, joint position limitations, elbow collision and a joint position potential to minimize singularities.

4.5.1 Joint Velocity Limit

The hardware limitations on joint velocities can be seen in table 3.2. The joint velocity limits on all joints are chosen to be 1 rad/s or 57 deg/s. The joint velocity limit is set up as two tasks, one for the lower limit and one for the upper limit. The inequality constraints for upper-velocity limits, see equation (4.53). Where $\mathbf{h}$ is the limits expressed in rad/s.

$$\mathbf{G}\mathbf{x} = \mathbf{I}_{14}\mathbf{x} \leq \mathbf{h} = \begin{bmatrix} 1 \\ ... \\ 1 \end{bmatrix} \tag{4.53}$$

For the lower velocity limits.

$$\mathbf{G}\mathbf{x} = -\mathbf{I}_{14}\mathbf{x} \leq \mathbf{h} = \begin{bmatrix} 1 \\ ... \\ 1 \end{bmatrix} \tag{4.54}$$

4.5.2 Joint Position Limits

The joint position limits are also set up as two tasks, one for lower and one for upper limits. The joint velocity $\dot{\mathbf{q}}$ can be integrated $\Delta_\mathrm{q} = \Delta_\mathrm{T}\dot{\mathbf{q}}$. For upper joint limits, the Δ_q can not be larger than the remaining distance from the current joint position $\mathbf{q}$ to the upper joint limits $\mathbf{q}_\mathrm{b}^\mathrm{u}$. The limits for the joints can be seen in table 3.2 and they are converted to radians. For the upper joint limits, the constraints become:

$$\mathbf{G}\mathbf{x} = \Delta_\mathrm{T}\mathbf{I}_{14}\mathbf{x} \leq \mathbf{h} = \mathbf{q}_\mathrm{b}^\mathrm{u} - \mathbf{q} \tag{4.55}$$

where Δ_T is the step time. Similarly the lower limits can be written as:

$$\mathbf{G}\mathbf{x} = -\Delta_\mathrm{T}\mathbf{I}_{14}\mathbf{x} \leq \mathbf{h} = -\mathbf{q}_\mathrm{b}^\mathrm{l} + \mathbf{q} \tag{4.56}$$

4.5.3 Elbow Proximity Limit

With the inverse differential kinematics, it is only the grippers that are directly controlled. While configuration of the arms are indirectly controlled. Each arm is redundant, having 7 DOF while only 6 DOF are directly controlled. There are often multiple configurations that can achieve the same pose for the grippers. To avoid self-collision for the parts of the arms that are not directly controlled, a minimum proximity for the elbows are introduced. This task has a higher priority than the control objective, see table 4.1. The proximity is done only in the y-axis for simplicity and they are not allowed to be within 0.2 meters from each other. This is set up as two tasks, one for each elbow. The following is described for the right arm but the same applies for the other, but mirrored. The elbow positions can be seen in figure 3.3. A point on the y-axis which the right elbow can not cross $Y_\mathrm{limit}^\mathrm{r}$ is calculated:

$$Y_\mathrm{limit}^\mathrm{r} = y_\mathrm{l} - 0.2 \tag{4.57}$$

y_l is the y-position for the left elbow. From the kinematics, see section 4.1, a Jacobian $\mathbf{J}_\mathrm{elbow}^\mathrm{r}$ have been calculated and only the forward differential kinematics

in the y-direction is used. Similar to how joint position limits were calculated in section 4.5.2 the y-position limit can be expressed as constraints with:

$$\mathbf{G}\mathbf{x} = \Delta_T \mathbf{J}^{\mathrm{r}}_{\mathrm{elbow_y}}\mathbf{x} \leq h = Y^{\mathrm{r}}_{\mathrm{limit}} - y_{\mathrm{r}} \tag{4.58}$$

For the left elbow the constraints instead become:

$$\mathbf{G}\mathbf{x} = -\Delta_T \mathbf{J}^{\mathrm{l}}_{\mathrm{elbow_y}}\mathbf{x} \leq h = -Y^{\mathrm{l}}_{\mathrm{limit}} + y_{\mathrm{l}} \tag{4.59}$$

4.5.4 Joint Positions Potential

The configuration of the robot arms are not directly controlled, this can lead to problems beyond just collisions. There are configurations where the Jacobian can become singular, a solution to the inverse differential kinematics problem may not exist then. These singularities can occur, for example, if the elbow joint is fully extended or multiple joints align. To minimize the extent of singularities occurring and joint limits saturated, a joint position potential is introduced. This task has the lowest priority and only acts when other tasks are not affected by it. The idea is to define a good configuration, that is far away from any singularities and joint position limits but also close to the intended operational state. This position can be thought of as a neutral position, see figure 3.3 for the chosen neutral position. The potential then tries to pull each joint towards the neutral position. For the right arm, the neutral position $\mathbf{q}^{\mathrm{r}}_{\mathrm{N}}$, in degrees, for joint space is chosen to be:

$$\mathbf{q}^{\mathrm{r}}_{\mathrm{N}} = \begin{bmatrix} 40 & -97 & -45 & 57 & -126 & 57 & 0.0 \end{bmatrix}^{\mathrm{T}} \tag{4.60}$$

And the left arm is mirrored in physical space.

$$\mathbf{q}^{\mathrm{l}}_{\mathrm{N}} = \begin{bmatrix} -40 & -97 & 45 & 57 & 126 & 57 & 0.0 \end{bmatrix}^{\mathrm{T}} \tag{4.61}$$

The constraints are then setup as equality constraints, with a single task for both arms.

$$\mathbf{G}\mathbf{x} = \Delta_T \mathbf{I}_{14}\mathbf{x} = \mathbf{h} = \mathbf{K}\left(\begin{bmatrix} \mathbf{q}^{\mathrm{r}}_{\mathrm{N}} \\ \mathbf{q}^{\mathrm{l}}_{\mathrm{N}} \end{bmatrix} - \begin{bmatrix} \mathbf{q}_{\mathrm{r}} \\ \mathbf{q}_{\mathrm{l}} \end{bmatrix}\right) \tag{4.62}$$

Where $\mathbf{K}$ is a gain vector where all values except for joints 6 are set to $\frac{1}{200}$ and joints 6 are set to $\frac{1}{400}$. The difference in gain is because joints 6 should be able to rotate more freely without having a large effect on the overall configuration.

4.6 Safety Checks

There is still a possibility that something can go wrong, either with the HQP solver or that something external has happened. If the constraints in the HQP solver becomes inconsistent, the quadratic programming solver throws an error. In that case, the target velocities are set to 0, resulting in that the robot stops immediately. Human intervention is then needed to reset the system. The constraints can become inconsistent for multiple reasons. The most common is that one of the Jacobians have become singular or that an infeasible instruction has been sent. For trajectory

following, there is a safety check that ensures that the grippers are not too far off the trajectories. If any element in position error, see equations (4.27), (4.51) and (4.52), is larger then 0.01m or if any of the elements in the angular error, see equation (4.28), is larger then 0.1 rad all joint velocities are set to 0. This is important, because the `coordinated manipulation` relies on both arms and if one deviates from the trajectory then the other is affected as well. This scenario can happen if, for example, the collision system in YuMi is triggered. Then the controller should stop the robot. There are also safety checks in the kinematics node that makes sure that both arms have an active EGM session i.e. the arms are active and the communication to them works.

4.7 Grippers

The grippers are controlled through ROS service and ABB rws library. Two modes are used for controlling the grippers, *move to* and *grip in*. The *move to* command moves the grippers to the desired position [mm] and the *grip in* moves the gripper inward until a certain force has been achieved. The force is pre-set in YuMi to 20 N. Each set of trajectory parameters contain information for the grippers. If the position for the gripper in the trajectory parameters is set to 0 then the *grip in* command is used. Otherwise, the *move to* command is used.

5
Computer vision

The computer vision system perceives the environment, using a single Realsense d435i camera, see figure 3.2b. The objective of the computer vision system is to track the DLO in real-time and find the poses of the fixtures. The camera is used with the realsense-ros wrapper[1]. The camera has 5 frames, 2 of which are of importance to this thesis. The frame *camera_link* refers to the entire cameras pose, see figure 3.3. The frame *camera_color_optical_frame* or for short *camera_color* refers to the frame where the color sensor is located. The depth image is aligned to the color camera in the driver.

5.1 AprilTags

The apriltag system can identify the pose of a fiducial marker from a single image. This is used to identify the pose of the camera and the fixtures relative to the robot. A ready made ROS wrapper[2] for the apriltag detection is used. The apriltag detection only needs to be running during initialization, the camera and the fixtures are assumed to be stationary.

5.1.1 Camera Pose

The pose of the camera relative to YuMi is calculated by placing an apriltag with know position relative to the *base frame*. A frame with the same pose as that tag will be referred to as *world frame*, see figure 3.3. The apriltag system outputs a transformation $T_{\text{tag}}^{\text{color}}$ from the *camera_color* frame to the tag. This is then transformed to give the transformation $T_{\text{tag}}^{\text{link}}$ from *camera_link* to the tag by:

$$T_{\text{tag}}^{\text{link}} = T_{\text{color}}^{\text{link}} T_{\text{tag}}^{\text{color}} \tag{5.1}$$

As *world frame* and the tag has the same pose, the pose of the camera can be calculated by the inverse of the transform.

$$T_{\text{link}}^{\text{world}} = T_{\text{tag}}^{\text{link}^{-1}} \tag{5.2}$$

5.1.2 Fixture pose

Each fixture, see figure 3.2a, has a built-in tag. Each tag for the fixtures are unique and contain an ID. The ID is used later in the path planner to distinguish between

[1]For more information visit `https://github.com/IntelRealSense/realsense-ros`
[2]For more information visit`http://wiki.ros.org/apriltag_ros`

the fixtures. The apriltag system is used to find the transform $T_{\text{tag}}^{\text{color}}$ for each tag in the fixtures. As the pose of the camera is already connected to the tf tree, it is enough to find the transform $T_{\text{fixture}}^{\text{color}}$. The transform for the tag to the fixture base $T_{\text{fixture}}^{\text{tag}}$ is only a translation in $x = -0.034[m]$, see figure 3.2a. This gives (5.3), which can be connected to the tf tree.

$$T_{\text{fixture}}^{\text{color}} = T_{\text{tag}}^{\text{color}} T_{\text{fixture}}^{\text{tag}} \tag{5.3}$$

5.2 Deformable Object Tracking

The tracking of the DLO can be seen as two parts, preprocessing and estimation. The preprocessing takes the raw output from the camera and outputs a point cloud of the observation. The estimation fits a point cloud representing the DLO to the point cloud extracted from the camera. In figure 5.1, the overall flow of the computer vision system can be seen. In the preprocessing, a color mask is applied to separate the DLO from the background. The color mask is then used to find the corresponding points in the depth image. A point cloud that represents the raw observation of the DLO can then be extracted. A version of the SPR algorithm is then used to estimate the shape of the DLO from the point cloud. The SPR algorithm uses the previous estimation as the initial guess.

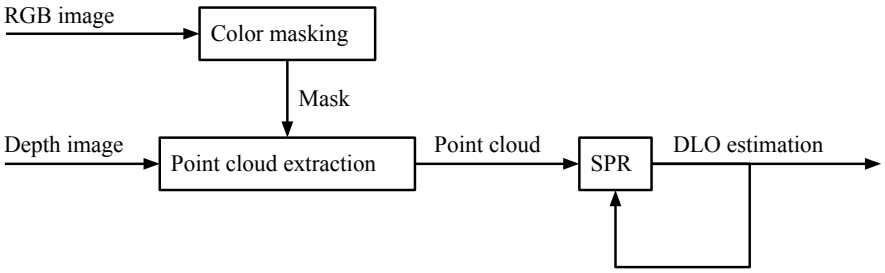


Figure 5.1: Tracking DLO overview

5.2.1 Color Mask

The first step is to separate the DLO, see figure 3.2c, from the background. This is achieved using a simple color filter. The image from the camera is in RGB color space and is transformed into HSV color space. Using upper and lower thresholds for the pixel values the DLO can be masked. The values chosen can be seen in table 5.1.

Table 5.1: HSV values

Threshold	Hue	Saturation	Value
Lower	100	235	100
Upper	120	255	255

The result is an image mask, that masks out anything that is not very close to the color of the DLO.

5.2.2 Point Cloud

The pixel coordinates are extracted from the color mask and the coordinates are converted to homogeneous coordinates p_{color}.

$$p_{\text{color}} = \begin{bmatrix} x_1 & ... & x_n \\ y_1 & ... & y_n \\ 1 & ... & 1 \end{bmatrix} \tag{5.4}$$

The points are then down-sampled to be around 300. The camera projection is then removed from the points. Where K_{color} is the camera matrix for the color camera. The camera intrinsics are then removed:

$$\mathbf{p}_{\text{cal}} = K_{\text{color}}^{-1} p_{\text{color}} \tag{5.5}$$

The depth image has already been aligned to the color image from the driver. The corresponding depth values $\mathbf{p}_{\text{depth}}$ from the depth image can be extracted. These values represent the orthogonal distance from the camera plane and have been converted to meters. To reconstruct the 3D points $\mathbf{p}_{\text{3d}}$ each point in p_{cal} is multiplied by the correspond depth value. $\odot$ represents column-wise multiplication.

$$\mathbf{p}_{\text{3d}}^{\text{color}} = \mathbf{p}_{\text{cal}} \odot \mathbf{p}_{\text{depth}} \tag{5.6}$$

The extracted 3d points are in the camera *color_frame*, they need to be transformed to the base of the robot. First the $\mathbf{p}_{\text{3d}}^{\text{color}}$ converted to homogeneous coordinates $\mathbf{p}_{\text{h}}^{\text{color}}$ by adding a row of ones at the bottom, similar to equation (5.4). The transformation $\mathbf{T}_{\text{color}}$ is obtained through the tf tree, it transforms from the base of the robot to the camera *color_frame*. The transformed points are then calculated.

$$\mathbf{p}_{\text{h}} = \mathbf{T}_{\text{color}}^{\text{base}} \mathbf{p}_{\text{h}}^{\text{color}} \tag{5.7}$$

The 3d points $\mathbf{p}_{\text{3d}}$ are obtained from $\mathbf{p}_{\text{h}}$ by removing the scaling from the homogeneous coordinates. The physical color and depth sensor are not aligned in position and there is a probability that a depth point measurement can miss the DLO. Therefore further filtering is applied, the working surface is assumed to be level, therefore all points that are below the working surface are filtered away.

5.2.3 Structure Preserved Registration

The implementation of the SPR algorithm is based on previous student's work. There are some key differences to the algorithm compared with the original [11]. The main difference is the lack of a simulator, the original implementation is not stable without the simulator for real-time tracking. Due to time restrictions, a modification was made to the SPR algorithm, instead of implementing a simulator. The modification is to keep the local regularization constant from the initial estimation.

The performance of the tracking does therefore not represent the original implementation of the algorithm. The initial estimation has the DLO as linear. Then for each call to the SPR algorithm, the previous estimation is used as an initial guess. The output of the SPR algorithm is an ordered point cloud representation of the DLO.

6

Path planner

In this chapter, the path planner is discussed. The path planner is responsible for generating trajectory parameters to achieve the objective of solving the cable routing problem. The overall structure of the path planner can be seen in figure 6.1. The path planner can be seen as a state machine. During the first state, the path planner gathers information about the environment and sets up a roadmap for solving the objective. The next state generates trajectory parameters for the current task in the roadmap. The trajectory parameters are then validated with a series of tests. If they pass, the trajectory parameters are sent to the controller. If they instead fail, then a stochastic solver is called to solve the problems. If the stochastic solver fails to find a solution within three attempts then the path planner enters an end state. If the trajectory parameters have been sent to the controller, the path planner enters the reactive loop. The reactive loop tracks the progress of the current task that is being executed by the controller. When the task is completed or a problem is detected, a new trajectory is requested. When all the tasks in the roadmap have been completed, the path planner enters the end state.

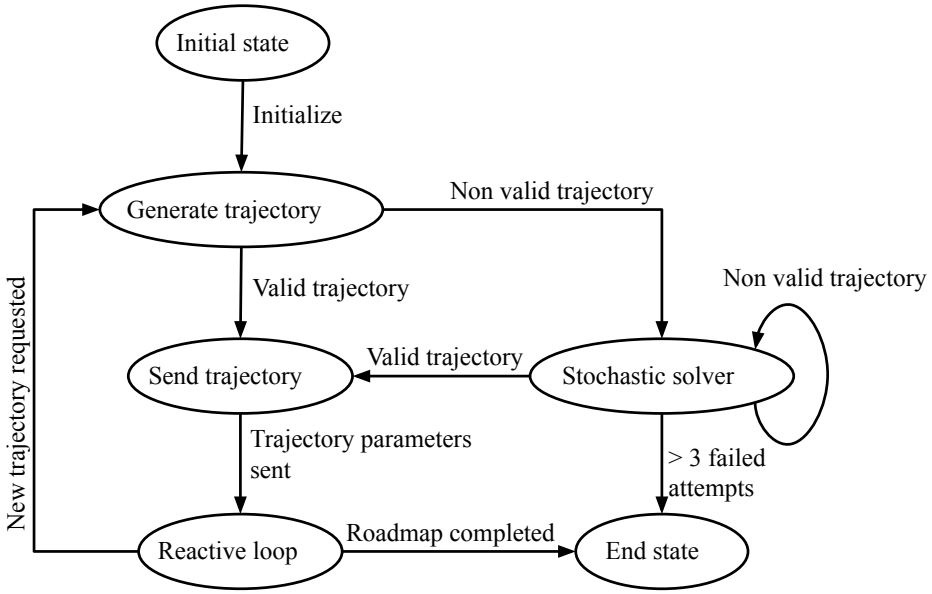


Figure 6.1: Path planner states, overall structure

6.1 Initialization

During initialization, the path planner registers the observed fixtures. Each fixture has a unique identification number embedded in the tag. The sequence of the cable routing is determined by the identification number. The DLO is routed in ascending order through the fixtures. Each fixture has a direction, the DLO will be routed in positive y-axis for the *fixture frame*, see figure 3.2a. The DLO representation also has a direction, as the point cloud is ordered, the positive direction is in ascending order of the points. An overview of the initialization algorithm can be seen in algorithm 4. Each fixture is added to a map, which stores the pose and physical attributes of each fixture. The roadmap for the routing problem consists of tasks. For each fixture there are two tasks, the first task is to grab the DLO and the second task is to clip the DLO into a fixture. Each task is composed of motion primitives, these are discussed in the subsequent section.

Algorithm 4: Algorithm for the initialization

Result: Initialization
for *Number of fixtures* **do**
    Append fixture to map;
    Append `GrabDLO` task to roadmap;
    Append `ClipIntoFixture` task to roadmap;
end

6.2 Trajectory Generation

The problem of routing the DLO through a series of fixtures is divided into tasks. As described in section 6.1 there are two tasks used for each fixture. The objective of trajectory generation is to generate trajectory parameters. An overview of the algorithm can be seen in algorithm 5.
The idea is to first try to generate trajectory parameters from a task in the roadmap. Which task depends on which fixture and what operation is to be performed, discussed further in section 6.3. Each task can compile its motion primitives and generate trajectory parameters, discussed further in section 6.4. The trajectory parameters are then evaluated for any illegal moves. If no problem is detected, then the trajectory parameters are sent to the controller and the path planner enters the reactive loop. If a problem is detected, then new trajectory parameters are generated to solve the problem. There are two cases for generating new trajectories, one is for resetting the orientation of the grippers and the other calls a stochastic solver for rerouting the DLO. The controller always takes the shortest direction between trajectory parameters. A sequence of orientations could result in joint limits being reached, especially if the initial configuration is bad. The problem is most evident for joint 6, as the orientation is only considered around the z-axis in the path planner, the problem is referred to as "over rotation". If over rotation is detected, then the orientation of the gripper is reset to a neutral pose. Then the task is retried. If any other problem is detected the robot is stopped (`HoldPose`), then the stochastic solver is called. In the solver, the environment is formulated as an optimization

problem and the solution should try to manipulate the DLO to a problem-free state. The trajectory parameters from the solver are evaluated. A maximum of 3 attempts to call the solver before the problem is determined to be non-solvable. For both the solver and `ResetOrientation`, a new task is generated outside of the roadmap and the reactive loop is entered.

Algorithm 5: Algorithm for trajectory generation

```
Result: Trajectory parameters
Generate trajectory parameters from a task in the roadmap;
Evaluation = evaluate trajectory parameters;
if Evaluation == over rotation detected then
    Generate trajectory parameters from RestOrientation task;
end
if Evaluation == problem detected then
    Generate trajectory parameters from HoldPose task;
    while number of attempts ≤ 3 do
        Generate solution with solver;
        Generate trajectory parameters from SolverRerouting task;
        Evaluation = evaluate trajectory parameters;
        if Evaluation == pass then
            Break;
        end
    end
    if number of attempts ≥ 3 then
        Enter non solvable end state;
    end
end
Send trajectory parameters to controller;
```

6.2.1 Evaluation

Before trajectory parameters are sent to the controller, they are evaluated to verify that they are executable. There are two separate evaluations, one for trajectory parameters of `individual manipulation` and one for `coordinated manipulation`. The different tests in the evaluation can be seen in table 6.1. If any of the tests fail then the trajectory parameters will not be sent. The evaluation is run on both the trajectories from the roadmap and the solver. Noted that the solver also has internal evaluations.

Table 6.1: Evaluation of trajectory parameters

Tests	Individual	Coordinated
1	WithinReach	WithinReach
2	InsideFixtureSpace	GrippersCross
3	TrajectoriesPassTooClose	
4	GrippersCross	
5	OverRotation	

6.2.1.1 `WithinReach` test

The arms have a maximum reach, see section 3.1.2, at the maximum extension there is a singularity in the kinematics as the elbow joint is fully extended. The test `WithinReach` checks if all trajectory parameters are within the reach of the robot. A margin for the reach is set to 0.03 m. The reachable volume can be modelled as a sphere, the centre of the sphere is close to joint 2, see figure 3.4. The volume of a sphere is a convex set, therefore the entire trajectory will be within reach if all trajectory parameters are within reach, under the assumption of linear motion between poses. The evaluation test the same for both individual and `coordinated manipulation`.

6.2.1.2 `InsideFixtureSpace` test

Each fixture has a virtual sphere around the base of the fixture that the grippers are not allowed to enter. Without it, the grippers can collide with the fixtures. The test is only run for the `individual manipulation` and not for the `coordinated manipulation` when the DLO is clipped into the fixture. The sphere has a radius of 0.06 m and because the centre is at the fixture base, the tested volume acts as a dome or half-sphere.

6.2.1.3 `TrajectoriesPassTooClose` test

The self-collision of the grippers is not dealt with in the controller, a design choice for not limiting the controller's capabilities. It then requires that the trajectory parameters are collision-free. The test assumes linear and constant velocity between the poses in the trajectory parameters and checks the closest distance. If the relative distance is closer than 0.12 m the test fails.

6.2.1.4 `GrippersCross` test

YuMi is a dual-arm robot and sometimes there are scenarios where the grippers need to cross each other on the y-axis. This is a risky manoeuvre, even if the grippers are far apart, other parts of the arm can collide. Some of this is mitigated in the controller, the elbows have a minimal allowed proximity to each other. Though other parts of the arms that are close to the grippers can not always be positioned in a collision-free configuration. Therefore a limit on how far the grippers can cross each other is introduced. The limit is set up as an angle limit, where the angle between the grippers has to be less than 20 degrees beyond the crossing point.

6.2.1.5 `OverRotation` test

Lastly for individual motion, there is a possibility that the sequence of orientations in the trajectory parameters can exceed the limitations of joint 6. The test checks if the sequence of orientations keeps the joint 6 within ± 225 degrees.

6.3 Reactive Loop

When the controller is executing the trajectory parameters, the path planner is in the reactive loop. The overall algorithm can be seen in 6. Each task is composed of several motion primitives. The path planner receives information from the controller on which motion primitive it is currently being executed. Each motion primitive can have a test, the test verifies if a motion primitive is performed correctly during run-time. Due to time and visual tracking limitations, not every motion primitive has a test. The specifics of the tasks are discussed in section 6.4.
The path planner always starts with the first task in the roadmap. If no problems are detected in the trajectory generation and the current task is completed, then the task is changed to the next task in the roadmap. When all the tasks in the roadmap have been completed, then the DLO routing is completed and the path planner enters an end state. If a problem is detected from the current motion primitive, then a new trajectory is requested. If a problem in the trajectory generation is detected then a new temporary task is generated outside of the roadmap. When the temporary task is completed, the path planner returns to the roadmap, it always returns to the most recent `GrabDLO` task. If instead a problem is detected during the execution of the temporary task, then trajectory generation is called as before.

Algorithm 6: Algorithm for the reactive loop

```
Reactive loop;
while not in end state do
    Check motion primitive in task;
    if problem detected then
        Generate new trajectory;
    end
    if task is completed then
        if roadmap is completed then
            Enter end state;
        else
            New task from roadmap;
            Generate new trajectory;
        end
    end
end
```

6.4 Tasks

The tasks are building blocks for the path planner. There are five different tasks used by the path planner, they are `GrabDLO`, `ClipIntoFixture`, `ResetOrientation`, `SolverRerouting` and `HoldPose`. Each task contains a list of motion primitives and tests that together comprise a task. Each motion primitive contains the information necessary to generate a pose for the trajectory parameters. When trajectory parameters are requested for a task by the trajectory generation, the task compiles the motion primitives and creates the trajectory parameters.

A simplified algorithm for how to trajectory parameters are generated can be seen in algorithm 7. The state of the system represents all information available i.e. the robots configuration, DLO estimation, the fixtures and the state of the path planner. The motion primitives in a task are calculated in series. Pose parameters contain the target pose for each gripper or absolute/relative pose depending on the control mode and the time it should take to reach the pose. The pose parameters are then appended to the trajectory parameters. The state is then updated and the next motion primitive calculation is based on the previous state. The trajectory parameters are returned when all motion primitives in a task have been added. Each task will be now be discussed individually.

Algorithm 7: Calculates trajectory parameters for a task

Result: Trajectory parameters
state = current state;
for *number of motion primitives* **do**
    pose parameters = motion primitive(state);
    Append pose parameters to the trajectory parameters;
    Update state from pose parameters;
end
return Trajectory parameters;

6.4.1 `GrabDLO` Task

This task is responsible for picking up the DLO. The grippers are controlled separately i.e. `individual manipulation`. The DLO can be expressed as a one-dimensional function $\mathbf{p}_{\text{DLO}} = DLO(s)$, where s is the distance along the rope in the positive direction and $\mathbf{p}_{\text{DLO}}$ is the pose of the DLO at that point.

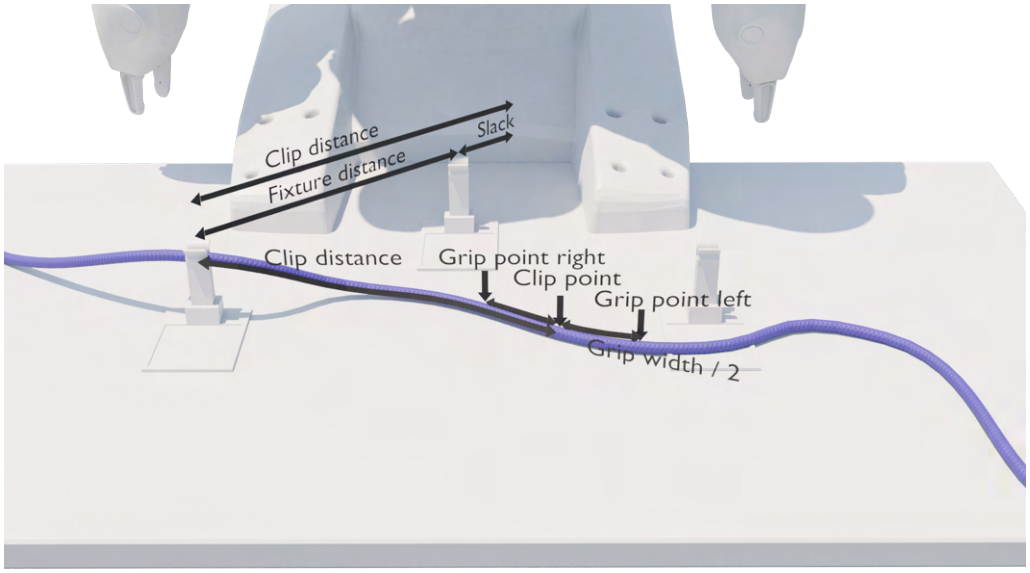


Figure 6.2: Rendition of how the grip points are calculated

A point called the clip point s_{clip} is defined, which represent the point on the DLO

that should be clipped into the fixture. The principles for how the grip points are calculated can be seen in figure 6.2. The clip point is calculated as the distance between the target fixture and the previous fixture with some margin. The margin is referred to as slack. For the first fixture where there is no previous fixture, the slack variable defines the clip position. The slack variables have been chosen to be 0.15m for the first fixture and 0.04 m for the following fixtures.
From the clip point s_{clip} the position for each gripper (s_{right}, s_{left}) on the DLO can be calculated.

$$\begin{aligned} s_{\text{right}} &= s_{\text{clipp}} \pm \frac{GripWidth}{2} \\ s_{\text{left}} &= s_{\text{clipp}} \pm \frac{GripWidth}{2} \end{aligned} \tag{6.1}$$

A variable $GripWidth$ is predefined and defines how far apart the grippers should grip the DLO, the grip width is set to 0.15 m. The direction the DLO should be gripped in, i.e. should the left gripper be in the positive or negative direction of the clip point, has to be determined. The grippers are not allowed to cross each other too far on the y-axis. The direction is determined from the orientation of the DLO. If the DLO orientation is within where both grippers can access on both sides, the orientation of the fixture determines the direction.
From the right and left grip points s_{right} and s_{left}, the poses for both grip points can be calculated with $\mathbf{p}_{right} = DLO(s_{\text{right}})$ and $\mathbf{p}_{left} = DLO(s_{\text{left}})$. From the poses for each grip point, the trajectory parameters can be calculated. The motion primitives that builds up this task can be seen in table 6.2. The `GoToHeight` motion primitive takes the current pose of the state and only changes the height. The target height is set to 0.1 m and at that height, the grippers clear the fixtures. `OverGrippPoints` takes the gripper over the target grip points $\mathbf{p}_{right}$ and $\mathbf{p}_{left}$ with the height of 0.1 m over the grip points. `OnGrippPoints` lowers the grippers to a height where the DLO is in between the fingers of the gripper. Both `OverGrippPoints` and `OnGrippPoints` have a test to check if the DLO has moved. `GripDLO` closes the fingers of the grippers and grabs the DLO. `LiftDLO` lifts the DLO to 0.1 m, it also checks if the DLO is lifted, if not then it can be assumed that the gripping has failed.

Table 6.2: Motion primitives

Order	Motion primitive	Tests
1	GoToHeight	-
2	OverGripPoints	DLOMoved
3	OnGripPoints	DLOMoved
4	GripDLO	-
5	LiftDLO	DLOInGrippers

6.4.2 `ClipIntoFixture` Task

This task assumes that the DLO is already in the grippers from the previous task in the roadmap. This task uses `coordinated manipulation`. The relative control is set to have a relative distance between the grippers, the width is 0.5 cm larger

than the grip points. This is for some extra tension in the DLO when clipping it into the fixture. The motion primitives that builds up this task can be seen in table 6.3. `OverFixture` puts the *absolute frame* over the *fixture frame*. As the clip point is the average of the grip points, the clip point and the *absolute frame* are aligned. `LowerOnFixture` lower the DLO and clips it into the fixture. `OpenGrippers` releases the DLO. Lastly, the grippers are raised to 0.1 m. This task does not have any tests and is therefore always assumed by the path planner to be completed successfully.

Table 6.3: Motion primitives

Order	Motion primitive	Tests
1	OverFixture	-
2	LowerOnFixture	-
3	OpenGrippers	-
4	GoToHeight	-

6.4.3 `ResetOrientation` Task

This task keeps the position of the grippers but changes the orientation to a neutral state. The motion primitives can be seen in table 6.4. The reason for two `ResetOrientation` motion primitives is to guarantee that the closest direction of the rotation is known, where the first acts as an intermediate step. The task uses `individual manipulation` and does not have any tests.

Table 6.4: Motion primitives

Order	Motion primitive	Tests
1	Reset0rientation	-
2	Reset0rientation	-

6.4.4 `SolverRerouting` Task

The stochastic solver, see section 6.5, outputs a solution. The solution can either be for `individual` or `coordinated manipulation`. The task converts the solution to trajectory parameters. Depending on the solution the task can either generate trajectory parameters for `individual` or `coordinated manipulation`. The motion primitives can be seen in table 6.5.
For `individual manipulation`, the solver outputs a grip point and the target pose for each gripper as well if each gripper is active. If one gripper is inactive then it will move out of the way and only the other gripper will manipulate the DLO. If both grippers are active then both grippers will manipulate the DLO. The motion primitives 1 - 5 are similar to the `GrabDLO` task, the only difference is that the grip points are determined by the stochastic solver. When the DLO has been lifted, it is then taken to a new position `GoToPosition`, this position is calculated by the stochastic solver. The grippers are then lowered (`GoToHeightDown`) to be 0.03 m over the workspace. Then the DLO is released and the grippers are returned to the height of 0.1m.

For `coordinated manipulation`, it is assumed that the DLO is already in the grippers. There is only one case solved with `coordinated manipulation`. That is when the DLO has to be turned around to and the grippers switch position. This is not possible in one move and this task performs the necessary intermediate step. The solver will find a configuration for the DLO where the grippers can swap position. `GoToPose` takes the DLO to the desired position, it is then lowered down and released. The grippers are raised and lastly, the grippers are rotated to be able to swap position with `RotateToOpposite`. The normal `GrabDLO` can from this configuration pick up the DLO correctly.

Table 6.5: Motion primitives

Order	Motion primitive individual	Tests	Motion primitive coordinated	Tests
1	GoToHeightUp	-	GoToPose	-
2	OverGripPoints	DLOMoved	GoToHeightDown	-
3	OverGripPoints	DLOMoved	ReleaseDLO	-
4	GripDLO	-	GoToHeightUp	-
5	LiftDLO	DLOInGrippers	RotateToOpposite	-
6	GoToPosition	-		
7	GoToHeightDown	-		
8	ReleaseDLO	-		
9	GoToHeightUp	-		

6.4.5 `HoldPose` Task

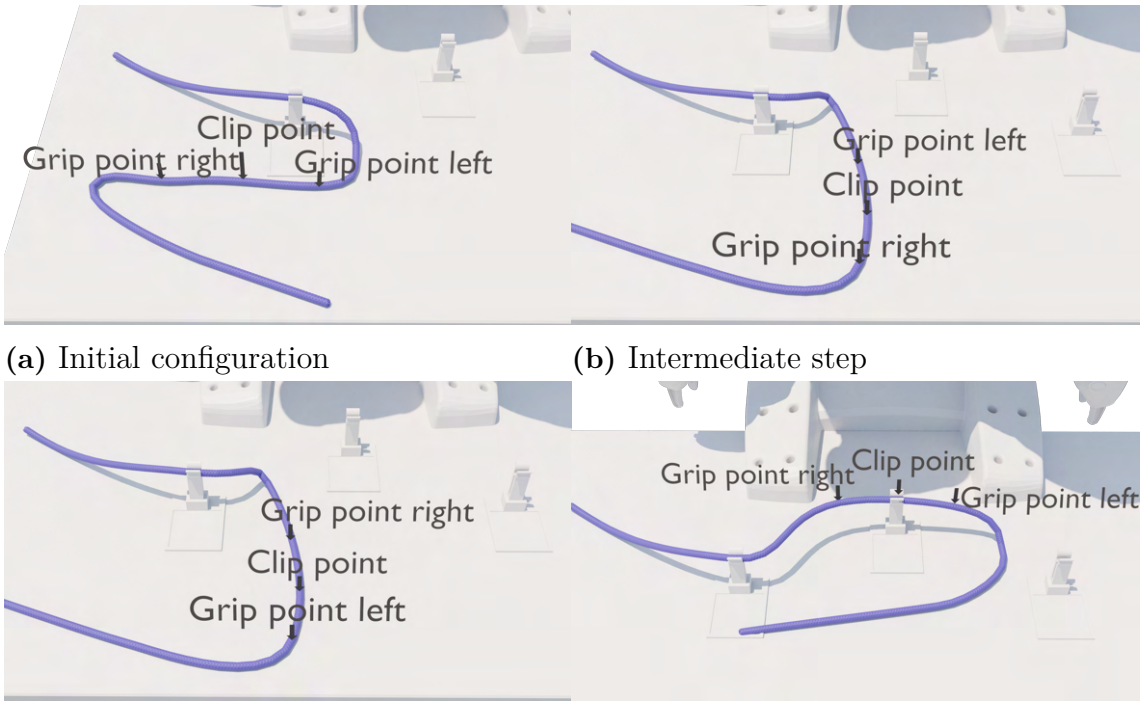
The HoldPose task outputs trajectory parameters that contain the same pose as the robots current configuration. This task is used to stop the robot.

Table 6.6: Motion primitives

Order	Motion primitive	Tests
1	HoldPose	-

6.5 Stochastic Solver

When a problem is detected in the trajectory parameters from trajectory generation, the stochastic solver is called. There are two main modes for the solver. If it is the `GrabDLO` task that failed, then the solvers objective is to manipulate the DLO in such a way that both grip points are reachable. The task can then be retried. When the `ClipIntoFixture` task fails and it is because the DLO needs to be turned around. Then the objective is to find an intermediate step where the grippers can swap position, see figure 6.3.



(a) Initial configuration **(b)** Intermediate step

(c) Intermediate step, grippers swapped **(d)** Final configuration

Figure 6.3: Turning around DLO with intermediate step

The overall algorithm for the stochastic solver can be seen in algorithm 1 and is described in section 2.2.1. The optimization algorithm is the same for both `individual` and `coordinated manipulation`, the difference is in the individuals and evaluation. The optimization is then done for 20 generations and 100 individuals, see table 6.7. When all generations have been completed, the best individual is returned as the solution.

Table 6.7: Parameters

Parameter	Individual	Coordinated
Generations	20	20
Individuals	100	100
Genes	5	3
P cross over	0.9	0.9
P gene mutation	0.2	0.33

6.5.1 Individuals

Each individual encodes the information for a solution, is table 6.8 the different genes for the individuals can be seen. For `coordinated manipulation`, there are 3 genes, two for position and one for orientation. The objective is to find a configuration similar to what can be seen in figure 6.3b. The problem is seen as two dimensional, as the height can be predefined. The rotation only needs to represent orientation around the z-axis as all other rotation axes are kept constant. This reduces the

number of variables that are solved. For `coordinated manipulation` it is assumed that the DLO is in the grippers and already lifted to a height, then the solution only needs to represent the final pose. For `individual manipulation`, the grip points for each gripper on the DLO and the final pose is solved. The final pose is represented as an absolute position and orientation, similar to `coordinated manipulation`. The individual poses for the grippers can be extracted from the absolute pose and known grip width. The grip width for `individual manipulation` is the distance between the grip points i.e. gene 1 and 2.

Table 6.8: Individuals

Gene	Individual	Coordinated
1	Solver grip point right	Absolute position x
2	Solver grip point left	Absolute position y
3	Absolute position x	Absolute rotation
4	Absolute position y	
5	Absolute rotation	

6.5.2 Generate initial population

The initial population should have a wide spread in the optimization space. For `individual manipulation`, the grip points are generated uniformly within a space on the DLO, see figure 6.4. The upper and lower limits for each solver grip point is decided by the original grip points and if there is a previous fixture already used.

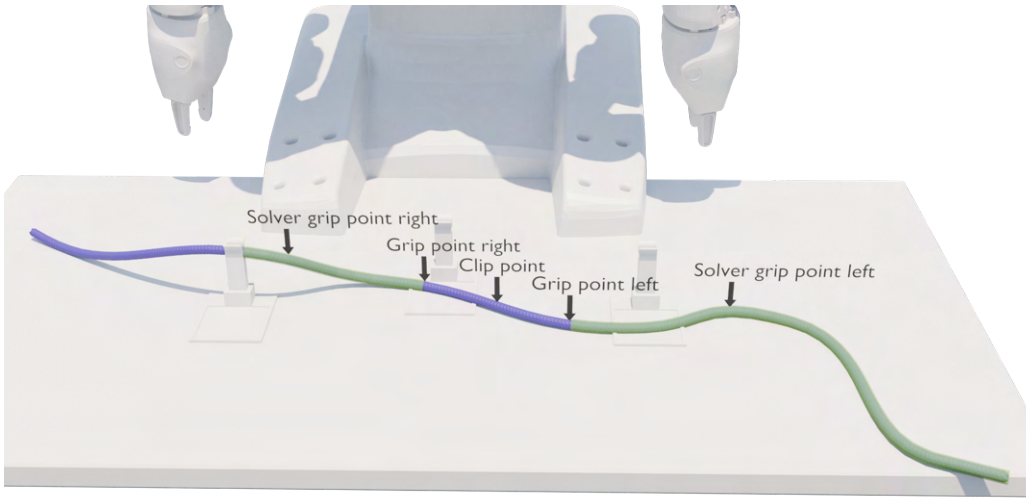


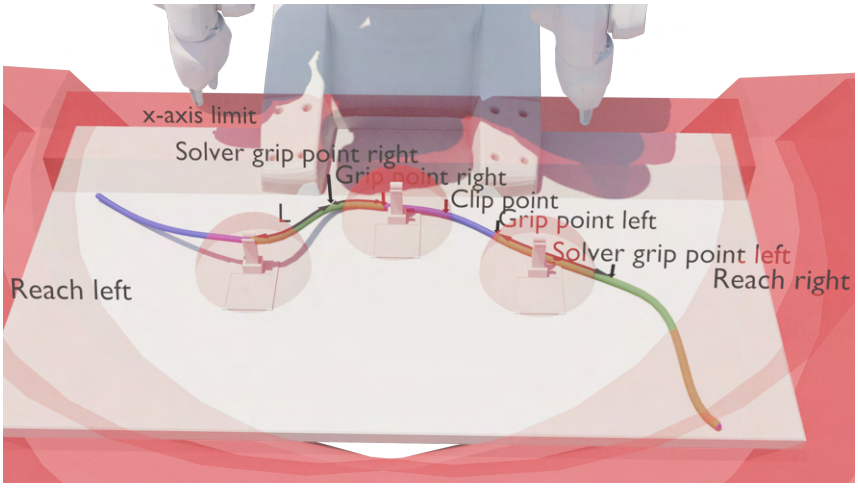
Figure 6.4: Rendition of the range for the solver grip points

The pose is generated from a normal distribution for the current pose. The solution should preferably be close to the current pose. The standard deviation used for the initial pose is set to four times the standard deviation used for mutation, see table 6.11. For `coordinated manipulation`, the objective is instead to have the DLO

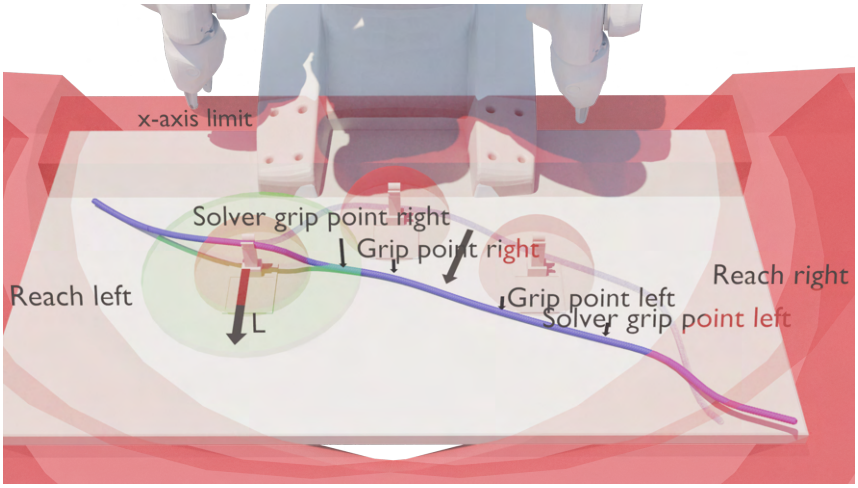
in the direction of the x-axis of the *base frame*. Thus the orientation is generated along the x-axis from a normal distribution. It can be in both positive and negative direction of the x-axis.

6.5.3 Evaluation fitness score

At the core of the optimization algorithm is the evaluation function. The evaluation function takes an individual and returns a fitness score. The evaluation is slightly different for the two cases, `individual` or `coordinated manipulation`. The fitness score for an individual is composed of the sum of several sub evaluations.
For `individual manipulation`, the different sub evaluations that make up the fitness score can be seen in table 6.9, where (x2) represents that the evaluation is done per gripper. The solver grip points are evaluated for both the initial and final configuration. The goal is to get the grip points to a valid configuration with minimal manipulation. The figure 6.5 illustrates some of the evaluations for both the initial and final configuration.



(a) Initial configuration for solver



(b) Final configuration for solver

Figure 6.5: Zones for optimization, `individual manipulation`

If a gripper is to close to a fixture there is a risk for collision, `InsideFixtureSpace` checks if a point is too close to a fixture. Each fixture has a sphere with radius 0.07 $[m]$ around the base of the fixture, see figure 6.5, and d_{fixture} represents the distance to a fixture. Each arm has a maximum reach, `WithinReach` checks if a point is within the reach of an arm. The reach is defined as a sphere around each shoulder of the robot. d_{reach} is the distance from the centre of the sphere to the end of the arm (*joint 6*). Parts of the constrained volume for the grippers can be seen in figure 6.5, where reach left is for the left gripper and reach right is for the right gripper. If the grip points are too close to each other then the grippers could collide. `GrippersTooClose` checks if two points are closer then $0.12[m]$. The `x-axis limit` is there to prevent the gripper to reach behind the workspace and potentially collide with the body of the robot, see figure 6.5. x_{gripper} is the x position of a gripper in the *base frame*. `GripperCross` checks if the grippers cross each other too far on the y-axis, similar to section 6.2.1.4. The angle α is the angle from the right to left gripper and $\alpha = 0$ means that the grippers are aligned on the y-axis.

Table 6.9: Evaluation for *individual manipulation*, fitness score

Nr	Evaluation	Criteria	Score pass	Score not pass
1	Initial configuration			
1.1	Solver grip points			
1.1.1	- InsideFixtureSpace (x2)	$d_{\text{fixture}} \geq 0.07[m]$	0	$-2 + d_{\text{fixture}}$
1.1.2	- WithinReach (x2)	$d_{\text{reach}} \leq 0.529[m]$	1	$-d_{\text{reach}}$
1.1.3	- GippersTooClose	$d_{\text{grippers}} \leq 0.12[m]$	0	-2
1.1.4	- x-axisLimit (x2)	$x_{\text{gripper}} \leq 0.13[m]$	0	$-1 + x_{\text{gripper}}$
1.1.5	- GrippersCross	$-110 \leq \alpha \leq 110[deg]$	0	-2
1.1.6	- DistanceFromGripPoints (x2)	True	$-d_{\text{gripPoint}}$	
2	Final configuration			
2.1	Grip points			
2.1.1	- InsideFixtureSpace (x2)	$d_{\text{fixture}} \geq 0.07[m]$	0	$-2 + d_{\text{fixture}}$
2.1.2	- WithinReach (x2)	$d_{\text{reach}} \leq 0.489[m]$	1	$-d_{\text{reach}}$
2.1.3	- GippersTooClose	$d_{\text{grippers}} \leq 0.12[m]$	0	-2
2.1.4	- x-axisLimit (x2)	$x_{\text{gripper}} \leq 0.13[m]$	0	$-1 + x_{\text{gripper}}$
2.1.5	- DistanceMoved (x2)	True	$-2d_{\text{moved}}$	
2.1.6	- GrippersCross	$-110 \leq \alpha \leq 110[deg]$	0	-2
2.2	Solver grip points			
2.2.1	- InsideFixtureSpace (x2)	$d_{\text{fixture}} \geq 0.07[m]$	0	$-2 + d_{\text{fixture}}$
2.2.2	- WithinReach (x2)	$d_{\text{reach}} \leq 0.529[m]$	1	$-d_{\text{reach}}$
2.2.3	- x-axisLimit (x2)	$x_{\text{gripper}} \leq 0.13[m]$	0	$-1 + x_{\text{gripper}}$
2.2.4	- GrippersCross	$-110 \leq \alpha \leq 110[deg]$	0	-2
2.2.5	- WithinDLOConstraint	$d_{\text{fixture}} \leq L - 0.02[m]$	0	$-2 + d_{\text{fixture}} - L$
2.4	RotationChange	True	$\frac{0.5}{r_{\text{change}}+1}$	
3	NotValidPenalty	Solver grip points valid	0	-5

The solver grip points should preferably be close to the grip points used to clip the DLO into the fixture. `DistanceFromGripPoints` gives a penalty based on the distance on the DLO between the solver grip points and the grip points. If the DLO is already clipped into a previous fixture then there is a constraint for how

the DLO can be manipulated. `WithinDLOConstraint` checks the length L of the DLO between the fixture and the closest solver grip point, see figure 6.5a. Then that solver grip point is then constrained to be within a distance from the fixture, see the green area in figure 6.5b. The final configuration should preferably be close to the initial configuration. `DistanceMoved` gives a penalty for how far each grip point has moved and `RotationChange` gives a score for how close the orientation is to the initial configuration. `NotValidPenalty` gives an extra penalty if both the solver grip points are not valid. A solver grip point is only valid if all the related evaluations are valid.

For `coordinated manipulation`,the different sub evaluations that make up the fitness score can be seen in table 6.10. The objective is to find an intermediate step where the grippers can change position to turn the DLO around, see figure 6.3. The DLO is assumed to already be in the grippers from the previous task. Many of the evaluations are equivalent as for `individual manipulation`, but there are some unique ones. Because the objective is to swap the grippers, see figures 6.3b and 6.3c, both of the grippers have to be able to reach both of the grip points. `WithinReachOppoiste` checks if a gripper can reach the other grip point. The grippers can swap position when the DLO is closely aligned with the x-axis. In figure 6.3b, the DLO is said to be aligned with the x-axis. The fitness score for the individual is then the sum of the evaluation scores.

Table 6.10: Evaluation for *coordinated manipulation*, fitness score

Nr	Evaluation	Criteria	Score pass	Score not pass
1	Final configuration			
1.1	Grip points			
1.1.1	- InsideFixtureSpace (x2)	$d_{\text{fixture}} \geq 0.07[m]$	0	$-2 + d_{\text{fixture}}$
1.1.2	- WithinReach (x2)	$d_{\text{reach}} \leq 0.499[m]$	1	$-d_{\text{reach}}$
1.1.3	- WithinReachOpposite (x2)	$d_{\text{reach}} \leq 0.499[m]$	1	$-d_{\text{reach}}$
1.1.4	- x-axisLimit (x2)	$x_{\text{gripper}} \leq 0.13[m]$	0	$-1 + x_{\text{gripper}}$
1.1.5	- DistanceMoved (x2)	True	$-2d_{\text{moved}}$	
1.1.6	- GrippersCross	$-110 \leq \alpha \leq 110[deg]$	0	-2
1.1.7	- WithinDLOConstraint	$d_{\text{fixture}} \leq L - 0.02[m]$	0	$-2 + d_{\text{fixture}} - L$
1.3	AlignedWithX-axis	$\|\|\alpha\| - 90\| \leq 20[deg]$	$1 + \frac{2}{\|\|\alpha\|-90\|+1}$	-1

When the score for each individual has been calculated, they are shifted and normalized. Let F_{i} represent the fitness score for individual i, then the normalized fitness score F_{i}^{N} and be calculated by:

$$F_{\text{i}}^{\text{N}} = \frac{F_{\text{i}} - \min(F)}{\sum_{\text{i}=1}^{\text{n}} F_{\text{i}} - \min(F)} \tag{6.2}$$

where

$$\sum_{i=1}^{n} F_{\text{i}}^{N} = 1 \tag{6.3}$$

Sampling is then done with roulette wheel sampling. The probability that an individual is sampled p_{i} equals the normalized fitness score F_{i}^{N} for that individual.

6.5.4 Mutation

The mutation is done to reduce the risk of getting stuck in a local maximum. The mutation is done per gene and with a probability that each gene is mutated. The probability is set to be 1 over the number of genes in a individual, $P_{\text{Mutation}} = \frac{1}{N_{\text{Genes}}}$ [33]. The mutation is sampled from a normal distribution, with the current gene value as mean and a standard deviation. The standard deviation for each gene can be seen in table 6.11. For `coordinated manipulation`, there is a probability $P = 0.5$ that the orientation is flipped 180 degrees, but only if there is a mutation on that gene.

Table 6.11: Mutation STD

Gene	Individual σ	Coordinated σ
1	0.05 $[m]$	0.05 $[m]$
2	0.05 $[m]$	0.05 $[m]$
3	0.05 $[m]$	10 $[deg]$
4	0.05 $[m]$	
5	15 $[deg]$	

7
Results

In this chapter, the experimental results are presented. The robot is connected to a PC through an Ethernet cable. The specification of the PC used can be seen in table 3.1. The results from the computer vision system and the controller are first presented separately and then the entire system is evaluated with a cable routing test.

7.1 Apriltags

The apriltag system is used for both estimating the pose of the camera and identify the fixtures. Results from a test can be seen in table 7.1, where a fixture with a known position where estimated. The cameras pose is estimated from a tag with 123 × 123 mm. The tag on the fixture has a size of 40 × 40 mm, see figure 3.2a. The opening on the fixture is 16mm and the tolerance for inserting the DLO into the fixture is about ±7 mm, in the fixtures x-axis. The result from the test shows that there is a probability that the pose estimation of the fixtures can be outside the necessary tolerances, assuming that everything else in the system is fully accurate. Through the pose estimation is within a reasonable range for single experiments.

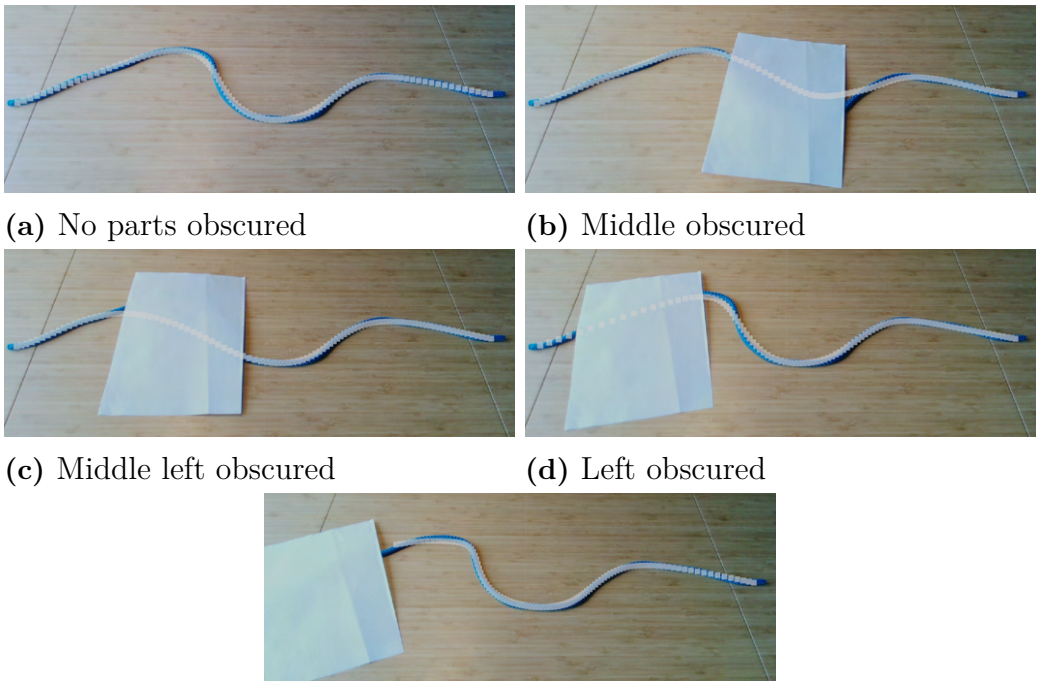
Table 7.1: Fixture position estimation

Position $(x, y, z)_{world}$	Estimated position	$\|\|Error\|\|_2$ $[m]$
(0, 0, 0)	(6e-4, 1.3e-3, 1.8e-3)	0.0023
(-0.3, 0, 0)	(0.2941, 9.1e-3, 1.09e-2)	0.0153
(0.3, 0, 0)	(0.3012, 1.0e-3, 2.1e-3)	0.0026
(0, 0.2, 0)	(4.1e-3, 0.20037, 3.2e-3)	0.0052
(0, -0.2, 0)	(-1.5e-3, -0.2018, -9e-4)	0.0025

7.2 DLO tracking

The camera view overlaid with the DLO estimation can be seen in figure 7.1. When no part of the DLO is occluded, see 7.1a, the estimations follow the shape of the DLO. The estimations are smoother than the true DLO, which can be seen in the curves. As describes in section 5.2.3, the SPR method is modified, with the local regularization trying to keep the estimation in a straight line. Some of the smoothing effects can be attributed to the local regularization. It can also be observed that the last bit of the DLO ends is not estimated. The SPR algorithm can also estimate

the DLO even if parts are occluded. From 7.1b and 7.1c it can be seen when bends are occluded. Although the bend is predicted, the estimation for the parts where the DLO is occluded is smoothed and not as accurate. The parts of the DLO not occluded is still estimated well. In 7.1d a straight portion of the DLO is occluded, even if there is not much of the DLO visible on the left side of the occlusion, it is still estimated. Estimation for occluded sections is more accurate for straight parts than for bends. When the ends of the DLO is completely covered, see figure 7.1e, the flaws of the SPR algorithm can be seen. The SPR algorithm does not keep the DLO length constant and the estimation ignores the occluded part. When running the implemented version of the SPR algorithm with the entire system, the average update rate for the tracking was around 6 Hz. The update rate can momentarily drop even lower and cause a significant lag in the estimation.



(a) No parts obscured

(b) Middle obscured

(c) Middle left obscured

(d) Left obscured

(e) Left end obscured

Figure 7.1: DLO estimation with SPR

7.3 Controller

The controller generates trajectories from the trajectory parameters provided by the path planner and then follows them. All the results in this section are from experiments using the real robot while solving the cable routing problem. The controller is running at 50 Hz. The recorded pose of the grippers is relying on the joint encoders and therefore does not accurately represent the real world pose as the encoder errors are ignored. Though it does accurately represent the pose the controller uses as feedback.
In figure 7.2 the trajectories and gripper position for a `GrabDLO` task can be seen. The task is using the `individual manipulation` mode, thus the arms are controlled

separately. It can be observed that the shape of the trajectory segments follows the cubic shape described in section 4.2. It can be seen that the recorded position of the grippers follows the trajectory.

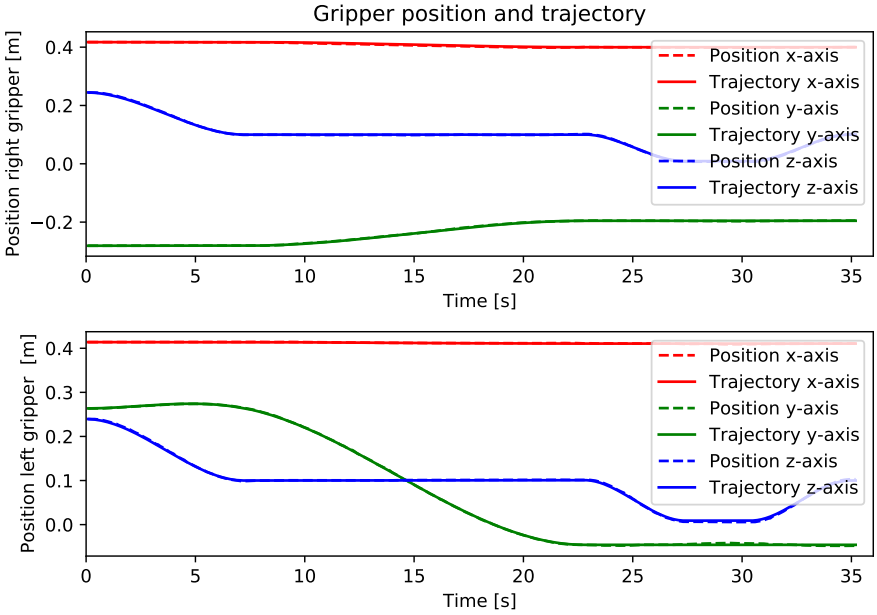


Figure 7.2: `Individual manipulation` position and trajectory

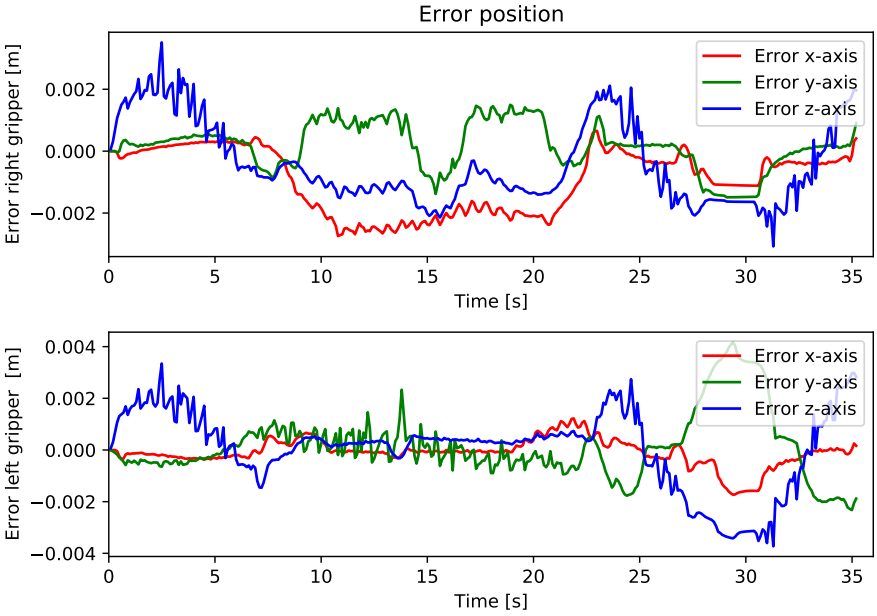


Figure 7.3: `Individual manipulation` position error

In figure 7.3 the positional error can be seen. The controller follows the trajectory well, even if very little parameter tuning was made. From table 7.2 the root mean

square error (rmse), mean and standard deviation (std) can be seen. The mean is close to zero which shows that there is very little bias for a sequence of motions. The standard deviation is just over a millimetre and the rmse is also small, relative to the precision required for the task. Through the grippers are moved at a very slow velocity, an average velocity of 2 cm/s. At these velocities, the dynamic effects have a low effect and are ignored in the controller.

The angular error can be seen in figure 7.4. The angular error is expressed as the angular error around each axis in the *base frame*. From table 7.3 it can bee seen that the rmse is 0.00192 radians or 0.11 degrees. With both the position and orientation error it can be concluded that the pose for each gripper follows the trajectory well. There is also no significant difference in the performance between the right and left side.

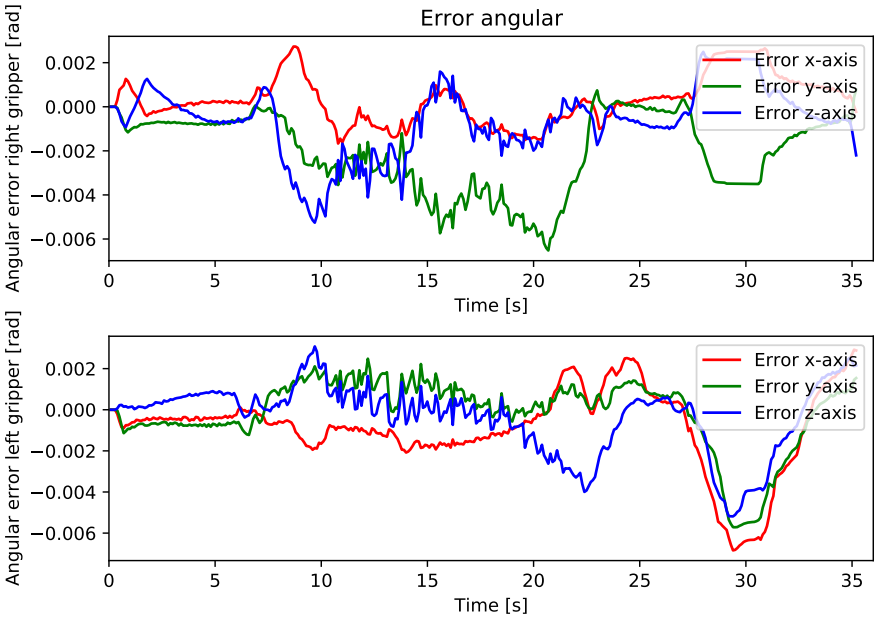


Figure 7.4: `Individual manipulation` orientation error

For `coordinated manipulation`, when the arms are controlled as one system, the trajectories and position can be seen in figure 7.5. For `absolute motion`, the pose is the average of the grippers and for `relative motion`, the pose is defined as the difference between the grippers in the *absolute frame*. The data for `coordinated manipulation` is recorded with the `ClipIntoFixture` task and a DLO is present in the grippers.

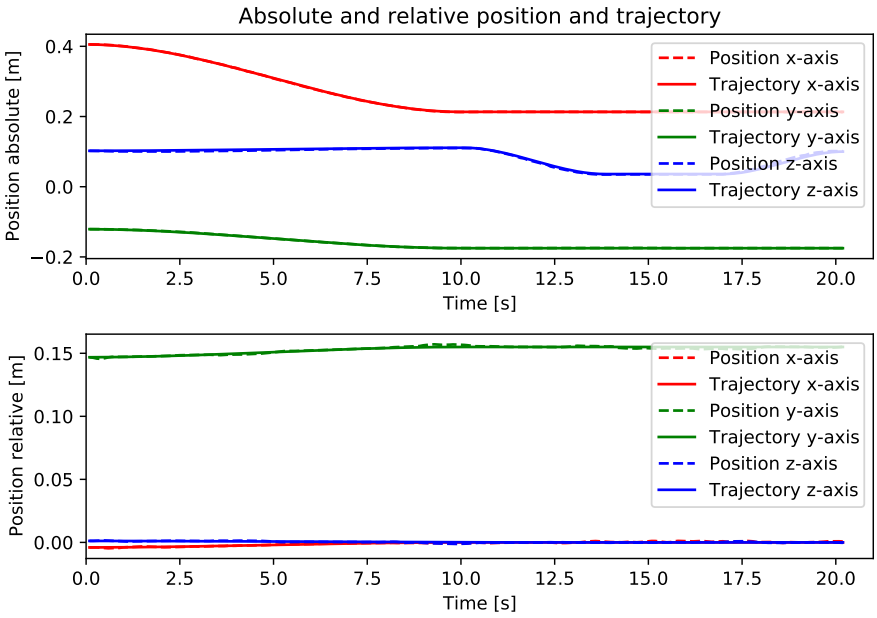


Figure 7.5: `Coordinated manipulation` position and trajectory

The position error for `Coordinated manipulation` can be seen in figure 7.6. The absolute position is the average of the position of the grippers. The relative position is the difference between the grippers in the absolute frame.

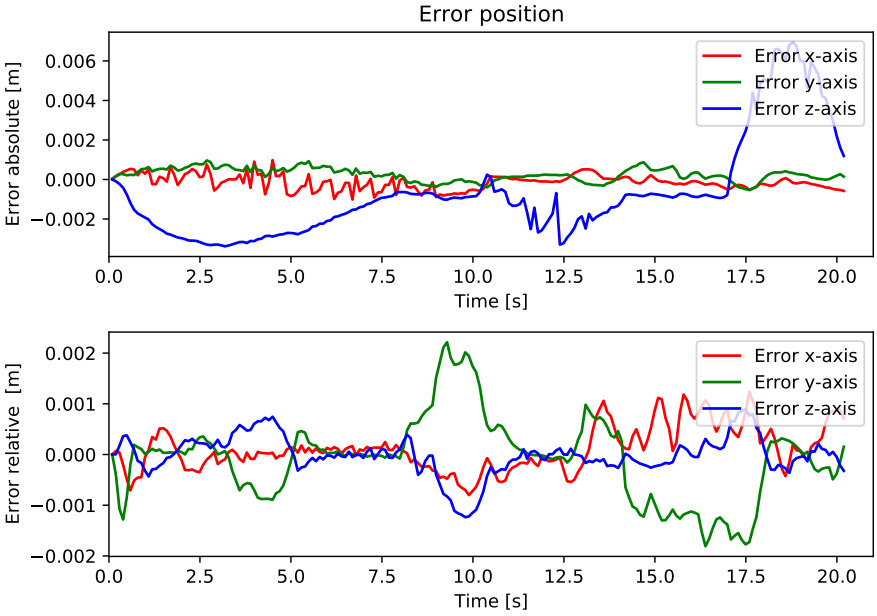


Figure 7.6: `Coordinated manipulation` position error

The angular error can be seen in 7.7. For both the `absolute` and `relative motion` the angular error stays under 0.3 degrees.

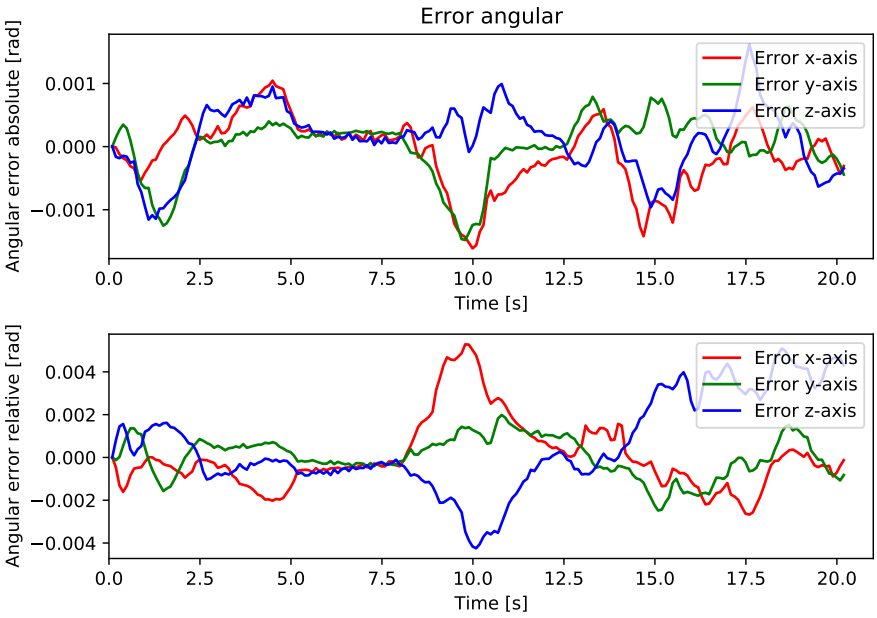


Figure 7.7: `Coordinated manipulation` orientation error

From the tables 7.2 and 7.3 and the figures it can be concluded the accuracy of the `coordinated manipulation` is similar to the accuracy for `individual manipulation`. The results show data from when cable routing was performed. The results reflect the operational accuracy, though it can be speculated that the DLO did not have a significant impact on the accuracy.

Table 7.2: Error position $[m]$

	Individual	Coordinated
rmse	0.00119	0.00116
mean	-0.00018	-0.00010
std	0.00117	0.00116

Table 7.3: Error angular $[rad]$

	Individual	Coordinated
rmse	0.00192	0.00121
mean	-0.00063	0.00012
std	0.00181	0.00126

7.4 Cable routing

In this section, the system is evaluated through a cable routing problem. The results will be presented with images from the execution and tables representing the path planners state. The test consists of three fixtures, see figure 7.8. The DLO (blue

rope) should be routed through all three fixtures. In table 7.4 the initialization process can be seen. The three fixtures are added to the internal representation. Then for each fixture, two tasks are added, `GrabDLO` and `ClipIntoFixture`. As seen by the timestamps, this happens directly when the path planner is started.

Table 7.4: Pathplanner states during initialization

State	Time [s]
Adding fixture 1	0.00
Adding fixture 2	0.00
Adding fixture 3	0.00
Adding Task 1, *GrabDLO*	0.00
Adding Task 2, *ClipIntoFixture*	0.00
Adding Task 3, *GrabDLO*	0.00
Adding Task 4, *ClipIntoFixture*	0.00
Adding Task 5,*GrabDLO*	0.00
Adding Task 6, *ClipIntoFixture*	0.00

Figure 7.8 shows a sequence of images for clipping the DLO into the first fixture. The corresponding internal state can be seen in figure 7.9. In the internal state, the fixtures are represented as coordinate frames. It can be seen the internal state closely resembles the real workspace. There is no implemented feedback for the state of the grippers, the internal state does therefore not represent if the grippers are opened or closed.

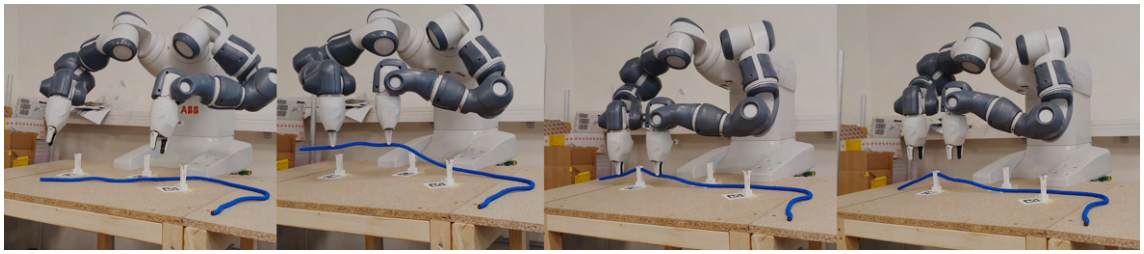


Figure 7.8: Clip into first fixture

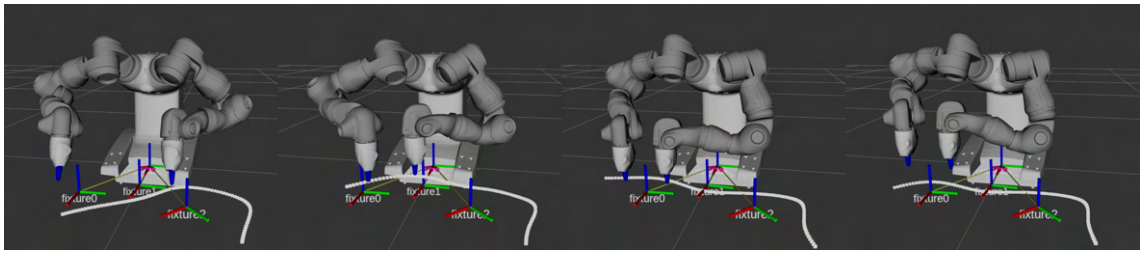


Figure 7.9: Clip into first fixture internal state

In table 7.5 the state in the path planner can be seen. First, a trajectory is generated from the `GrabDLO` task, it is then evaluated and checked for any problems. The

trajectory passes the evaluation and is sent to the controller and the path planner enters the reactive loop. In figure 7.8 the first two images show the robot picking up the DLO. After the DLO has been picked up a new trajectory is generated from the `ClipIntoFixture` task. As described earlier the trajectory is evaluated and then sent to the controller. In the last images in the figure, it can be seen that the DLO has successfully been clipped into the fixture. It took about 50 seconds to perform the two task for the first fixture, though speed is not an objective for this thesis.

Table 7.5: Pathplanner states for the first fixture

State	Time [s]
Generating trajectory parameters, task = 1	0.48
Evaluating trajectory parameters	0.48
Trajectory parameters pass and sent	0.49
Reactive loop	0.49
Task completed	33.89
Generating trajectory parameters, task = 2	33.98
Evaluating trajectory parameters	33.98
Trajectory parameters pass and sent	34.00
Reactive loop	34.00
Task completed	51.68

In figure 7.10 clipping of the second fixture can be seen and in table 7.6 the respective path planner states. As previously it first tries to generate trajectory parameters from the `GrabDLO` task. The trajectory parameter is then evaluated, through this time a problem is detected. One of the grip points along the DLO is too close to a fixture and would result in a collision is executed. The path planner sends trajectory parameters for holding the current pose while the stochastic solver is called. The stochastic solver takes about 10 seconds to find a solution. The trajectory parameters from the solution are evaluated and then sent to the robot. The three first images in figure 7.10 show the robot rerouting the DLO. In the new configuration, the `GrabDLO` task generates a new trajectory. This time the trajectory pass and the DLO is picked up, the fourth image. The `ClipIntoDLO` task is used to generate a new trajectory and the DLO is successfully clipped into the second fixture.

Figure 7.10: Clip into second fixture

Table 7.6: Pathplanner states for the second fixture

State	Time [s]
Generating trajectory parameters, task = 3	51.78
Evaluating trajectory parameters	51.78
Trajectory parameters fail;	51.79
- Grip points to close to fixture	
Hold position	51.79
Call stochastic solver	51.79
Evaluating trajectory parameters	61.89
Trajectory parameters pass and sent	61.91
Reactive loop	61.91
Task completed	114.61
Generating trajectory parameters, task = 3	114.72
Evaluating trajectory parameters	114.73
Trajectory parameters pass and sent	114.73
Reactive loop	114.73
Task completed	136.81
Generating trajectory parameters, task = 4	136.91
Evaluating trajectory parameters	136.91
Trajectory parameters pass and sent	136.92
Reactive loop	136.92
Task completed	152.91

The clipping of the third and final fixture can be seen in figure 7.11 and table 7.7. The execution is very similar to the second fixture, where the grip points are too close to the fixture. Through the task is successfully solved, the limitations of the system can be seen. The grip points that are used to reroute the DLO are close to the grip points that will later be used to clip the DLO. A human operator would have been able to use a nearby pick-up point to achieve the task without first rerouting the DLO. Through the objective was not to create the most efficient cable routing system.

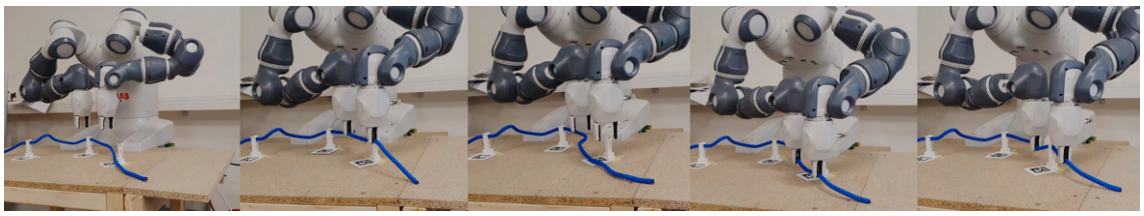


Figure 7.11: Clip into third fixture

Table 7.7: Pathplanner states for the third fixture

State	Time [s]
Generating trajectory parameters, task = 5	153.01
Evaluating trajectory parameters	153.01
Trajectory parameters fail;	153.02
- Grip points to close to fixture	
Hold position	153.02
Call stochastic solver	153.02
Evaluating trajectory parameters	164.65
Trajectory parameters pass and sent	164.67
Reactive loop	164.67
Task completed	203.87
Generating trajectory parameters, task = 5	203.97
Evaluating trajectory parameters	203.97
Trajectory parameters pass and sent	203.98
Reactive loop	203.98
Task completed	220.57
Generating trajectory parameters, task = 6	220.67
Evaluating trajectory parameters	220.68
Trajectory parameters pass and sent	220.69
Reactive loop	220.69
Task completed	233.47
All tasks completed	233.47

The cable routing test demonstrates that the system is capable of solving a real-world cable routing problem. In total it took 233 seconds to route the cable through all 3 fixtures. The system is not flawless, some of the flaws will be discussed in the next section.

7.5 Failure states and limitations

There are several limitations of the current system and scenarios that can not be solved. In figure 7.12 the `ClipIntoFixture` task fails. Several factors can cause this type of problem. As discussed in section 7.1, there is a probability that the pose estimation of the fixture is outside the necessary tolerances. Though this is not the main cause of the problem illustrated in figure 7.12. For the series of images, it can be observed that the DLO is curved between the grip points. From section 7.2, it was demonstrated that the estimation of the curves are slightly smoothed out. The result is that the physical length of the DLO between the gripper is longer than the estimated. From the second image in figure 7.12, it can be seen that the DLO is not stretched tight between the grippers and that the DLO retains some of that curvature. The physical clip point on the DLO is therefore not aligned with the absolute frame and the DLO is not successfully clipped into the fixture. The implemented tracking of the DLO is not accurate or responsive enough, to easily be

used as control feedback or to accurately validate if the DLO has been successfully clipped into the fixture. Therefore the path planner will assume that the DLO has been successfully clipped and continue with the next fixture.

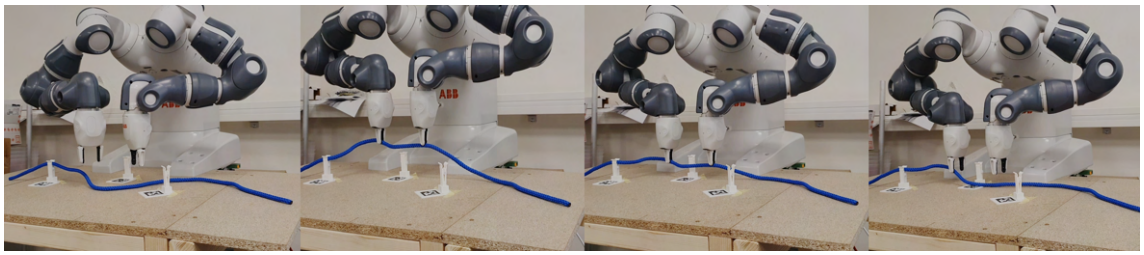


Figure 7.12: Fail to clip into second fixture

In figure 7.13, it can be seen that about a third of the DLO is occluded. From section 7.2, it was shown that the estimation becomes less accurate under occlusion. From the figure 7.13, it can also be speculated that with a different camera angle the occlusion could be even more severe. If the ends of the DLO are occluded, then the tracking is unable to estimate those parts. If the estimation is not accurate it can cause problems for the path planner. If the estimated DLO length does not match the actual length to the gripper points then the physical DLO constraints could be violated. There is also a possibility for a livelock situation. When the grippers move, different parts of the DLO may be occluded. Leading the estimation to change and the path planner to believe the DLO has moved, therefore retrying the same task again.

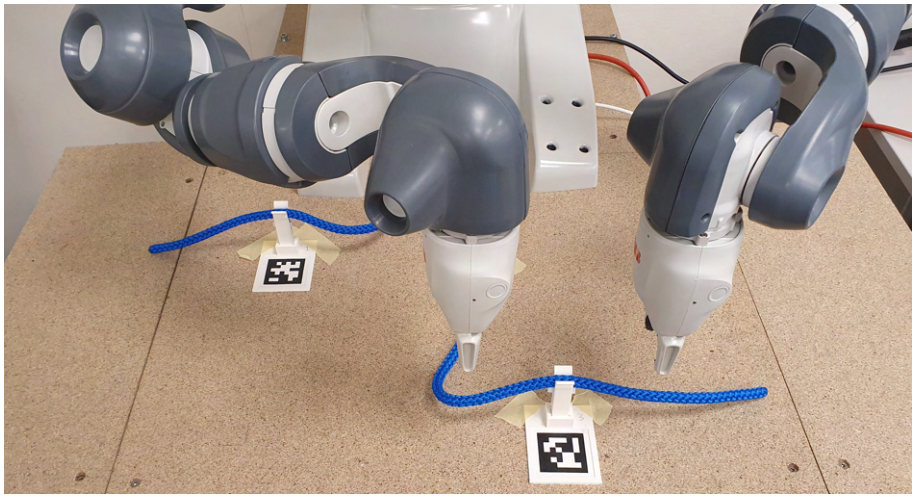


Figure 7.13: DLO occluded by robot arm

8

Conclusion

In this work, the manipulation of deformable linear objects is considered. A framework for solving cable routing problems with a dual-arm robot is proposed. The objective is to route a DLO from an arbitrary start, through several fixtures. The framework is built on three main parts, the controller, computer vision and the path planner.

The controller allows for both `individual` and `coordinated manipulation`. The inverse kinematics problem along with other feasibility objectives are solved with HQP. The `individual manipulation` is used to pick up the DLO. The DLO then acts as a physical constraint between the grippers. The `coordinated manipulation` allows for the arms to be controlled together, with `absolute motion` and `relative motion`. The `coordinated manipulation` is used for clipping the DLO into the fixture, where both arms collaborate to achieve the goal.

The computer vision system uses a single RGB-D camera. For pose estimation of the camera and the fixtures, the apriltag system is used. A color filter is used to separate the DLO from the background and then the depth information is used to generate a point cloud. From the point cloud, the SPR algorithm is applied to estimate the DLO.

The path planner generates trajectory parameters for the controller. A roadmap with tasks to solve the cable routing problem is constructed. The tasks are composed of motion primitives. The trajectory parameters are evaluated before they are sent to the controller. A biologically inspired stochastic optimization algorithm is used to solve any problem that is detected.

The work demonstrates successfully routing a rope through three fixtures with a dual-arm robot. The path planner demonstrates the ability to solve problems that occur during real-world testing. There is at the time of writing, no clear or standard method for how a problem of this type should be solved. Previous work has demonstrated a working solution with pre-recorded states. This thesis tries an alternative approach based on inverse differential kinematics and stochastic optimization, capable of solving cable routing with real-world experiments.

For future work, several things can be improved. Due to time constraints, the tracking algorithm was not prioritized. Implementing a more accurate and reactive tracking algorithm could improve both the existing solution and make it possible to add further capabilities. The gripper could be equipped with touch sensors and torque sensors, allowing for new control algorithms and precise manipulation with feedback. The touch sensor could also be used to validate that the DLO has been gripped securely and the torque sensors for controlling the tension in the DLO between the grippers. Multiple cameras could be composed to reduce critical occlusion

and increase accuracy.

Bibliography

[1] A. Jäger, C. Moll, and C. Lerch, "Analysis of the impact of robotic systems on employment in the european union - update," 12 2016.

[2] J. Sanchez, J. A. Corrales Ramon, B. C. BOUZGARROU, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey," *The International Journal of Robotics Research*, vol. 37, pp. 688 – 716, 06 2018.

[3] C. Lewis and A. Maciejewski, "Trajectory generation for cooperating robots," 09 1990, pp. 300 – 303.

[4] P. Chiacchio, S. Chiaverini, and B. Siciliano, "Direct and inverse kinematics for coordinated motion tasks of a two-manipulator system," *Journal of Dynamic Systems Measurement and Control-transactions of The Asme - J DYN SYST MEAS CONTR*, vol. 118, 12 1996.

[5] R. Jamisola, P. Kormushev, D. Caldwell, and F. Ibikunle, "Modular relative jacobian for dual-arms and the wrench transformation matrix," 07 2015.

[6] D. Almeida and Y. Karayiannidis, "Asymmetric dual-arm task execution using an extended relative jacobian," *CoRR*, vol. abs/1905.01248, 2019. [Online]. Available: http://arxiv.org/abs/1905.01248

[7] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, pp. 1006–1028, 05 2014.

[8] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[9] H. Chui and A. Rangarajan, "A feature registration framework using mixture models," 02 2000, pp. 190 – 197.

[10] A. Myronenko and X. Song, "Point set registration: Coherent point drift," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, pp. 2262–75, 12 2010.

[11] T. Tang and M. Tomizuka, "Track deformable objects from point clouds with structure preserved registration," *The International Journal of Robotics Research*, vol. 0, no. 0, 0.

[12] T. Tang, C. Wang, and M. Tomizuka, "A framework for manipulating deformable linear objects by coherent point drift," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3426–3433, 2018.

[13] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking deformable objects with point clouds," 05 2013, pp. 1130–1137.

[14] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," 2020.

[15] J. Zhu, B. Navarro, P. Fraisse, A. Crosnier, and A. Cherubini, "Dual-arm robotic manipulation of flexible cables," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 479–484.

[16] M. Rambow, T. Schauß, M. Buss, and S. Hirche, "Autonomous manipulation of deformable objects based on teleoperated demonstrations," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 2809–2814.

[17] S. Kudoh, T. Gomi, R. Katano, T. Tomizawa, and T. Suehiro, "In-air knotting of rope by a dual-arm multi-finger robot," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 6202–6207.

[18] M. Saha and P. Isto, "Manipulation planning for deformable linear objects," *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1141–1150, 2007.

[19] Y. Koga and J.-C. Latombe, "On multi-arm manipulation planning," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, pp. 945–952 vol.2.

[20] S. Duenser, R. Poranne, B. Thomaszewski, and S. Coros, "Robocut: hot-wire cutting with robot-controlled flexible rods," *ACM Transactions on Graphics*, vol. 39, 07 2020.

[21] M. Moll and L. Kavraki, "Path planning for deformable linear objects," *Robotics, IEEE Transactions on*, vol. 22, pp. 625 – 636, 09 2006.

[22] O. Roussel and M. Taïx, "Deformable Linear Object manipulation planning with contacts," in *Robot Manipulation: What has been achieved and what remains to be done? Full day workshop at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, United States, Sep. 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01159546

[23] J.-M. Ahuactzin, "Using genetic algorithms for robot motion planning," 05 1998.

[24] I. Bahaa, B. Kazem, A. Mahdi, and A. Talib, "Motion planning for a robot arm by using genetic algorithm," *Jordan Journal of Mechanical and Industrial Engineering*, vol. 3, 10 0002.

[25] E. Solteiro Pires and J. Tenreiro Machado, "A ga perspective of the energy requirements for manipulators maneuvering in a workspace with obstacles," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 2, 2000, pp. 1110–1116 vol.2.

[26] G. Nagib and W. Gharieb, "Path planning for a mobile robot using genetic algorithms," 10 2004, pp. 185– 189.

[27] S. Benhlima, L. Chaymaa, and A. Bekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, 03 2018.

[28] A. Karami and M. Hasanzadeh, "An adaptive genetic algorithm for robot motion planning in 2d complex environments," *Computers & Electrical Engineering*, vol. 43, 01 2015.

[29] Y. Hu and S. Yang, "A knowledge based genetic algorithm for path planning of a mobile robot," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, 2004, pp. 4350–4355 Vol.5.

[30] S. W. R. H. L. P. F. H. M. R. S. Ed. and D. H. or Corr. M., "Ii. on quaternions; or on a new system of imaginaries in algebra," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 25, no. 163, pp. 10–13, 1844. [Online]. Available: https://doi.org/10.1080/14786444408644923

[31] S. Bruno, S. Lornzo, V. Luigi, and G. Oriolo, *Robotics, Modelling, Planning and Control*. London: Springer, 2009.

[32] J. C. A. Barata and M. S. Hussein, "The moore–penrose pseudoinverse: A tutorial review of the theory," *Brazilian Journal of Physics*, vol. 42, no. 1-2, p. 146–165, Dec 2011. [Online]. Available: http://dx.doi.org/10.1007/s13538-011-0052-z

[33] M. Wahde, *Biologically Inspired Optimization Methods*. Boston: WIT press, 2008.

[34] G. H. Joblove and D. Greenberg, "Color spaces for computer graphics," vol. 12, no. 3, p. 20–25, Aug. 1978. [Online]. Available: https://doi.org/10.1145/965139.807362

[35] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2003.

[36] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2010.

[37] E. Olson, "Apriltag: A robust and flexible visual fiducial system," 06 2011, pp. 3400 – 3407.

[38] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[39] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[40] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, April 2013, pp. 1–6.

[41] K. Shoemake, "Animating rotation with quaternion curves," *SIGGRAPH Comput. Graph.*, vol. 19, pp. 245–254, Jul. 1985.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY