

Model predictive control of an assistive robotic arm for shared autonomy implementation

by

Anton Kim

Submitted to the Department of Robotics and Mechatronics
in partial fulfillment of the requirements for the degree of

Master of Science in Robotics

at the

NAZARBAYEV UNIVERSITY

March 2021

© Nazarbayev University 2021. All rights reserved.

Author
Department of Robotics and Mechatronics
March 19, 2021

Certified by
Almas Shintemirov
Associate Professor
Thesis Supervisor

Accepted by
Vassilios D. Tourassis
Dean, School of Engineering and Digital Sciences

Model predictive control of an assistive robotic arm for shared autonomy implementation

by

Anton Kim

Submitted to the Department of Robotics and Mechatronics
on March 19, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Robotics

Abstract

This work describes MPC implementation for 6-DOF robotic arm which can autonomously reach a target pose generated by a grasping pose estimation algorithm and allows the user to send Cartesian velocity commands to the robot via 3-axis joystick thanks to the system model based on Jacobian. The developed controller makes use of variable weighting matrices to make movement in different control modes more smooth and realistic. It also includes a visual constraint to minimize occlusion of target object by the manipulator. The next step of this project is to implement a shared autonomy scheme based on the developed system.

Thesis Supervisor: Almas Shintemirov
Title: Associate Professor

Acknowledgments

I would like to thank my supervisor Prof. Almas Shintemirov for his support, help, and advices. This thesis would have been much difficult without him being present.

I would like to thank my labmate Sergey Soltan, who provided me great deal of assistance in writing code and adopting processing algorithms for point clouds - the fields I am not so proficient in myself.

I would like to thank all my labmates at ALARIS Lab who I spend most of my time with. We had a lot of thoughtful conversations, knowledge sharing, fun time and moral support.

Special thanks to my dear friend Aigerim Nurbayeva and my family who I could share all my thoughts and worries with, who give me strength and energy to go on, and who keep believing in me no less than myself.

Contents

1	Introduction	11
2	Related work	13
2.1	Shared autonomy	13
2.1.1	Arbitration by blending	13
2.1.2	Markov processes	15
2.1.3	Visualization	16
2.2	Model Predictive Control	16
2.3	Grasping pose estimation	18
3	Methodology	21
3.1	Framework	21
3.1.1	ROS	21
3.1.2	ACADO Toolkit	22
3.2	Quaternions	23
3.3	Problem definition	24
3.3.1	System model	24
3.3.2	MPC formulation	25
4	Implementation	27
4.1	Hardware setup	27
4.2	System calibration	27
4.3	Grasping pose estimation	28

4.4	Joystick control	29
4.4.1	Control modes	29
4.4.2	Intuitive frame	31
4.5	Real-Time NMPC solver	31
4.5.1	NMPC constraints	32
4.5.2	Choice of weights for weighting matrices	33
5	Results	37
5.1	Autonomous reaching	37
5.2	Teleoperated reaching	38
5.3	Discussion and future work	42
6	Conclusion	45
A	Jacobian generation for Kinova Jaco v2 robotic manipulator	47
B	NMPC formulation for ACADO Code Generation	51

List of Figures

1-1	Robot setup used in this work	12
3-1	The simple program in the ROS (<i>www.ros.org</i>)	21
4-1	System calibration	28
4-2	Visualization of estimated grasp	29
4-3	Joystick control modes	30
4-4	Visual occlusion avoidance	33
5-1	End-effector's trajectory during autonomous reaching	38
5-2	End-effector error during autonomous object reaching	39
5-3	Joint positions and joint velocities during the autonomous reaching .	39
5-4	End-effector velocity during autonomous grasping	40
5-5	End-effector trajectory during teleoperated reaching	40
5-6	End-effector reference and actual velocity during teleoperated grasping	41
5-7	Joint positions and joint velocities during the teleoperated reaching .	41
5-8	Feedback step execution time	42

Chapter 1

Introduction

Shared autonomy or human-machine cooperation can be viewed as a robot control scheme when both the user and the robot have some or significant degree of control over a system or over a task[10]. Such systems can be useful for people with lower-body disability and/or partial upper-body disability. Such people are capable of performing some tasks on their own, but other tasks may become unavailable to them, and a robotic system can compensate some of the lost capabilities. A properly designed system could potentially help in performing activities of daily living.

Research tells that it is important to give the user a sufficient degree of control over a system[20]. Users should be given as much control as possible, while it is important for the system to remain intuitive and comfortable to use. The system should not be too simple as it will probably constraint allowed task space for the user, but it also should not be too complex as it might become too tiring for the user. From a study by Dragan and Srinivasa[6] it is possible to learn that the assistance level in shared autonomy scheme should vary depending on how difficult the task is for the user.

For this work, an attempt was made to implement a control scheme with an assistive 6-DOF robot manipulator using a setup from a previous work to develop a shared autonomy in the future[27](see Figure1-1). A joystick was used to receive user commands and an external RGB-D camera for machine vision.

The control algorithm for this work was chosen to be model predictive control (MPC). MPC is an advanced control method that is widely used in different areas.

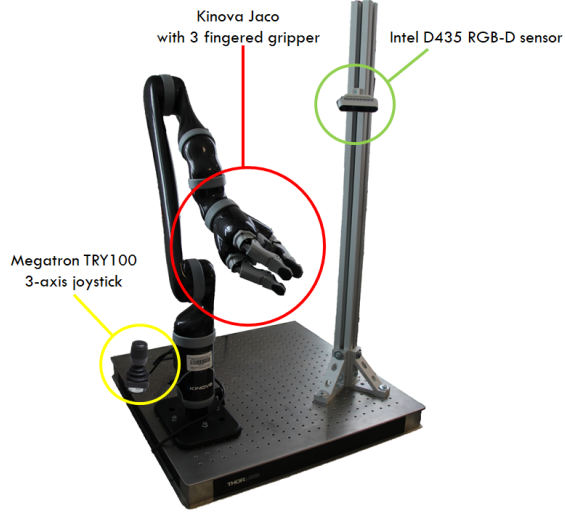


Figure 1-1: Robot setup used in this work

It differs from more classical Linear Quadratic Regulators (LQR) in that it optimizes problem after each time sample over a finite time horizon, taking into consideration future states and control actions.

Approach used in this work is robotic manipulator control in velocity space using MPC controller. Novelty of this approach is the use of analytical Jacobian of the manipulator as system model. The purpose of this approach is to control the manipulator in joint velocity space and to be able to integrate joystick for shared control of Cartesian velocity of the manipulator's end-effector.

This work consists of the following chapters: Introduction, Related work, Methodology, Implementation, Results and Conclusion

Chapter 2

Related work

2.1 Shared autonomy

Current robots are not intelligent enough to work fully autonomously. This is especially true for complex robots like manipulators or other multiple DOF systems. They are usually preprogrammed for specific tasks and work without people in the direct vicinity (e.g. car factory manipulators and manufacturing set-ups). However robots can be used in multiple domains, where it is important to have a safe robot, such as domestic use, assistive systems, medical applications and so on.

For an assistive system it is important that system can do its primary task - assist a user in performing a task. At the same time, user should have a sufficient level of control over the system[20]. There are multiple works that implement some type of assistive or shared autonomy schemes. In general shared autonomy problem consists of two steps: 1) user intent prediction, and 2) help in intent prediction[6].

2.1.1 Arbitration by blending

Probably one of the most early works in shared autonomy was done by Dragan and Srinivasa[6] where they reviewed different arbitration approaches and introduced their policy blending algorithm in the context of an assistive manipulator. The algorithm consists of two step: goal prediction and trajectory prediction. Goal prediction is

modeled as a maximum value of a probability function, which is based on cost function of potential goal. Trajectory prediction can be done via any trajectory generation algorithm, in the case of my thesis work it can be MPC. Arbitration is performed based on the systems confidence in choosing the right goal.

Geravand et.al.[11] developed a shared autonomy scheme for a mobility assistance robot (MAR). The system has a low-level and high-level control: kinematic model and a admittance control model. Kinematic model controls the motion on motors, while admittance control generates the control signals. The shared control scheme manipulates variables inside the admittance control. There are three decision making functions based on sensorial, physical and cognitive feedback. They have form of a reward function and each control a variable in admittance control model. The test results with 35 elderly people indicate that this shared autonomy scheme greatly reduced number of collisions with obstacles thanks to limiting approach velocity near obstacles.

Gopinath and Argall[12, 13] published two consecutive works on shared autonomy optimization and intent disambiguation. In their first work[12] they proposed to view shared autonomy as a form of optimal control problem. The optimization is performed by the end-user via verbal commands and the optimized function is a linear blending function described by three parameters. Their next work[13] focused on intent disambiguation. Each potential goal is assigned a probability distribution function (PDF). The disambiguation algorithm performs operations with the PDFs to calculate parameters for disambiguation metrics. The disambiguation metrics are there to decide which control mode to enter and which control dimensions to control, based on confidence in intent estimation.

In the work by Jin and Pagilla[17] they predict operators intent in reaching a subgoal based on two prediction functions: action based prediction and transition based prediction. Action based prediction estimates probability of a subgoal being the operator’s target based on angle between the action required to reach the subgoal and the operator’s action. The transition based prediction estimated probability of a subgoal being the operator’s goal by taking into account repetitive form of the

process.

Li[22] use a simple blending function to give more control to either robot or human. The arbitration coefficient is estimated based on two uncertainty models: intent uncertainty and autonomy uncertainty. The probability is modeled as Gaussian probability function and confidence is estimated with Gauss error function over probability distribution function for target.

In summary shared autonomy schemas share a common feature in that they often use a linear blending function to arbitrate between giving control to a human or to a system. The blending function has a common form as in (2.1)

$$X = (1 - \alpha)X_{robot} + \alpha X_{human} \quad (2.1)$$

What researchers are trying to do is to develop smart algorithms to estimate α based on different probabilistic models for human operator's intent and the ability of robot to guess the intent.

2.1.2 Markov processes

There are also shared autonomy schemas that are built using Markov processes [16, 23, 4] which are stochastic processes, where value of the next state depends solely on the value of the current state and the control action. They use different variations such as partially observable Markov decision processes (POMDP) and hidden Markov models (HMM).

A typical Markov decision process is defined as follows [23]:

$$MDP = \{S, A, T(s^i, a_k, s^j), R(s^i, a_k)\} \rightarrow \pi(s^i) \quad (2.2)$$

where $s^i \in S$, $a_k \in A$, $R(s^i, a_k)$ is the set of state-dependent rewards for performing an action a_k at a state s^i , and $T(s^i, a_k, s^j) = p(s^j | s^i, a_k)$ is the set of all transition probabilities.

Javdani et.al.[16] developed an assistive control scheme based on hindsight opti-

mization and modeled as a partially observable Markov decision process. They utilize a cascade of POMDP and MDP. First generates guesses about user’s goal based on a set of beliefs, while second uses the guesses as known targets and model robot actions.

2.1.3 Visualization

Another interesting idea for control sharing is an assistive control. It can be a pure visual assistance[4, 25] or autonomous passive assistance[1, 2].

Quintero et.al.[25] developed a visual only interface called VIBI. The interface is a 2D image-based interface where user selects a desired target and a pose robot should approach the target with. At the same time the user can take control over the autonomous task execution and perform a teleoperation.

Brooks and Szafir[4] supplemented a shared autonomy with augmented reality (AR) to increase user acceptance of the shared autonomy. AR shows user what is system’s intended target and how it intends to move to the target.

Farraj et.al.[1, 2] developed a shared autonomy scheme for remote telemanipulation. There are two manipulators: one teleoperated by a human operator and the second is autonomous with a camera mounted in the end-effector.

2.2 Model Predictive Control

Model Predictive Control (MPC) has a long history and found its application in different areas[3, 7, 19]. MPC uses system model to find optimal control values for a given time window. The main feature of MPC is moving horizon optimization - the optimization is done over a relatively small time (horizon). However, unlike in more standard optimization techniques like LQR, only the first step of generated control sequence is implemented. Then optimization routine is repeated.

With the advancements in CPUs, it has become possible to solve MPC problems for complex systems such as 6-DOF manipulators in real-time. There are several works that used MPC for different robotic systems such as drones(MAV)[9, 18], manipulators[28, 36] and mobile robots[26].

Zube[36] propose control of a 7 DOF redundant manipulator and a 10 DOF mobile manipulator with NMPC controller. The system model is based on the forward kinematics of the manipulator which are derived using DH parameters. The cost function minimizes joint displacements and joint velocities as well as trajectory following error. Obstacle avoidance is performed by keeping the distance between test points on the robot and the obstacles greater than some minimum allowed distance. The author used sampling time $T_s = 0.1s$ and $N = 10$ prediction steps to have prediction horizon of 1s and report that control loop execution time is about 70ms on 2.8GHz Intel Core i7 processor. It was shown that introduction of nonlinear constraints into NMPC formulation can increase computation time by more than two times.

Kamel[18] use NMPC for trajectory tracking of micro aerial vehicle (MAV) and compare its performance with LMPC. The cost function in this work is a simple squared distance between desired trajectory and the performed trajectory. The authors implemented controllers using CVX-GEN and ACADO Toolkit for LMPC and NMPC respectively and were running the controllers on 3.1GHz Intel i7 processor. The sampling time $T_s = 0.1s$ and $N = 20$ prediction steps were chosen which is equivalent to prediction horizon of 2 seconds. The tests performed in this paper show that NMPC performs better for hovering task, position step response and trajectory tracking. For this specific case authors recorded that control loop for NMPC runs much faster than for LMPC.

Another MPC implementation for drones[9] was presented, which takes into consideration camera parameters into system model to track target in camera frame and introduce perception component. They implement controller using ACADO Toolkit with sampling time $T_s = 0.1s$ and $N = 20$ prediction steps. The hardware used for experiments was ARM processor with 2.26GHz clock frequency. The task in this work was to track a target and the developed controller has shown good results.

Another work[28] implemented MPC controller to perform teleoperation of a 6 DOF manipulator. Authors used robot forward kinematics as system model and Jacobian matrix for singularity avoidance. Multiple test points on the robot body were used to ensure that robot would not collide with environment and would avoid ob-

stacks. The MPC controller was implemented using ACADO Toolkit with sampling time $T_s = 0.1s$ and $N = 10$ prediction steps to obtain prediction horizon of 1 second. The controller was tested on machine with 2.3GHz Intel Core i7 processor with control loop execution time less than 3 ms. The experimental results show that MPC controller can achieve a good real-time trajectory tracking.

More recently NMPC was also implemented for mobile robot with Ackermann steering[26] with classical kinematic single-track model. In this work authors used variable weighting matrices as opposed to previous works where weighting matrices were fixed. The controller was implemented using ACADO Toolkit with sampling time $T_s = 0.025s$ and $N = 80$ prediction steps which results in prediction horizon of 2s. The controller was tested on Raspberry PI 3 with ARM Cortex-A53 processor and has shown a good path tracking on a scaled two lane road.

It can be seen that MPC has a great potential as it was used in different successful projects. For MPC to give the best performance and precision it is important to define a proper system model and correctly prioritize optimization variables by assigning convenient scaling weights in the cost function. It is also important to use variable weighting matrices, to be able to integrate into controller multiple behaviours.

2.3 Grasping pose estimation

To generate a set of potential targets it is possible to use YOLO like object detection and recognition algorithms together position extraction from depth frame[27]. However, with this method it is only possible to get a position of the potential object and not the orientation. To get a complete pose of a potential target it is better to use pose estimation algorithms. There are works which focused specifically on generating a set of possible gripping poses for an object given a point cloud representation. Some of them train neural network models[30, 35, 14], while others use geometrical approaches to process the incoming pointclouds[34, 15].

The general procedure in these approaches is to first preprocess point cloud and remove background and ground plane. Then clusterization/voxelization will be ap-

plied to detect individual objects. When the point clouds for individual objects are extracted, the grasping pose estimation algorithm is applied.

Pose estimation includes two main steps. First, find main axis of grasp - human intuitively grasps an object perpendicularly to its longest axis. Second, determine its center of mass and estimate a grasping pose. In [34] they do pose estimation by first estimating finger contact points and then based on that knowledge calculate necessary grasp pose. In [30] multiple grasp poses generated using sampling and then CNN ranks them individually.

Chapter 3

Methodology

3.1 Framework

3.1.1 ROS

Most of the implementations during this thesis are done in Robot Operating System (ROS). ROS is an actively developing middleware, has the community support and is an open-source. It is a popular framework among robotics developers as it has many useful tools and libraries that make developing a robot program easier.

The basic building block of ROS is a node (Figure 3-1), a program which does a specific routine and can communicate with other nodes through topics and services.

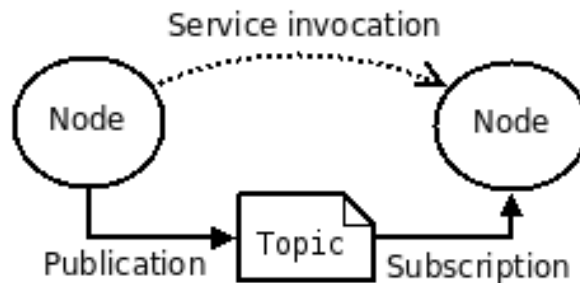


Figure 3-1: The simple program in the ROS (www.ros.org)

Many hardware developers create ROS oriented drivers for their devices that provide data stream and can be controlled via ROS interface. These include devices I work with as well, which are Kinova Gen2 Jaco, Intel RealSense D435 RGB-D camera

and a 3-axis joystick.

Finally, ROS has a lot of standard definitions for different message types, which are used across different packages. Because of this standardized structure, users don't have to adapt to different SDKs and it becomes easier to develop your own code inside ROS environment.

3.1.2 ACADO Toolkit

One of the key components of this thesis is Automatic Control and Dynamic Optimization Toolkit (ACADO Toolkit) [31]. It is a collection of algorithms and implementations of integrators for optimal control, all wrapped into user friendly syntax, which allows users to focus on problem definitions instead of syntax. ACADO Toolbox was used extensively during this thesis work to generate different versions of a solver for nonlinear model predictive control.

ACADO accepts MPC formulations of the following form:

$$\begin{aligned}
& \min_{\substack{x_0, \dots, x_N \\ u_0, \dots, u_{N-1}}} \sum_{k=0}^{N-1} \|h(x_k, u_k) - \hat{y}_k\|_{W_k}^2 + \|h_N(x_N) - \hat{y}_N\|_{W_N}^2 \\
& \text{subject to } x_0 = \hat{x}_0 \\
& x_{k+1} = F(x_k, u_k, z_k), k = 0, \dots, N-1 \\
& x_k^{lb} \leq x_k \leq x_k^{ub}, k = 0, \dots, N \\
& u_k^{lb} \leq u_k \leq u_k^{ub}, k = 0, \dots, N-1 \\
& r_k^{lb} \leq r_k(x_k, u_k) \leq r_k^{ub}, k = 0, \dots, N-1 \\
& r_N^{lb} \leq r_N(x_n) \leq r_N^{ub}
\end{aligned} \tag{3.1}$$

where the cost function is a squared norm of difference between the desired values and the current values. The user defines reference functions $h(x_k, u_k)$ and $h_N(x_N)$, continuous system model $F(x, u, z)$ which discretized by ACADO into $F(x_k, u_k, z_k)$, constraint functions $r_k(x_k, u_k)$ and $r_N(x_n)$, tunes weighting matrices W_k and W_N , and chooses parameters such as horizon length, sampling time, and upper and lower bounds on states, control variables and constraints.

With the problem formulated in this way ACADO can generate C-code which contains functions and variables necessary to solve NMPC. Specifically, it implements Gauss-Newton RT algorithm, ODE, derivation and integration routines. Acado Manual contains detailed description of the generated files and advanced code generation options .

3.2 Quaternions

Quaternion is a representation of rotation between frames as well as a rotation operator of the following form

$$\mathbf{q} = [\cos \phi \quad \mathbf{i} * r_x * \sin \phi \quad \mathbf{j} * r_y * \sin \phi \quad \mathbf{k} * r_z * \sin \phi]^T \quad (3.2)$$

where vector $\mathbf{r} = [r_x, r_y, r_z]$ represents rotation axis and ϕ rotation angle about \mathbf{r} . Usually, a quaternion is represented in form of $\mathbf{q} = [w, x, y, z]$, where w is the real part and $[x, y, z]$ is the imaginary part.

Suppose there are two frames which have orientation expressed as quaternions $\mathbf{q}_1 = [w_1, x_1, y_1, z_1]$ and $\mathbf{q}_2 = [w_2, x_2, y_2, z_2]$. Then quaternion error can be expressed with the following equation:

$$\delta \mathbf{q} = w_1 \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} - w_2 \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \quad (3.3)$$

which, after the required operations, can be rewritten in the following form:

$$\delta \mathbf{q} = \begin{bmatrix} w_1 * x_2 - w_2 * x_1 + z_1 * y_2 - y_1 * z_2 \\ w_1 * y_2 - w_2 * y_1 - z_1 * x_2 + x_1 * z_2 \\ w_1 * z_2 - w_2 * z_1 + y_1 * x_2 - x_1 * y_2 \end{bmatrix} \quad (3.4)$$

It was shown by Yuan in [33] that $\delta \mathbf{q} = 0$ in Equation 3.4 is necessary and sufficient condition for two coordinate frames to coincide.

Quaternion rate of change can be expressed in different ways [8][9][24][29] which depends on the choice of reference frame (body frame or fixed frame) and handedness of quaternion (right-handed or left-handed). For this particular case the quaternion rate of change can be expressed in the following form:

$$\dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{bmatrix} \mathbf{q} \quad (3.5)$$

where $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]$ is the angular velocity of body expressed in global frame, and \mathbf{q} is the current orientation of body frame in global frame. It can be observed that the matrix is a skew-symmetric operator.

3.3 Problem definition

3.3.1 System model

The system model is defined by the following state vector:

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{\theta} & \mathbf{p}_{ee} \end{bmatrix}^T \in \mathbf{R}^{13} \quad (3.6)$$

where $\boldsymbol{\theta} = [\theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6]^T$ is the joint positions of the manipulator, and $\mathbf{p}_{ee} = [\mathbf{r}_{ee}, \mathbf{q}_{ee}] = [x_{ee}, y_{ee}, z_{ee}, q_{0ee}, q_{1ee}, q_{2ee}, q_{3ee}, q_{4ee}]$ is the position and orientation of the end-effector of the manipulator. The system control variables are joint velocities:

$$\mathbf{U} = \dot{\boldsymbol{\theta}} \in \mathbf{R}^6 \quad (3.7)$$

The corresponding system model has the following form:

$$\mathbf{X} = F(x(t), u(t), z(t)) = \begin{bmatrix} \dot{\boldsymbol{\theta}}(t) \\ \mathbf{v}_{ee}(t) \\ \dot{\mathbf{q}}_{ee}(t) \end{bmatrix} \in \mathbf{R}^{13} \quad (3.8)$$

where rate of change of quaternion $\dot{\mathbf{q}}(t)$ is calculated by Equation 3.5.

The vector $[\mathbf{v}_{ee}, \boldsymbol{\omega}_{ee}] = [v_{xee}, v_{yee}, v_{zee}, \omega_{xee}, \omega_{yee}, \omega_{zee}]$ is estimated based on the manipulator's Jacobian:

$$\begin{bmatrix} \mathbf{v}_{ee} \\ \boldsymbol{\omega}_{ee} \end{bmatrix} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \quad (3.9)$$

Jacobian is derived analytically from forward kinematics of the manipulator, described in the Kinova User Guide[21]. Refer to Matlab code in Appendix A for Jacobian derivation.

3.3.2 MPC formulation

The following form of MPC was formulated using ACADO Toolkit:

$$\begin{aligned} \min_{\substack{x_0, \dots, x_N \\ u_0, \dots, u_{N-1}}} & \sum_{k=0}^{N-1} \|h(x_k, u_k) - \hat{y}_k\|_{W_k}^2 + \|h_N(x_N) - \hat{y}_N\|_{W_N}^2 \\ \text{subject to} & \quad x_0 = \hat{x}_0 \\ & \quad x_{k+1} = F(x_k, u_k, z_k), \text{ for } k = 0, \dots, N-1 \\ & \quad x_k^{lb} \leq x_k \leq x_k^{ub}, \text{ for } k = 0, \dots, N \\ & \quad u_k^{lb} \leq u_k \leq u_k^{ub}, \text{ for } k = 0, \dots, N-1 \\ & \quad r_k^{lb} \leq r_k(x_k, u_k) \leq r_k^{ub}, \text{ for } k = 0, \dots, N-1 \\ & \quad r_N^{lb} \leq r_N(x_n) \leq r_N^{ub} \end{aligned} \quad (3.10)$$

with number of discretization steps $N = 10$ and sampling time $T_s = 0.1s$. \hat{y}_k and \hat{y}_N denote real-time dynamic reference and $W_k \in \mathbf{R}^{18 \times 18}$ and $W_N \in \mathbf{R}^{6 \times 6}$ are diagonal variable weighting matrices. $r_k(k_k, u_k)$ and $r_N(x_n)$ are the path and point constraints.

Superscripts lb and ub denote lower and upper bounds. $F(x_k, u_k, z_k)$ is the discretized system model described in the Equation 3.8.

The reference functions $h(x_k, u_k)$ and $h_N(x_N)$ are defined in the following way:

$$h(x_k, u_k) = \begin{bmatrix} \mathbf{r}_{ee} \\ \delta \mathbf{q}_{ee} \\ \mathbf{v}_{ee} \\ \boldsymbol{\omega}_{ee} \\ \dot{\boldsymbol{\theta}} \end{bmatrix} \in \mathbf{R}^{18} \quad (3.11)$$

$$h_N(x_N) = \begin{bmatrix} \mathbf{r}_{ee} \\ \delta \mathbf{q}_{ee} \end{bmatrix} \in \mathbf{R}^6 \quad (3.12)$$

with $\delta \mathbf{q}_{ee}$ being the quaternion error defined in the Equation 3.4.

Chapter 4

Implementation

4.1 Hardware setup

I used Kinova Jaco v2 robotic manipulator. It is a lightweight assistive manipulator which has six degrees of freedom and three finger end-effector. For robot vision Intel RealSense D435 RGB-D camera was used. For user input I used MEGATRON 3-axis joystick with 2 buttons. All of the three devices are connected to PC via USB ports. The PC runs on 3.30GHz Intel Core i9-7900X CPU and holds 32GB RAM.

4.2 System calibration

The calibration procedure is estimating pose of the camera frame relative to the robot frame. For this it is needed to gather Cartesian coordinates of four points in camera frame and in robot frame. Then run a simple algorithm to estimate the relative transformation from camera frame to robot frame.

For gathering the points there were two different approaches. The first was used in the initial stages of the thesis and is less precise than the second. Here only the second will be described.

With the robot and camera drivers on, a script will be launched that will listen to topics `/clicked_point` and end-effector tool pose. Topic `/clicked_point` is RViz topic which gives Cartesian coordinates of a clicked point. A user will click at a red reference

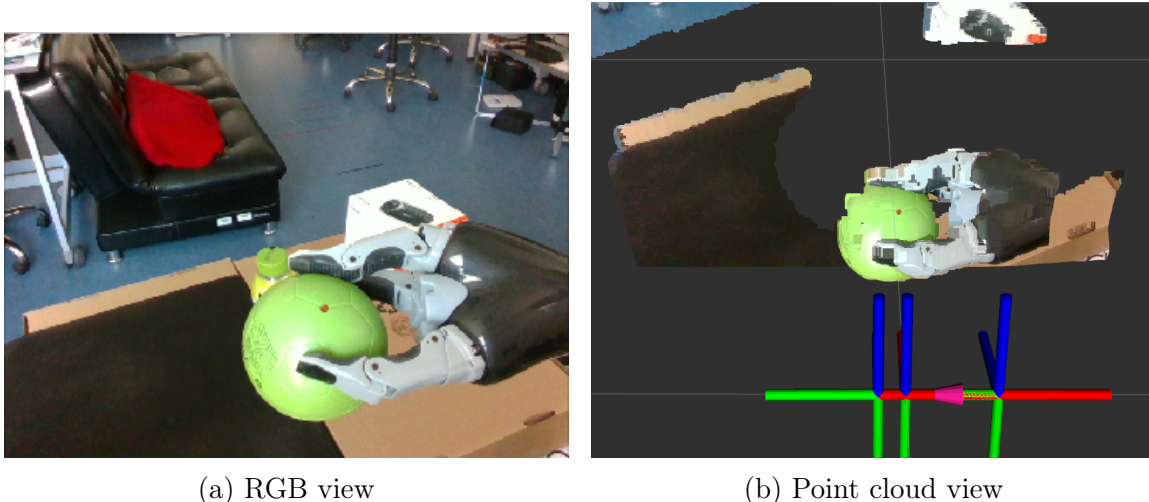


Figure 4-1: System calibration

point in point cloud in RViz as shown in Figure 4-1). The script will gather five points from `/clicked_point` and compute their average, then it will read end-effector pose and extract end-effector position. Then both averaged value and end-effector position will be written into a file. Finally, the user will move end effector to a new place and the procedure will be repeated three more times. The points gathered should lie in one plane with except of one, which should be outside of the plane.

When four points are gathered, we follow a procedure described in [27] to find transformations from an arbitrary frame to the robot and to the camera. Knowing these two transforms it becomes straightforward to calculate transformation from camera to robot and its inverse.

4.3 Grasping pose estimation

For reference generation approach presented in [34] was adopted. It generates reference end-effector position and orientation and optionally visualizes fingers contact points as shown in Figure 4-2. The reference pose is estimated in the camera frame, so it is transformed into the robot frame using transform found during system calibration.

Before transforming the generated reference pose into the robot frame it is nec-

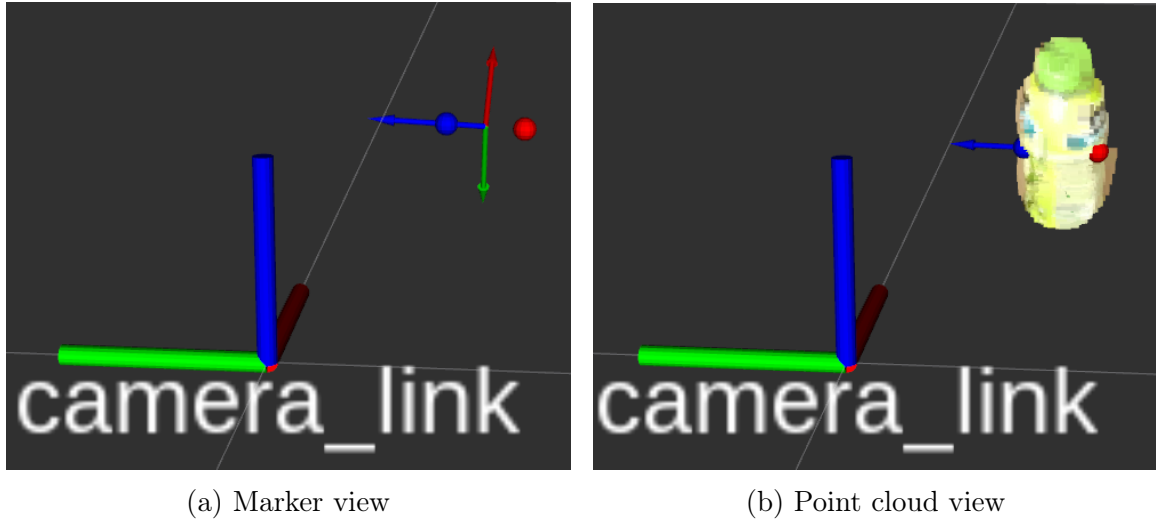


Figure 4-2: Visualization of estimated grasp

essary to rotate it around Z axis by 180 degrees to make it a more natural grip for manipulator. This way X and Y axes of the reference pose and of the end-effector will have similar directions, and therefore it will be easier to reach the target.

Also, there are several frames attached to the camera, for example optical frame, depth frame, body frame and so on. The reference pose is generated in the *camera_color_optical_frame* and first should be transformed into main camera frame *camera_link* before transforming it into the robot base frame. This is done to ensure that ROS can handle the constructed transformation tree and will be able to visualize it in RViz.

4.4 Joystick control

4.4.1 Control modes

There are three control modes implemented for robot teleoperation: position control, orientation control and finger control (see Figure 4-3). The robot behaviour in these modes is almost identical to that observed when controlling it with factory joystick. Switching between modes is done via buttons on the joystick. Right button will switch between mode 1 and mode 2. Left button switches between modes 1 and

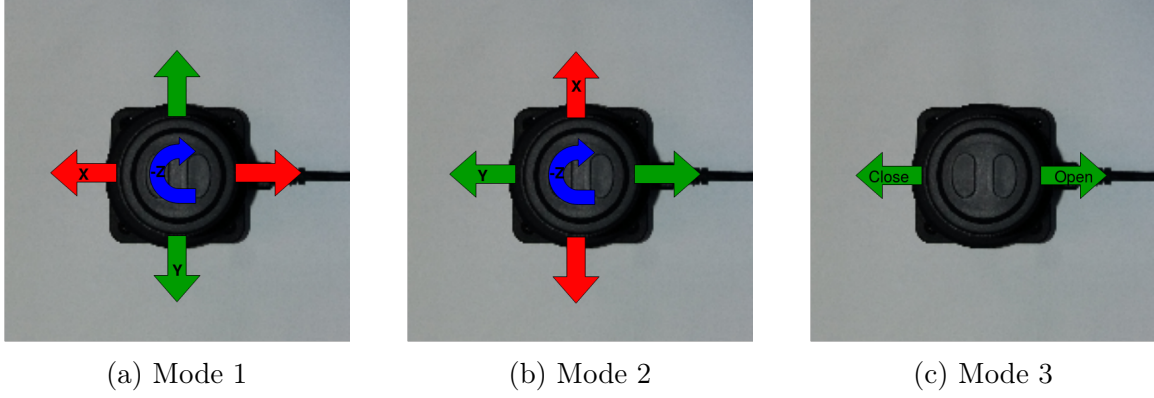


Figure 4-3: Joystick control modes

3. Simultaneously pressing both buttons will move the manipulator to the home configuration. The manipulator does not accept any other commands while moving to the home configuration.

In the first mode, user controls the end-effector's linear velocity in the Cartesian space. The commands are sent in the form of linear velocity vector and NMPC solver generates joint velocities. The maximum linear velocity hardcoded into the solver is 0.4m/s and user command is scaled not to exceed 0.2m/s. Forward push will move end-effector forward and in negative Y direction relative to base frame. Left push moves end-effector to the left and in the positive X direction in the base frame. Finally, counter-clockwise rotation will move end-effector up in the positive direction of the base frame.

In the second mode, user controls end-effector's angular velocity in the Cartesian space. As with linear velocity, user sends angular velocity vector and NMPC solves for joint velocities. The maximum angular velocity of end-effector is constrained at 0.8rad/s by solver and user command is scaled to be within 0.4rad/s. Forward push rolls end-effector in positive direction of its X-axis. Left push pitches end-effector around its Y-axis. Rotation with yaw end-effector around its Z-axis.

In mode 3, user controls speed of opening or closing the fingers. Tilting to the left will start closing the fingers and tilting to the right will start opening the fingers.

4.4.2 Intuitive frame

The solver finds solutions in the base frame of the manipulator. However, controlling the end-effector in base frame is not intuitive and Campeau-Lecours et.al. [5] suggest to use an additional virtual frame $EE2 = [x_2, y_2, z_2]$.

Assuming that the base frame is $O = [x_0, y_0, z_0]$ and end-effector frame is $EE = [x_1, y_1, z_1]$, we define z_2 axis to be the same as z_1 , i.e. pointing out of the end-effector. The x_2 axis lays in the horizontal plane and is perpendicular to the both z_2 and z_0 :

$$\mathbf{e}_{x2} = \frac{\mathbf{e}_{z0} \times \mathbf{e}_{z2}}{\|\mathbf{e}_{z0} \times \mathbf{e}_{z2}\|} \quad (4.1)$$

To keep x_2 defined at all times, it is necessary to redefine it as follows:

$$\text{if } \alpha < \alpha_{min} \text{ then } \mathbf{e}_{x2} = \mathbf{e}_{x0} \quad (4.2)$$

where α_{min} is a threshold angle and

$$\alpha = \cos^{-1}(\mathbf{e}_{z0} \cdot \mathbf{e}_{z2}) \quad (4.3)$$

4.5 Real-Time NMPC solver

ACADO Code Generation tool was used to encode kinematic model of the robot, system dynamics and constraints and generate NMPC solver. Full program for code generation is presented in Appendix B. One thing worth noting is the number of integration steps defined in line 219 of the code, which will be discussed later.

As it was discussed in the previous chapter it is important to use variable weighting matrices to be able to reproduce different motion behaviours. For this thesis many different weighting matrix coefficients were tested and the best ones according to author were chosen empirically.

4.5.1 NMPC constraints

The NMPC formulation was developed in the previous section and finalized in the equation 3.10. Here the chosen constraints will be discussed.

Angular velocities of the joints are constrained by 0.8rad/s. The values are chosen based on the maximum joint velocity when the manipulator operated with its factory joystick.

Angular positions of the joints are constrained by the robot shape and its working zone. They have the following values:

$$\begin{aligned} 3.14 \leq \theta_1 \leq 6.28 & \quad 2.5 \leq \theta_2 \leq 5.46 \\ 0.33 \leq \theta_3 \leq 5.95 & \quad -6.28 \leq \theta_4 \leq 6.28 \\ -6.28 \leq \theta_5 \leq 6.28 & \quad -6.28 \leq \theta_6 \leq 6.28 \end{aligned}$$

The maximum Cartesian velocity of the end-effector is limited by 0.4m/s for linear component and 0.8rad/s for angular component. They were chosen such that maximum velocity would be approximately the same as with factory joystick and it will not be too fast from the user perspective. Also, the end-effector position is constrained by the lowest level of 5cm above the ground.

Finally, there is a constraint to partially avoid visual occlusion of a target object by the manipulator movement and it is similar to one used in [32]. Suppose there are a reference object and an obstacle object which can potentially create an occlusion.

Suppose the coordinates of the reference object and the obstacle object in the camera frame are known and are given by two vectors α and β respectively as shown in Figure 4-4. It is desirable that the obstacle does not enter a cone which goes from the origin of the camera frame to the reference object along α and has base radius r . It means that the distance d from the reference to the projection of the obstacle onto the plane of the base of the cone should be greater than radius r of the base of the cone at all times:

$$d^2 > r^2$$

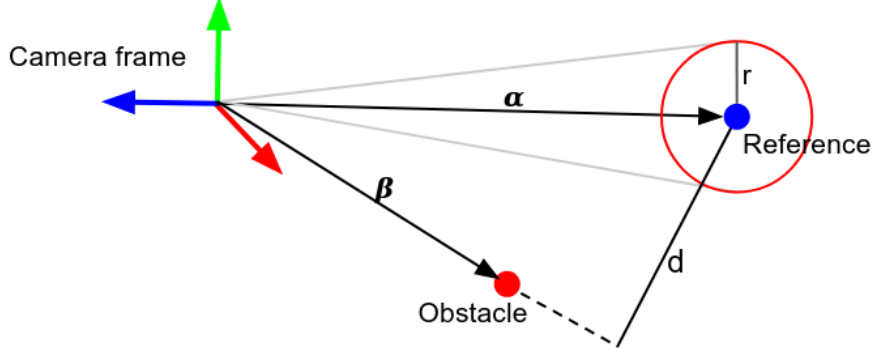


Figure 4-4: Visual occlusion avoidance

squaring both sides to avoid performing square root operation. d can be calculated as difference between α and its projection onto β :

$$d^2 = \alpha^T \left(I - \frac{\beta \beta^T}{\beta^T \beta} \right) \alpha$$

Therefore the final equation for visual occlusion has the following form:

$$\alpha^T \left(I - \frac{\beta \beta^T}{\beta^T \beta} \right) \alpha - r^2 > 0 \quad (4.4)$$

This equation is imposed as a constraint for the manipulator's third and fifth joints. The coordinates of the joints are calculated based on the forward kinematics of the manipulator and transformed into the camera frame by the transform estimated during the system calibration phase.

4.5.2 Choice of weights for weighting matrices

As it was discussed before, weighting matrices $W_k \in \mathbf{R}^{18 \times 18}$ and $W_N \in \mathbf{R}^{6 \times 6}$ from equation 3.10 are assigned dynamically. Weights are changed depending on the joystick control state discussed in subsection 4.4.1. When we are in the position control mode, we do not want to change orientation so weights should be changed so as to retain end-effector orientation. In the same way, when we are in the orientation control mode, weights should be changed so that end-effector position remains the same.

The reference functions were defined in the subsection 3.3.2 and have the following form:

$$h(x_k, u_k) = \begin{bmatrix} \mathbf{r}_{ee} \\ \delta \mathbf{q}_{ee} \\ \mathbf{v}_{ee} \\ \boldsymbol{\omega}_{ee} \\ \dot{\boldsymbol{\theta}} \end{bmatrix} \in \mathbf{R}^{18} \quad (4.5)$$

$$h_N(x_N) = \begin{bmatrix} \mathbf{r}_{ee} \\ \delta \mathbf{q}_{ee} \end{bmatrix} \in \mathbf{R}^6 \quad (4.6)$$

When we are in the position control mode, we need to assign much higher weights to the end-effector's orientation compared to the end-effector's position. At the same time we need to have weights responsible for the end-effector's velocity much higher than for the end-effector's pose. Also weights responsible for the end-effector's linear velocity should be significantly higher than weights for the end-effector's angular velocity. Meanwhile weights for joint velocities can be set to a very low level, because they are there just to be sure that we do not exceed joint velocity limit. Therefore, the corresponding weighting matrices during position control mode are the following:

$$W_k = \begin{bmatrix} 5e^{-4} \cdot \mathbf{I}_{3 \times 3} & \dots & \dots & \dots & \dots \\ \dots & 5e^{-1} \cdot \mathbf{I}_{3 \times 3} & \dots & \dots & \dots \\ \dots & \dots & 5e^4 \cdot \mathbf{I}_{3 \times 3} & \dots & \dots \\ \dots & \dots & \dots & 1e^2 \cdot \mathbf{I}_{3 \times 3} & \dots \\ \dots & \dots & \dots & \dots & 1e^{-8} \cdot \mathbf{I}_{6 \times 6} \end{bmatrix} \quad (4.7a)$$

$$W_N = \begin{bmatrix} 1e^{-6} \cdot \mathbf{I}_{3 \times 3} & \dots \\ \dots & 1e^{-3} \cdot \mathbf{I}_{3 \times 3} \end{bmatrix} \quad (4.7b)$$

and dots are filled with zeros.

Similarly, when we are in the orientation control mode, weights for the end-

effector's position are much higher than weights for the end-effector's orientation. Weights for the end-effector's linear velocity are now significantly lower than weights for the end-effector's angular velocity. The overall scale of the weighting matrices stays the same as in the position control mode and in the end, they will have the following form:

$$W_k = \begin{bmatrix} 3e4 \cdot \mathbf{I}_{3 \times 3} & \dots & \dots & \dots & \dots \\ \dots & 5e^{-4} \cdot \mathbf{I}_{3 \times 3} & \dots & \dots & \dots \\ \dots & \dots & 1e^2 \cdot \mathbf{I}_{3 \times 3} & \dots & \dots \\ \dots & \dots & \dots & 5e^4 \cdot \mathbf{I}_{3 \times 3} & \dots \\ \dots & \dots & \dots & \dots & 1e^{-8} \cdot \mathbf{I}_{6 \times 6} \end{bmatrix} \quad (4.8a)$$

$$W_N = \begin{bmatrix} 1e^{-3} \cdot \mathbf{I}_{3 \times 3} & \dots \\ \dots & 1e^{-6} \cdot \mathbf{I}_{3 \times 3} \end{bmatrix} \quad (4.8b)$$

There is also an option of fully autonomous movement. In this case all weights except weights for joint velocities are on the same order. This way user still is able to send velocity commands to controller, but it will be less responsive and will always try to reach the target. Weighting matrices in this case have the following form:

$$W_k = \begin{bmatrix} 5e-4 \cdot \mathbf{I}_{3 \times 3} & \dots & \dots & \dots & \dots \\ \dots & 5e^{-4} \cdot \mathbf{I}_{3 \times 3} & \dots & \dots & \dots \\ \dots & \dots & 1e^{-3} \cdot \mathbf{I}_{3 \times 3} & \dots & \dots \\ \dots & \dots & \dots & 5e^{-3} \cdot \mathbf{I}_{3 \times 3} & \dots \\ \dots & \dots & \dots & \dots & 1e^{-8} \cdot \mathbf{I}_{6 \times 6} \end{bmatrix} \quad (4.9a)$$

$$W_N = \begin{bmatrix} 1e^{-6} \cdot \mathbf{I}_{3 \times 3} & \dots \\ \dots & 1e^{-6} \cdot \mathbf{I}_{3 \times 3} \end{bmatrix} \quad (4.9b)$$

Chapter 5

Results

This chapter will present graphics showing performance of the implemented system. For the tests two scenarios were chosen: fully autonomous object reaching and tele-operated object reaching.

5.1 Autonomous reaching

During fully autonomous reaching, user does not provide reference velocities to the controller. The controller automatically chooses joint trajectories to reach the desired goal configuration and therefore controls both linear and angular velocities of the end-effector.

Figure 5-1 visualizes the end-effector's trajectory during autonomous object reaching. The cone is the visualization of the visual constraint with its tip being the origin of the camera frame and the center of the base being the position of the reference object. The manipulator's base frame is located at the origin. The axis limits show approximately the allowed working zone of the manipulator. It can be seen that the end-effector follows a smooth trajectory and does not show an unexpected behaviour.

Figure 5-2a shows end-effector position error with respect to Cartesian reference position and Figure 5-2b shows distance left to the object. The motion is very smooth and there is practically no overshooting. The position error is less than 1cm in approximately 10s. A little notch on the graph along Z axis is when end-effector

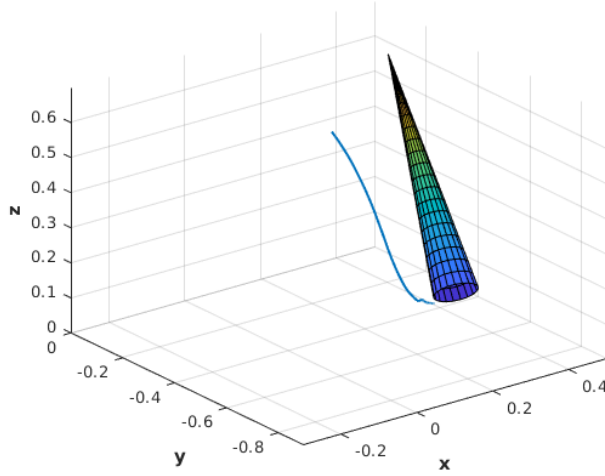


Figure 5-1: End-effector's trajectory during autonomous reaching

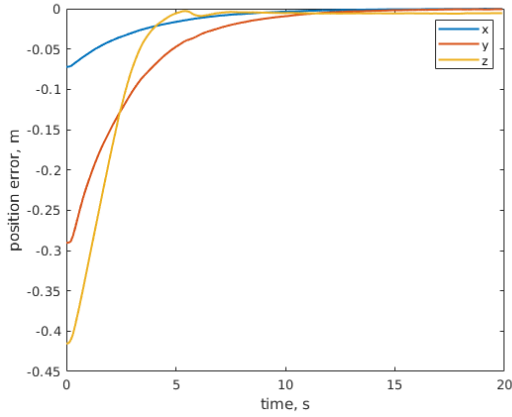
almost reached ground and was pushed up a little to respect constraints.

Figure 5-3 shows joint trajectories during the autonomous reaching. After approximately 14s the joint positions and joint velocities almost reached steady state values. None of the constraints imposed on joint positions and joint velocities are violated - joint positions are within margins for each corresponding joint and joint velocities are lower than allowed 0.8rad/s. It can be seen that most of the displacement occurred during the initial 8 seconds, then it slowly converges to the ideal position.

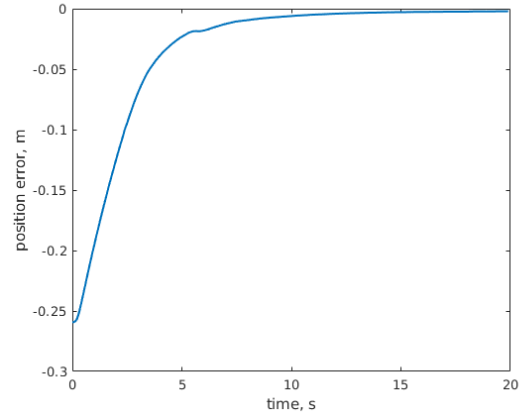
Figure 5-4 shows the plot of the end-effector's linear velocity in the robot base frame. It can be seen that the input reference velocity is zero at all times and the manipulator moves autonomously. Also note that the linear velocities do not exceed the allowed 0.4m/s limit. The graphics are noisy because the manipulator's driver does not provide end-effector velocities. They were estimated using simple first order derivative of the end-effector's position which is also has a little noisy.

5.2 Teleoperated reaching

During teleoperated reaching the user sends linear velocity commands, and the controller tries to follow reference linear velocities while maintaining end-effector orientation. There is still option to switch joystick control mode as was explained in section

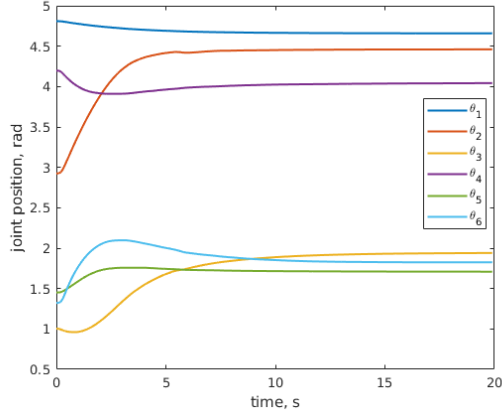


(a) Position error for each axis

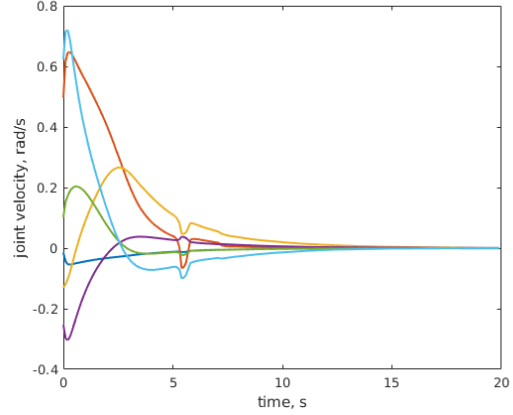


(b) Distance to the object

Figure 5-2: End-effector error during autonomous object reaching



(a) Joint positions



(b) Joint velocities

Figure 5-3: Joint positions and joint velocities during the autonomous reaching

4.4.1, however the idea is that the controller will do the orientation adjustments automatically and the user would not need to switch the modes.

Figure 5-5a visualizes the trajectory followed by the end-effector during teleoperated reaching. Here the reference object is the same as with the autonomous reaching task. Note the notch at the bottom of the trajectory. It was intentional to verify that the ground constraint is working and the end-effector will not go lower than 5 cm. Because the velocity command was very high, the end-effector almost bumped into the ground constraint, however it bounced off resulting in the notch. It can be seen in Figure 5-5b that the constraint was not violated. There is another factor for the

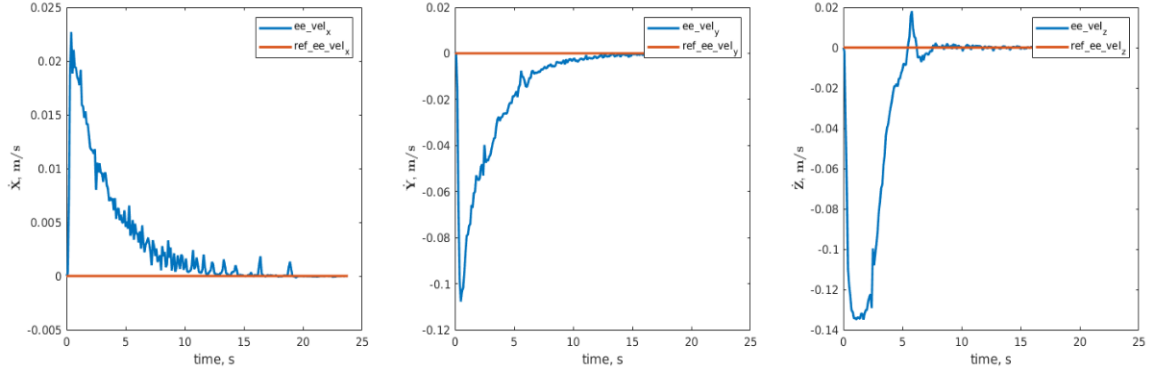
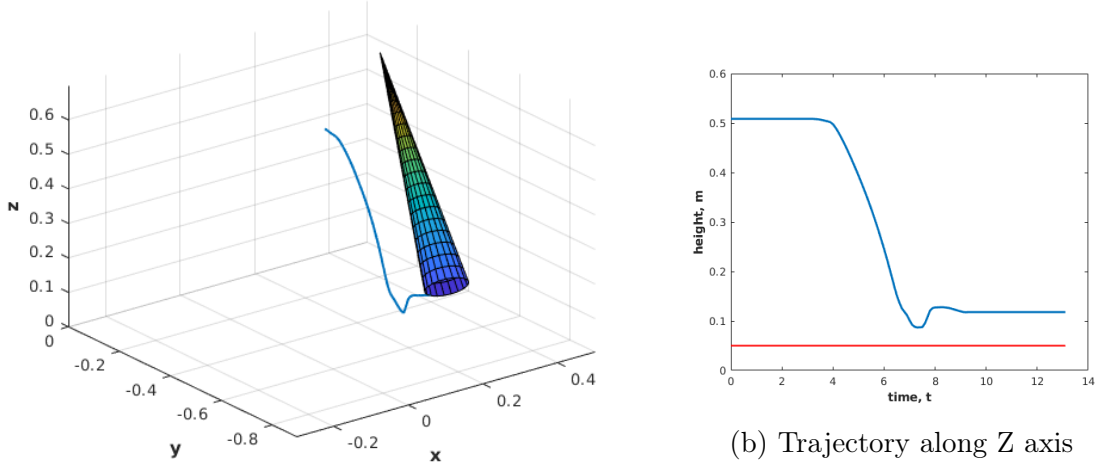


Figure 5-4: End-effector velocity during autonomous grasping



(a) 3D trajectory

(b) Trajectory along Z axis

Figure 5-5: End-effector trajectory during teleoperated reaching

notch, which is the autonomous orientation control. When the linear velocity is very high, the controller can't keep up and the orientation drifts a little. Then when reference linear velocity drops, the controller catches up and adjusts the orientation. And when adjusting orientation, the end-effector can enter a short period of oscillatory motion.

Figure 5-6 shows end-effector reference velocity (red) vs. actual end-effector velocity (blue). As it can be seen the controller is able to follow the reference most of the time. The velocity spikes happen at the time when end-effector has reached the ground. However the constraints are never violated and linear velocity remains

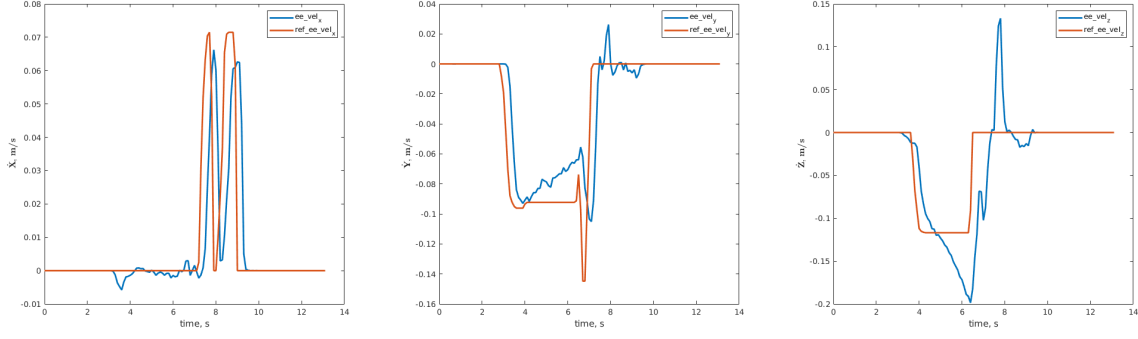


Figure 5-6: End-effector reference and actual velocity during teleoperated grasping

inside the allowed margin. It should be noted that there is a latency approximately 200-250ms between the control signal and the actual execution of the control signal. This is expected as the controller runs with the frequency of 10Hz or equivalently with the period of 100ms. It takes at least one control loop iteration to react to a new reference. Taking into consideration the complexity of the system model and the previous moving horizon solutions, it is normal to have MPC converge to a new solution in two control loop iterations.

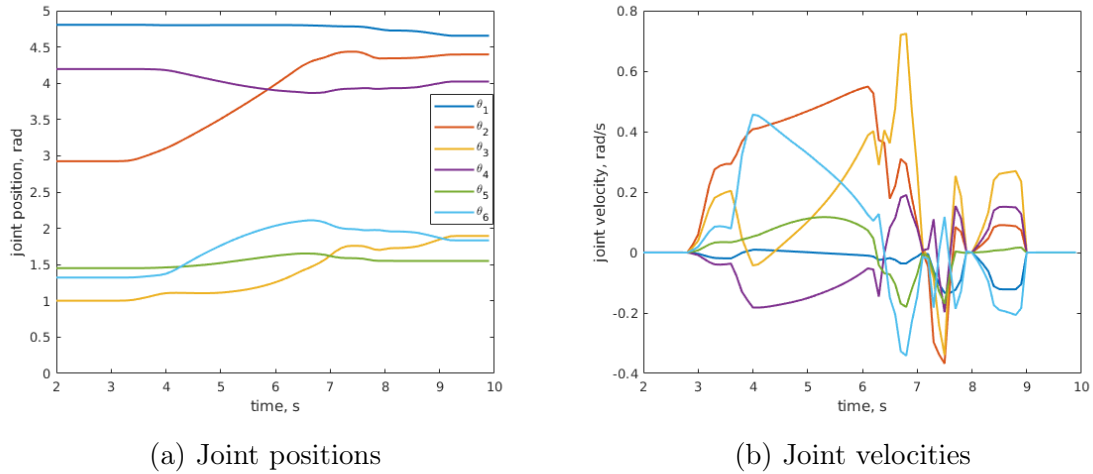


Figure 5-7: Joint positions and joint velocities during the teleoperated reaching

Figure 5-7 shows joint positions and joint velocities during teleoperated object reaching task. It can be seen that the joint velocities are much more disordered compared to joint velocities during autonomous object reaching in Figure 5-3b. This is due to the fact that a user can't control so many degrees of freedom in parallel and

does not optimize for joint positions or joint velocities. The sole intention of the user is to grab an object the way they feel will be correct.

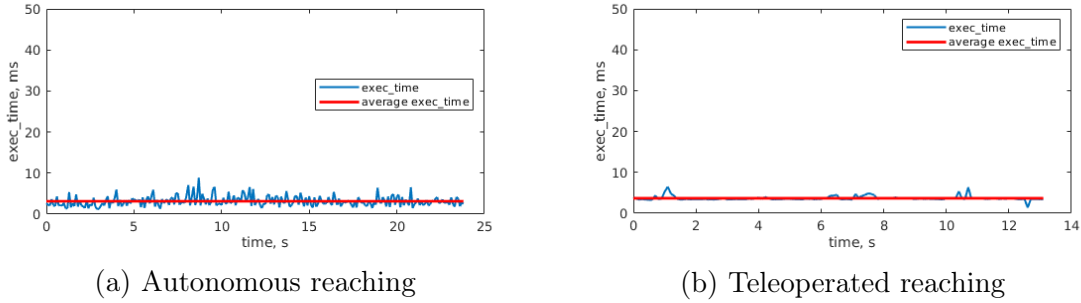


Figure 5-8: Feedback step execution time

Figure 5-8 shows execution times for controller’s feedback step during autonomous and teleoperated object reaching. They are very similar with average execution time of 3ms for autonomous reaching and 3.4ms for teleoperated reaching.

5.3 Discussion and future work

The results presented in this work still need revision and improvement. For example, the weights are not perfectly chosen and should be tuned further to obtain a smooth end-effector velocity even in the presense of rapidly changing user control signals. One way of doing so may be weights scaling based on the user input and distance to the object. This is similar to the methods described in the literature review section and is planned to be implemented.

The controller frequency should be increased to a higher level. Previously, the low controller frequency was chosen because the solver was not optimized and it was taking more than 40ms on average to perform a single feedback step and was approximately around 60ms in some cases. So it was not possible to increase controller frequency further. Now with feedback step taking less than 10ms, it is possible to implement the controller with higher update rate to reduce latency and achive more smooth motions.

Another feature that I would like to add is to increase robot speed in the final stage of autonomous reaching to improve task completion time. As it can be seen from

Figure 5-3b, currently, most of the motion is performed in the first half of the process. At this stage manipulator's end-effector has almost reached the target object. For big objects like a bottle or a cup, this accuracy is already good and the object can be grasped. But for smaller objects like box of vitamins, this accuracy is not sufficient and user would have to wait a little longer until end-effector's position converges. Is it possible that increasing the controller frequency will partially solve this problem. However it is probably worth to try putting constraints on minimum joint velocities.

Finally, there is still shared autonomy scheme to be implemented. I would like to first try blending approaches such as those described in the literature review section. I will try different arbitration methods and particularly want to try arbitration based on Markov processes.

Chapter 6

Conclusion

In this thesis work a manipulator controller based on MPC was developed, which can be used in the future for developing a shared autonomy scheme. The main feature of this controller is the system model which is described by the Jacobian matrix of the manipulator. This way it is possible to give the controller reference velocity in Cartesian space and at the same time reference pose in the Cartesian space.

In parallel with regular constraints like joint position and joint velocity limits, the controller also includes constraints for avoiding visual occlusion of a target by the manipulator. This is done by introducing a line of sight cone with its tip at the camera frame and the base at the target.

The controller makes use of variable weighting matrices to adapt MCP solver for different control modes. There are position mode, orientation mode and finger control mode. Additionally, there is a set of weights for an autonomous task execution, where user still can change end-effector's velocity, but the controller will try to reach the target pose.

Appendix A

Jacobian generation for Kinova Jaco v2 robotic manipulator

```
1 clear all
2 close all
3 %% Real Jaco robot quats and DH rotations
4 % define all symbolic variables
5 syms theta_1 theta_2 theta_3 theta_4 theta_5 theta_6 D1 D2 D3 D4 D5 D6 aa e2
   real
6 syms theta_dot_1 theta_dot_2 theta_dot_3 theta_dot_4 theta_dot_5 theta_dot_6 real
7 assume(theta_1<10 & theta_1>-10)
8 assume(theta_2<10 & theta_2>-10)
9 assume(theta_3<10 & theta_3>-10)
10 assume(theta_4<10 & theta_4>-10)
11 assume(theta_5<10 & theta_5>-10)
12 assume(theta_6<10 & theta_6>-10)
13
14 u = [theta_dot_1; theta_dot_2; theta_dot_3; theta_dot_4; theta_dot_5; theta_dot_6
      ];
15
16 % define all constants
17 % robot length values (metres)
18 D1 = 0.2755;
19 D2 = 0.4100;
```

```

20 D3 = 0.2073;
21 D4 = 0.0743;
22 D5 = 0.0743;
23 D6 = 0.1687;
24 e2 = 0.0098;
25
26 % alternate parameters
27 aa = pi/6;
28 ca = cos(aa);
29 sa = sin(aa);
30 c2a = cos(2*aa);
31 s2a = sin(2*aa);
32 d4b = D3 + sa/s2a*D4;
33 d5b = sa/s2a*D4 + sa/s2a*D5;
34 d6b = sa/s2a*D5 + D6;
35
36 % prepare all data
37 alpha=[pi/2 pi pi/2 aa*2 aa*2 pi];
38 a=[0 D2 0 0 0 0 0];
39 d=[D1 0 -e2 -d4b -d5b -d6b];
40 q=[-theta_1 theta_2-pi/2 theta_3+pi/2 theta_4 theta_5-pi theta_6+pi/2];
41
42 % calculate forward kinematics
43 i = 1;
44 T1 = [cos(q(i)) -sin(q(i))*cos(alpha(i)) sin(q(i))*sin(alpha(i)) a(i)*cos(q(i));
45       sin(q(i)) cos(q(i))*cos(alpha(i)) -cos(q(i))*sin(alpha(i)) a(i)*sin(q(i));
46       0 sin(alpha(i)) cos(alpha(i)) d(i);
47       0 0 0 1];
48
49 i = 2;
50 T2 = [cos(q(i)) -sin(q(i))*cos(alpha(i)) sin(q(i))*sin(alpha(i)) a(i)*cos(q(i));
51       sin(q(i)) cos(q(i))*cos(alpha(i)) -cos(q(i))*sin(alpha(i)) a(i)*sin(q(i));
52       0 sin(alpha(i)) cos(alpha(i)) d(i);
53       0 0 0 1];
54
55 i = 3;

```



```

56 T3 = [cos(q(i)) -sin(q(i))*cos(alpha(i)) sin(q(i))*sin(alpha(i)) a(i)*cos(q(i));
57        sin(q(i)) cos(q(i))*cos(alpha(i)) -cos(q(i))*sin(alpha(i)) a(i)*sin(q(i));
58        0 sin(alpha(i)) cos(alpha(i)) d(i);
59        0 0 0 1];
60
61 i = 4;
62 T4 = [cos(q(i)) -sin(q(i))*cos(alpha(i)) sin(q(i))*sin(alpha(i)) a(i)*cos(q(i));
63        sin(q(i)) cos(q(i))*cos(alpha(i)) -cos(q(i))*sin(alpha(i)) a(i)*sin(q(i));
64        0 sin(alpha(i)) cos(alpha(i)) d(i);
65        0 0 0 1];
66
67 i = 5;
68 T5 = [cos(q(i)) -sin(q(i))*cos(alpha(i)) sin(q(i))*sin(alpha(i)) a(i)*cos(q(i));
69        sin(q(i)) cos(q(i))*cos(alpha(i)) -cos(q(i))*sin(alpha(i)) a(i)*sin(q(i));
70        0 sin(alpha(i)) cos(alpha(i)) d(i);
71        0 0 0 1];
72
73 i = 6;
74 T6 = [cos(q(i)) -sin(q(i))*cos(alpha(i)) sin(q(i))*sin(alpha(i)) a(i)*cos(q(i));
75        sin(q(i)) cos(q(i))*cos(alpha(i)) -cos(q(i))*sin(alpha(i)) a(i)*sin(q(i));
76        0 sin(alpha(i)) cos(alpha(i)) d(i);
77        0 0 0 1];
78
79 T06 = T1*T2*T3*T4*T5*T6;
80
81
82 % extract rotation matrixes
83 R01 = T1(1:3,1:3);
84 R12 = T2(1:3,1:3);
85 R23 = T3(1:3,1:3);
86 R34 = T4(1:3,1:3);
87 R45 = T5(1:3,1:3);
88 R56 = T6(1:3,1:3);
89
90 % directions of the joint axes z(i-1)
91 k = [0 0 1]';

```

```

92 z0 = k;
93 z1 = R01*k;
94 z2 = R01*R12*k;
95 z3 = R01*R12*R23*k;
96 z4 = R01*R12*R23*R34*k;
97 z5 = R01*R12*R23*R34*R45*k;
98
99 % extract position vector p
100 p6 = T06(1:3,4);
101
102 % Construct Geometric Jacobian
103 J = simplify([diff(p6,theta_1) diff(p6,theta_2) diff(p6,theta_3) diff(p6,theta_4)
               diff(p6,theta_5) diff(p6,theta_6);...
               z0 z1 z2 z3 z4 z5])
104
105
106 % simplify Jacobian
107 threshold = 1e-6;
108 Jsimpl = vpa(mapSymType(J, 'vpareal', @(x) piecewise(abs(x)<=threshold, 0, x)),
               8)
109
110 % extract velocity vector and simplify
111 v = vpa(simplify(Jsimpl*u),8);
112 v = vpa(mapSymType(v, 'vpareal', @(x) piecewise(abs(x)<=threshold, 0, x)), 8)
113
114 % write velocity vector to a text file
115 fid = fopen('res.txt', 'w' );
116 for i=1:6
117     fprintf(fid, ' %s;\n', char( vpa(expand(v(i), 'ArithmeticOnly', true),6) ));
118 end
119 fclose(fid);
120 disp("THE END");

```

Appendix B

NMPC formulation for ACADO Code Generation

```
1 #include <acado_code_generation.hpp>
2
3 using namespace std;
4
5 USING_NAMESPACE_ACADO
6
7 int main( )
8 {
9     USING_NAMESPACE_ACADO
10
11
12     // INTRODUCE THE VARIABLES:
13     // -----
14     DifferentialState      theta_1; // joint 1 angular position
15     DifferentialState      theta_2; // joint 2 angular position
16     DifferentialState      theta_3; // joint 3 angular position
17     DifferentialState      theta_4; // joint 4 angular position
18     DifferentialState      theta_5; // joint 5 angular position
19     DifferentialState      theta_6; // joint 6 angular position
20
21     DifferentialState      x; // x-end effector
```

```

22 DifferentialState      y; // y-end effector
23 DifferentialState      z; // z-end effector
24 DifferentialState      q0; // end-effector orientation w
25 DifferentialState      q1; // end-effector orientation x
26 DifferentialState      q2; // end-effector orientation y
27 DifferentialState      q3; // end-effector orientation z
28
29 Control      theta_dot_1; // joint 1 angular velocity
30 Control      theta_dot_2; // joint 2 angular velocity
31 Control      theta_dot_3; // joint 3 angular velocity
32 Control      theta_dot_4; // joint 4 angular velocity
33 Control      theta_dot_5; // joint 5 angular velocity
34 Control      theta_dot_6; // joint 6 angular velocity
35
36 double D1 = 0.2755;          // Base to elbow
37 double D2 = 0.4100;          // Arm Length
38 double D3 = 0.2073;          // Front arm length
39 double D4 = 0.0743;          // First wrist length
40 double D5 = 0.0743;          // Second wrist lenght
41 double D6 = 0.1687;          // Wrist to center of the hand
42 double e2 = 0.0098;
43 double pi = 3.1415;
44
45 OnlineData q0ref;
46 OnlineData q1ref;
47 OnlineData q2ref;
48 OnlineData q3ref;
49
50 IntermediateState s1 = sin(theta_1); IntermediateState s2 = sin(theta_2);
51 IntermediateState s3 = sin(theta_3); IntermediateState s4 = sin(theta_4);
52 IntermediateState s5 = sin(theta_5); IntermediateState s6 = sin(theta_6);
53
54 IntermediateState c1 = cos(theta_1); IntermediateState c2 = cos(theta_2);
55 IntermediateState c3 = cos(theta_3); IntermediateState c4 = cos(theta_4);
56 IntermediateState c5 = cos(theta_5); IntermediateState c6 = cos(theta_6);
57

```

```

IntermediateState vx = 0.41*theta_dot_2*c1*c2 - 0.0098*theta_dot_1*c1 +
0.165924*theta_dot_1*c1*c4 - 0.41*theta_dot_1*s1*s2 - 0.165924*theta_dot_4*s1*
s4 - 0.345994*theta_dot_2*c1*c2*c3 + 0.345994*theta_dot_3*c1*c2*c3 -
0.0916242*theta_dot_1*c1*c4*c5 - 0.345994*theta_dot_1*c2*s1*s3 + 0.345994*
theta_dot_1*c3*s1*s2 - 0.345994*theta_dot_2*c1*s2*s3 + 0.345994*theta_dot_3*c1
*s2*s3 + 0.183248*theta_dot_1*c1*s4*s5 + 0.183248*theta_dot_4*c4*s1*s5 +
0.0916242*theta_dot_4*c5*s1*s4 + 0.0916242*theta_dot_5*c4*s1*s5 + 0.183248*
theta_dot_5*c5*s1*s4 - 0.158698*theta_dot_2*c1*c2*c3*c5 + 0.158698*theta_dot_3
*c1*c2*c3*c5 + 0.165924*theta_dot_4*c1*c2*c3*c4 - 0.165924*theta_dot_1*c2*c3*
s1*s4 - 0.158698*theta_dot_1*c2*c5*s1*s3 + 0.158698*theta_dot_1*c3*c5*s1*s2 +
0.165924*theta_dot_2*c1*c2*s3*s4 - 0.165924*theta_dot_2*c1*c3*s2*s4 -
0.158698*theta_dot_2*c1*c5*s2*s3 - 0.165924*theta_dot_3*c1*c2*s3*s4 +
0.165924*theta_dot_3*c1*c3*s2*s4 + 0.158698*theta_dot_3*c1*c5*s2*s3 +
0.165924*theta_dot_4*c1*c4*s2*s3 - 0.158698*theta_dot_5*c1*c2*s3*s5 +
0.158698*theta_dot_5*c1*c3*s2*s5 - 0.165924*theta_dot_1*s1*s2*s3*s4 -
0.0916242*theta_dot_4*c1*c2*c3*c4*c5 - 0.183248*theta_dot_5*c1*c2*c3*c4*c5 +
0.183248*theta_dot_1*c2*c3*c4*s1*s5 + 0.0916242*theta_dot_1*c2*c3*c5*s1*s4 -
0.183248*theta_dot_2*c1*c2*c4*s3*s5 - 0.0916242*theta_dot_2*c1*c2*c5*s3*s4 +
0.183248*theta_dot_2*c1*c3*c4*s2*s5 + 0.0916242*theta_dot_2*c1*c3*c5*s2*s4 +
0.183248*theta_dot_3*c1*c2*c4*s3*s5 + 0.0916242*theta_dot_3*c1*c2*c5*s3*s4 -
0.183248*theta_dot_3*c1*c3*c4*s2*s5 - 0.0916242*theta_dot_3*c1*c3*c5*s2*s4 +
0.183248*theta_dot_4*c1*c2*c3*s4*s5 - 0.0916242*theta_dot_4*c1*c4*c5*s2*s3 +
0.0916242*theta_dot_5*c1*c2*c3*s4*s5 - 0.183248*theta_dot_5*c1*c4*c5*s2*s3 +
0.183248*theta_dot_1*c4*s1*s2*s3*s5 + 0.0916242*theta_dot_1*c5*s1*s2*s3*s4 +
0.183248*theta_dot_4*c1*s2*s3*s4*s5 + 0.0916242*theta_dot_5*c1*s2*s3*s4*s5;

```

```

IntermediateState vy = 0.0098*theta_dot_1*s1 - 0.41*theta_dot_1*c1*s2 - 0.41*
theta_dot_2*c2*s1 - 0.165924*theta_dot_1*c4*s1 - 0.165924*theta_dot_4*c1*s4 -
0.345994*theta_dot_1*c1*c2*s3 + 0.345994*theta_dot_1*c1*c3*s2 + 0.345994*
theta_dot_2*c2*c3*s1 - 0.345994*theta_dot_3*c2*c3*s1 + 0.0916242*theta_dot_1*
c4*c5*s1 + 0.183248*theta_dot_4*c1*c4*s5 + 0.0916242*theta_dot_4*c1*c5*s4 +
0.0916242*theta_dot_5*c1*c4*s5 + 0.183248*theta_dot_5*c1*c5*s4 + 0.345994*
theta_dot_2*s1*s2*s3 - 0.345994*theta_dot_3*s1*s2*s3 - 0.183248*theta_dot_1*s1
*s4*s5 - 0.165924*theta_dot_1*c1*c2*c3*s4 - 0.158698*theta_dot_1*c1*c2*c5*s3 +
0.158698*theta_dot_1*c1*c3*c5*s2 + 0.158698*theta_dot_2*c2*c3*c5*s1 -
0.158698*theta_dot_3*c2*c3*c5*s1 - 0.165924*theta_dot_4*c2*c3*c4*s1 -
0.165924*theta_dot_1*c1*s2*s3*s4 - 0.165924*theta_dot_2*c2*s1*s3*s4 +

```

$0.165924*\theta_{\dot{2}}*c_3*s_1*s_2*s_4 + 0.158698*\theta_{\dot{2}}*c_5*s_1*s_2*s_3 +$
 $0.165924*\theta_{\dot{3}}*c_2*s_1*s_3*s_4 - 0.165924*\theta_{\dot{3}}*c_3*s_1*s_2*s_4 -$
 $0.158698*\theta_{\dot{3}}*c_5*s_1*s_2*s_3 - 0.165924*\theta_{\dot{4}}*c_4*s_1*s_2*s_3 +$
 $0.158698*\theta_{\dot{5}}*c_2*s_1*s_3*s_5 - 0.158698*\theta_{\dot{5}}*c_3*s_1*s_2*s_5 +$
 $0.183248*\theta_{\dot{1}}*c_1*c_2*c_3*c_4*s_5 + 0.0916242*\theta_{\dot{1}}*c_1*c_2*c_3*c_5*s_4 +$
 $0.0916242*\theta_{\dot{4}}*c_2*c_3*c_4*c_5*s_1 + 0.183248*\theta_{\dot{5}}*c_2*c_3*c_4*c_5*s_1 +$
 $0.183248*\theta_{\dot{1}}*c_1*c_4*s_2*s_3*s_5 + 0.0916242*\theta_{\dot{1}}*c_1*c_5*s_2*s_3*s_4 +$
 $0.183248*\theta_{\dot{2}}*c_2*c_4*s_1*s_3*s_5 + 0.0916242*\theta_{\dot{2}}*c_2*c_5*s_1*s_3*s_4 -$
 $0.183248*\theta_{\dot{2}}*c_3*c_4*s_1*s_2*s_5 - 0.0916242*\theta_{\dot{2}}*c_3*c_5*s_1*s_2*s_4 -$
 $0.183248*\theta_{\dot{3}}*c_2*c_4*s_1*s_3*s_5 - 0.0916242*\theta_{\dot{3}}*c_2*c_5*s_1*s_3*s_4 +$
 $0.183248*\theta_{\dot{3}}*c_3*c_4*s_1*s_2*s_5 + 0.0916242*\theta_{\dot{3}}*c_3*c_5*s_1*s_2*s_4 -$
 $0.183248*\theta_{\dot{4}}*c_2*c_3*s_1*s_4*s_5 + 0.0916242*\theta_{\dot{4}}*c_4*c_5*s_1*s_2*s_3 -$
 $0.0916242*\theta_{\dot{5}}*c_2*c_3*s_1*s_4*s_5 + 0.183248*\theta_{\dot{5}}*c_4*c_5*s_1*s_2*s_3 -$
 $0.183248*\theta_{\dot{4}}*s_1*s_2*s_3*s_4*s_5 - 0.0916242*\theta_{\dot{5}}*s_1*s_2*s_3*s_4*s_5;$

60 IntermediateState vz = $0.41*\theta_{\dot{2}}*s_2 + 0.158698*\theta_{\dot{3}}*\sin(\theta_2 - \theta_3)*c_5 -$
 $0.158698*\theta_{\dot{5}}*\cos(\theta_2 - \theta_3)*s_5 + 0.0916242*$
 $\theta_{\dot{3}}*\cos(\theta_2 - \theta_3 + \theta_4)*s_5 + 0.0916242*\theta_{\dot{3}}*\cos(\theta_3 - \theta_2 + \theta_4)*s_5 +$
 $0.0458121*\theta_{\dot{3}}*\sin(\theta_2 - \theta_3 + \theta_4)*c_5 + 0.0458121*\theta_{\dot{3}}*\sin(\theta_3 - \theta_2 + \theta_4)*c_5 +$
 $0.345994*\theta_{\dot{2}}*c_2*s_3 - 0.345994*\theta_{\dot{2}}*c_3*s_2 - 0.345994*\theta_{\dot{3}}*c_2*s_3 +$
 $0.165924*\theta_{\dot{2}}*c_2*c_3*s_4 + 0.158698*\theta_{\dot{2}}*c_2*c_5*s_3 -$
 $0.158698*\theta_{\dot{2}}*c_3*c_5*s_2 + 0.165924*\theta_{\dot{2}}*s_2*s_3*s_4 - 0.0916242*$
 $\theta_{\dot{4}}*\sin(\theta_2 - \theta_3)*c_4*c_5 - 0.183248*\theta_{\dot{5}}*\sin(\theta_2 - \theta_3)*c_4*c_5 +$
 $0.183248*\theta_{\dot{4}}*\sin(\theta_2 - \theta_3)*s_4*s_5 + 0.0916242*\theta_{\dot{5}}*\sin(\theta_2 - \theta_3)*s_4*s_5 -$
 $0.183248*\theta_{\dot{2}}*c_2*c_3*c_4*s_5 - 0.0916242*\theta_{\dot{2}}*c_2*c_3*c_5*s_4 -$
 $0.183248*\theta_{\dot{2}}*c_4*s_2*s_3*s_5 - 0.0916242*\theta_{\dot{2}}*c_5*s_2*s_3*s_4;$

61 IntermediateState wx = $\theta_{\dot{3}}*s_1 - \theta_{\dot{2}}*s_1 - 0.866025*\theta_{\dot{5}}*c_4*s_1 -$
 $0.433013*\theta_{\dot{6}}*c_4*s_1 - \theta_{\dot{4}}*c_1*c_2*s_3 + \theta_{\dot{4}}*c_1*c_3*s_2 - 0.5*\theta_{\dot{5}}*c_1*c_2*s_3 +$
 $0.5*\theta_{\dot{5}}*c_1*c_3*s_2 - 0.25*\theta_{\dot{6}}*c_1*c_2*s_3 + 0.25*\theta_{\dot{6}}*c_1*c_3*s_2 +$
 $0.433013*\theta_{\dot{6}}*c_4*c_5*s_1 - 0.866025*\theta_{\dot{6}}*s_1*s_4*s_5 - 0.866025*\theta_{\dot{5}}*c_1*c_2*c_3*s_4 -$
 $0.433013*\theta_{\dot{6}}*c_1*c_2*c_3*s_4 - 0.75*\theta_{\dot{6}}*c_1*c_2*c_5*s_3 + 0.75*\theta_{\dot{6}}*c_1*c_3*c_5*s_2 -$
 $0.866025*\theta_{\dot{5}}*c_1*s_2*s_3*s_4 - 0.433013*\theta_{\dot{6}}*c_1*s_2*s_3*s_4 +$
 $0.866025*\theta_{\dot{6}}*c_1*c_2*c_3*c_4*s_5 + 0.433013*\theta_{\dot{6}}*c_1*c_2*c_3*c_5*s_4 +$
 $0.866025*\theta_{\dot{6}}*c_1*c_4*s_2*s_3*s_5 + 0.433013*\theta_{\dot{6}}*c_1*c_5*s_2*s_3*s_4;$

```

62   IntermediateState wy = theta_dot_3*c1 - theta_dot_2*c1 - 0.866025*theta_dot_5
*c1*c4 - 0.433013*theta_dot_6*c1*c4 + 0.433013*theta_dot_6*c1*c4*c5 +
theta_dot_4*c2*s1*s3 - theta_dot_4*c3*s1*s2 + 0.5*theta_dot_5*c2*s1*s3 - 0.5*
theta_dot_5*c3*s1*s2 + 0.25*theta_dot_6*c2*s1*s3 - 0.25*theta_dot_6*c3*s1*s2 -
0.866025*theta_dot_6*c1*s4*s5 + 0.866025*theta_dot_5*c2*c3*s1*s4 + 0.433013*
theta_dot_6*c2*c3*s1*s4 + 0.75*theta_dot_6*c2*c5*s1*s3 - 0.75*theta_dot_6*c3*
c5*s1*s2 + 0.866025*theta_dot_5*s1*s2*s3*s4 + 0.433013*theta_dot_6*s1*s2*s3*s4
- 0.866025*theta_dot_6*c2*c3*c4*s1*s5 - 0.433013*theta_dot_6*c2*c3*c5*s1*s4 -
0.866025*theta_dot_6*c4*s1*s2*s3*s5 - 0.433013*theta_dot_6*c5*s1*s2*s3*s4;
63   IntermediateState wz = theta_dot_1 - theta_dot_4*c2*c3 - 0.5*theta_dot_5*c2*
c3 - 0.25*theta_dot_6*c2*c3 - theta_dot_4*s2*s3 - 0.5*theta_dot_5*s2*s3 -
0.25*theta_dot_6*s2*s3 - 0.75*theta_dot_6*c2*c3*c5 + 0.866025*theta_dot_5*c2*
s3*s4 - 0.866025*theta_dot_5*c3*s2*s4 + 0.433013*theta_dot_6*c2*s3*s4 -
0.433013*theta_dot_6*c3*s2*s4 - 0.75*theta_dot_6*c5*s2*s3 - 0.866025*
theta_dot_6*c2*c4*s3*s5 - 0.433013*theta_dot_6*c2*c5*s3*s4 + 0.866025*
theta_dot_6*c3*c4*s2*s5 + 0.433013*theta_dot_6*c3*c5*s2*s4;

64
65
66   /*
67   * Implementation of 4.20 from JiaolePhD.pdf
68   */
69   // position of joint3 in camera frame (beta vector)
70   IntermediateState betax = 0.0072387*c1 + 0.25266*c2 + 0.0026761*s1 - 0.11196*
c1*s2 + 0.30284*s1*s2 + 0.24629;
71   IntermediateState betay = 0.0035618*c1 - 0.0093948*c2 - 0.0091275*s1 +
0.38187*c1*s2 + 0.14902*s1*s2 - 0.53482;
72   IntermediateState betaz = 0.0055635*c1 - 0.32275*c2 + 0.0023606*s1 -
0.098761*c1*s2 + 0.23276*s1*s2 - 0.21311;
73   /*
74   * position of reference in camera frame (alpha vector)
75   */
76   OnlineData prx, pry, prz;
77   /*
78   * rejection of alpha on beta
79   */
80   IntermediateState d_f_i_norm = (betax*betax*pry*pry + betax*betax*prz*prz -

```

```

2.0*betax*betay*prx*pry - 2.0*betax*betaz*prx*prz + betay*betay*prx*prx +
betay*betay*prz*prz - 2.0*betay*betaz*pry*prz + betaz*betaz*prx*prx + betaz*
betaz*pry*pry)/(betax*betax + betay*betay + betaz*betaz);

81
82
83 DifferentialEquation f;
84
85 f << dot(theta_1) == theta_dot_1;
86 f << dot(theta_2) == theta_dot_2;
87 f << dot(theta_3) == theta_dot_3;
88 f << dot(theta_4) == theta_dot_4;
89 f << dot(theta_5) == theta_dot_5;
90 f << dot(theta_6) == theta_dot_6;
91 f << dot(x) == vx;
92 f << dot(y) == vy;
93 f << dot(z) == vz;
94 f << dot(q0) == 0.5*(0.0*q0 - wx*q1 - wy*q2 - wz*q3);
95 f << dot(q1) == 0.5*(wx*q0 + 0.0*q1 - wz*q2 + wy*q3);
96 f << dot(q2) == 0.5*(wy*q0 + wz*q1 + 0.0*q2 - wx*q3);
97 f << dot(q3) == 0.5*(wz*q0 - wy*q1 + wx*q2 + 0.0*q3);
98
99 // DEFINE LEAST SQUARE FUNCTION:
100 // -----
101 Function h, hN;
102
103 IntermediateState q1_err = q0ref*q1 - q0*q1ref + q3ref*q2 - q2ref*q3;
104
105 IntermediateState q2_err = q0ref*q2 - q0*q2ref - q3ref*q1 + q1ref*q3;
106
107 IntermediateState q3_err = q0ref*q3 - q0*q3ref + q2ref*q1 - q1ref*q2;
108
109 h << x;
110 h << y;
111 h << z;
112 h << q1_err;
113 h << q2_err;

```



```

114     h << q3_err;
115
116     // x_dot
117     h << vx;
118     // y_dot
119     h << vy;
120     // z_dot
121     h << vz;
122     // wx
123     h << wx;
124     // wy
125     h << wy;
126     // wz
127     h << wz;
128
129     h << theta_dot_1;
130     h << theta_dot_2;
131     h << theta_dot_3;
132     h << theta_dot_4;
133     h << theta_dot_5;
134     h << theta_dot_6;
135
136
137     //hN - 19 elements
138     // x
139     hN << x;
140     // y
141     hN << y;
142     // z
143     hN << z;
144     // q0
145     hN << q1_err;
146     // q1
147     hN << q2_err;
148     // q2
149     hN << q3_err;

```

```

150
151
152 BMatrix Q = eye<bool>(h.getDim());
153 BMatrix QN = eye<bool>(hN.getDim());
154
155
156
157 // DEFINE AN OPTIMAL CONTROL PROBLEM:
158 // -----
159 const double tStart = 0.0;
160 const double tSampl = 0.1;
161 const double nSteps = 10;
162 const double tEnd    = tSampl*nSteps;
163
164 OCP ocp( tStart, tEnd, nSteps);
165
166 ocp.minimizeLSQ( Q, h );
167 ocp.minimizeLSQEndTerm( QN, hN);
168
169 ocp.subjectTo( f );
170
171 float theta_dot_max = 0.35;
172 ocp.subjectTo( -theta_dot_max <= theta_dot_1 <= theta_dot_max );
173 ocp.subjectTo( -theta_dot_max <= theta_dot_2 <= theta_dot_max );
174 ocp.subjectTo( -theta_dot_max <= theta_dot_3 <= theta_dot_max );
175 ocp.subjectTo( -theta_dot_max <= theta_dot_4 <= theta_dot_max );
176 ocp.subjectTo( -theta_dot_max <= theta_dot_5 <= theta_dot_max );
177 ocp.subjectTo( -theta_dot_max <= theta_dot_6 <= theta_dot_max );
178 ocp.subjectTo( 3.14 <= theta_1 <= 6.28 );
179 ocp.subjectTo( 2.50 <= theta_2 <= 5.46 );
180 ocp.subjectTo( 0.33 <= theta_3 <= 5.95 );
181 ocp.subjectTo( -6.28 <= theta_4 <= 6.28 );
182 ocp.subjectTo( -6.28 <= theta_5 <= 6.28 );
183 ocp.subjectTo( -6.28 <= theta_6 <= 6.28 );
184
185 // float v_max = 0.1;

```

```

186 // ocp.subjectTo( -v_max <= vx <= v_max );
187 // ocp.subjectTo( -v_max <= vy <= v_max );
188 // ocp.subjectTo( -v_max <= vz <= v_max );
189
190 // float w_max = 0.174;
191 // ocp.subjectTo( -w_max <= wx <= w_max );
192 // ocp.subjectTo( -w_max <= wy <= w_max );
193 // ocp.subjectTo( -w_max <= wz <= w_max );
194
195 /*
196  * Implementation of 4.20 from JiaolePhD.pdf
197  */
198 // double r = 0.03;
199 // ocp.subjectTo( d_f_i_norm - r*r >= 0 );
200
201 // self-collision avoidance
202 // double j2_x = 0, j2_y = 0, j3_z = 0.2755;
203 // double minDist = 0.2;
204 // IntermediateState constraint = (x-j2_x)*(x-j2_x) + (y-j2_y)*(y-j2_y) + (z-
j3_z)*(z-j3_z) - minDist*minDist;
205 // ocp.subjectTo(constraint>=0);
206
207 ocp.subjectTo( z >= 0.1 );
208
209 ocp.setNOD( 7 );
210
211
212 // Export the code:
213 OCPexport mpc( ocp );
214
215 mpc.set( HESSIAN_APPROXIMATION, GAUSS_NEWTON );
216 mpc.set( DISCRETIZATION_TYPE, MULTIPLE_SHOOTING );
217 mpc.set( SPARSE_QP_SOLUTION, FULL_CONDENSING_N2 );
218 mpc.set( INTEGRATOR_TYPE, INT_RK4 );
219 mpc.set( NUM_INTEGRATOR_STEPS, 1 );
220 mpc.set( QP_SOLVER, QP_QPOASES );

```

```

221     mpc.set( HOTSTART_QP, YES );
222     mpc.set( LEVENBERG_MARQUARDT, 1.0e-5 );
223     mpc.set( USE_SINGLE_PRECISION, BT_TRUE );
224     mpc.set( CG_USE_VARIABLE_WEIGHTING_MATRIX, YES );
225     mpc.set( GENERATE_TEST_FILE, BT_FALSE );
226     mpc.set( GENERATE_MAKE_FILE, BT_FALSE );
227
228     if (mpc.exportCode( "/home/robot/CatkinWorkspaces/grad_2019_jaco/src/mpc_jaco
/solver" ) != SUCCESSFUL_RETURN)
229         exit( EXIT_FAILURE );
230
231     mpc.printDimensionsQP( );
232
233     return EXIT_SUCCESS;
234 }

```

Bibliography

- [1] F. Abi-Farraj, C. Pacchierotti, O. Arenz, G. Neumann, and P. R. Giordano. A haptic shared-control architecture for guided multi-target robotic grasping. *IEEE Transactions on Haptics*, 13(2):270–285, 2020.
- [2] F. Abi-Farraj, N. Pedemonte, and P. Robuffo Giordano. A visual-based shared control architecture for remote telemanipulation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4266–4273, 2016.
- [3] Michele Arnold and Göran Andersson. Model predictive control of energy storage including uncertain forecasts. In *Power Systems Computation Conference (PSCC), Stockholm, Sweden*, volume 23, pages 24–29. Citeseer, 2011.
- [4] Connor Brooks and Daniel Szafrir. Visualization of intended assistance for acceptance of shared control. 2020.
- [5] A. Campeau-Lecours, U. Côté-Allard, D. Vu, F. Routhier, B. Gosselin, and C. Gosselin. Intuitive adaptive orientation control for enhanced human-robot interaction. *IEEE Transactions on Robotics*, 35(2):509–520, 2019.
- [6] Anca Dragan and Siddhartha Srinivasa. A policy blending formalism for shared control. *International Journal of Robotics Research*, May 2013.
- [7] John W. Eaton and James B. Rawlings. Model-predictive control of chemical processes. *Chemical Engineering Science*, 47(4):705–720, 1992.
- [8] Michael Erhard, Greg Horn, and Moritz Diehl. A quaternion-based model for optimal control of an airborne wind energy system. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 97(1):7–24, 2017.
- [9] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza. Pampc: Perception-aware model predictive control for quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, 2018.
- [10] F. Flemisch, D. Abbink, M. Itoh, M-P. Pacaux-Lemoine, and G. Weßel. Shared control is the sharp end of cooperation: Towards a common framework of joint action, shared control and human machine cooperation. *IFAC-PapersOnLine*,

49(19):72 – 77, 2016. 13th IFAC Symposium on Analysis, Design, and Evaluation of Human-Machine Systems HMS 2016.

- [11] Milad Geravand, Christian Werner, Klaus Hauer, and Angelika Peer. An integrated decision making approach for adaptive shared control of mobility assistance robots. *International Journal of Social Robotics*, 8(5):631–648, 2016.
- [12] D. Gopinath, S. Jain, and B. D. Argall. Human-in-the-loop optimization of shared autonomy in assistive robotics. *IEEE Robotics and Automation Letters*, 2(1):247–254, 2017.
- [13] D. E. Gopinath and B. D. Argall. Active intent disambiguation for shared control robots. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(6):1497–1506, 2020.
- [14] M. Gualtieri, A. ten Pas, K. Saenko, and R. Platt. High precision grasp pose detection in dense clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 598–605, 2016.
- [15] S. Jain and B. Argall. Grasp detection for assistive robotic manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2015–2021, 2016.
- [16] Shervin Javdani, Henny Admoni, Stefania Pellegrinelli, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization for teleoperation and teaming. *The International Journal of Robotics Research*, 37(7):717–742, 2018.
- [17] Z. Jin and P. R. Pagilla. Operator intent prediction with subgoal transition probability learning for shared control applications. In *2020 IEEE International Conference on Human-Machine Systems (ICHMS)*, pages 1–6, 2020.
- [18] Mina Kamel, Michael Burri, and Roland Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017. 20th IFAC World Congress.
- [19] Marian P Kazmierkowski. Model predictive control of high power converters and industrial drives [book news]. *IEEE Industrial Electronics Magazine*, 12(3):55–56, 2018.
- [20] D. Kim, R. Hazlett-Knudsen, H. Culver-Godfrey, G. Rucks, T. Cunningham, D. Portee, J. Bricout, Z. Wang, and A. Behal. How autonomy impacts performance and satisfaction: Results from a study with spinal cord injured subjects using an assistive robot. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 42(1):2–14, 2012.
- [21] Kinova. Kinova™ Ultra lightweight robotic arm. User Guide. [www.kinovarobotics.com/sites/default/files/ULWS-RA-JAC-UG-INT-EN%](http://www.kinovarobotics.com/sites/default/files/ULWS-RA-JAC-UG-INT-EN%2017-01-20.pdf)

20201804-1.0%20(KINOVA%E2%84%A2%20Ultra%20lightweight%20robotic%20arm%20user%20guide)_0.pdf, 2018.

- [22] Songpo Li, Michael Bowman, and Xiaoli Zhang. A general arbitration model for robust human-robot shared control with multi-source uncertainty modeling. *arXiv preprint arXiv:2003.05097*, 2020.
- [23] Catharine LR McGhan, Ali Nasir, and Ella M Atkins. Human intent prediction using markov decision processes. *Journal of Aerospace Information Systems*, 12(5):393–397, 2015.
- [24] Hardik Parwana and Mangal Kothari. Quaternions and attitude representation. *arXiv preprint arXiv:1708.08680*, 2017.
- [25] C. P. Quintero, O. Ramirez, and M. Jägersand. Vibi: Assistive vision-based interface for robot manipulation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4458–4463, 2015.
- [26] R. Quirynen, K. Berntorp, K. Kambam, and S. Di Cairano. Integrated obstacle detection and avoidance in motion planning and predictive control of autonomous vehicles. In *2020 American Control Conference (ACC)*, pages 1203–1208, 2020.
- [27] S. Rakhimkul, A. Kim, A. Pazyzbekov, and A. Shintemirov. Autonomous object detection and grasping using deep learning for design of an intelligent assistive robot manipulation system. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3962–3968, 2019.
- [28] M. Rubagotti, T. Taunyazov, B. Omarali, and A. Shintemirov. Semi-autonomous robot teleoperation with obstacle avoidance via model predictive control. *IEEE Robotics and Automation Letters*, 4(3):2746–2753, 2019.
- [29] Joan Sola. Quaternion kinematics for the error-state kalman filter. *arXiv preprint arXiv:1711.02508*, 2017.
- [30] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473, 2017.
- [31] M. Vukob, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl. Auto-generated Algorithms for Nonlinear Model Predictive Control on Long and on Short Horizons. In *Proceedings of the 52nd Conference on Decision and Control (CDC)*, 2013.
- [32] Jiaole Wang. *Novel Strategies for Enhanced Navigation in Robotic Surgical System*. PhD thesis, The Chinese University of Hong Kong, 2012.
- [33] J. S. Yuan. Closed-loop manipulator control using quaternion feedback. *IEEE Journal on Robotics and Automation*, 4(4):434–440, 1988.

- [34] Brayan S Zapata-Impata, Pablo Gil, Jorge Pomares, and Fernando Torres. Fast geometry-based computation of grasping points on three-dimensional point clouds. *International Journal of Advanced Robotic Systems*, 16(1):1729881419831846, 2019.
- [35] A. Zeng, K. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1386–1383, 2017.
- [36] A. Zube. Cartesian nonlinear model predictive control of redundant manipulators considering obstacles. In *2015 IEEE International Conference on Industrial Technology (ICIT)*, pages 137–142, 2015.