CHALMERS
UNIVERSITY OF TECHNOLOGY

Myoelectric control within the framework of Hierarchical Quadratic Programming

Systems, control and mechatronics

Akos Vass

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis

Myoelectric control within the framework of Hierarchical Quadratic Programming

Akos Vass

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

Myoelectric control within the framework of Hierarchical Quadratic Programming
AKOS VASS

Supervisor: Rita Laezza, Department of Electrical Engineering
Examiner: Yiannis Karayiannidis, Department of Electrical Engineering

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

Cover figure: Thalmic Labs' Myo armband and Universal Robots' UR10 robot; hardware systems used in the thesis.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Myoelectric control within the framework of Hierarchical Quadratic Programming
AKOS VASS
Department of Electric Engineering
Chalmers University of Technology

Abstract

As technical advancements make robotic manipulators cheaper and more widely accessible overall, the quality of human-robot interaction becomes increasingly important. Modern robots are expected to safely function in a wide variety of workspaces together with humans and to offer simple interfaces through which they can be commanded. In this thesis, a kinematic control framework is investigated based on solving a sequence of Quadratic Programs to satisfy a hierarchically ordered set of tasks, known as Hierarchical Quadratic Programming (HQP). A task in this context essentially refers to a system of linear constraints in the optimization problem, which may consist of equalities or inequalities. A method for incorporating myoelectric control signals for open loop control is proposed. In particular, IMU and EMG-based pattern recognition signals are sourced from a Myo armband sensor to manipulate the position and orientation, respectively, of the end-effector of a 6-DoF UR10 robot. The algorithm is implemented in ROS using Python and tested in simulation in five different experiments. The results of the experiments are used to demonstrate the broad feasibility of the system as well as to highlight some of the current shortcomings and propose further improvements for them.

Keywords: robotic control, myoelectric control, hierarchical quadratic programming, task-function, optimisation, kinematics.

Acknowledgements

I would like to thank my examiner Yiannis Karayiannidis and supervisor Rita Laezza for giving me the opportunity to work on this project, and their patience and support.

Akos Vass, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1

Introduction

1.1 General context

The perpetual creation and utilization of ever improving tools can be considered one of the prime movers of human history. Although not as unique to us as originally thought, the multitude of human tools and their sophistication definitely far exceeds those used by any other species. This discrepancy has been especially increasing since the industrial revolution, when much of manual labour was replaced by machines powered by natural resources, leading to an increase in both quantity and quality of the goods produced by society, a trend which is persistent to this day.

Better quality goods include better quality tools for manufacturing. The industrial machines of today are far more complex and powerful compared to the steam-driven spinning mules and lathes of the industrial revolution. On the other hand, their fundamental function within the factory has not changed much; traditional industrial robots follow sets of predetermined instructions given by computers, which they repeat uniformly. The activities of humans and robots are separated in order to ensure safety; the machines do the repetitive parts which would be strenuous and/or dangerous, while people continue to fill in for processes that are hard to automate.

The development of technology continues to open new avenues in human-machine interaction, however. Thanks to advancements in computational capacity, sensor technology and lightweight materials, there is an opportunity for humans to work together with collaborative robots, also called cobots. As mentioned, robots in classical industrial applications have been designed by engineers to carry out specific tasks in a static setting. Since cobots are meant to directly interact with and support humans in a large variety of environments, it is necessary that they are safe to be around and can easily be controlled even by laypersons for the task at hand.

While the development and application of robotics involve highly interdisciplinary studies, ranging from materials science to microchip engineering to logistics, the focus of this thesis is limited to the algorithmic control of industrial manipulators.

The content of this thesis consists of two main parts. The first being the investigation and implementation of an optimization algorithm with hierarchical constraints in the context of robot control, referred to as Hierarchical Quadratic Programming. The

second one is about the use of electromyographic signals provided by a biomedical sensor as a ways in which humans can give high-level commands to robots without technical expertise.

1.2 Relevant work

Expressing different objectives by the means of task-functions goes back to 1990, when Samson and Espiau proposed the system for controlling general mechatronic systems in [1]. The relationship of task-functions and redundancy in structures with high degrees of freedom was analyzed in [2] and [3]. The algorithms for prioritizing different tasks in this framework have gradually evolved and become more sophisticated through the years. In [4], prioritization is achieved by combining each task with a weight parameter to express their relative importance, and solving the combined problem in the least squares sense. The problem with this approach is that when different tasks are in conflict, the solution minimizing their weighted ensemble may be a trade-off that satisfies neither objective.

Enforcing strict prioritization by solving a lower-level task within the null-space of its superior was first proposed by [5] and generalized to any number of priority levels in [3]. However, these approaches were still limited to only handling equality tasks. Inequality objectives like joint limits could be included by expressing them in equality form through artificial potential field functions, as proposed in [6], but this workaround would be limited to keeping such tasks at the bottom of the hierarchy.

A generalized solution to prioritizing an arbitrary mixture of equality and inequality tasks was achieved in [7], [8]. The algorithm relies on sequentially solving Quadratic Programs to find the optimal set of solutions of each task, going from highest to lowest priority. Each subsequent task is solved as best as possible within the previously found sets, thus ensuring that the resolution of secondary objectives does not impair more important ones.

A method for varying the priority levels and adding/removing tasks smoothly during operation is proposed in [9]. In [10], a dedicated solver relying on complete orthogonal decomposition is proposed. The approach for enforcing the prioritization is fundamentally similar to [8], but computationally more efficient as it resolves the active set of constraints for the whole stack of tasks in one step, in contrast to dealing with Quadratic Programs of growing complexity in the original algorithm.

A comprehensive review of the state-of-the-art in collaborative human-robot interfaces can be found in [11]. At the time of writing, the authors conclude that while the adaptation of collaborative systems in various industrial and medical settings is becoming increasingly widespread, the demand for intuitive and reliable interfaces for communication between humans and robots remains high.

A comprehensive review of the modern applications and standard practices for surface electromyography can be found in [12] and [13]. In [14] a Support Vector Machine is used to classify user wrist gestures into up to 19 categories based on

estimated direction and torque applied, and control a 2-DoF rehabilitative wrist exoskeleton prototype. In [15] an impedance control method based on estimating user joint torques from electromyographic (EMG) signals using a neurofuzzy matrix modifier is proposed and applied in real-time to a 7-DoF full-arm power-assist exoskeleton. A similar system with minimal user- or task-specific configuration requirements has been proposed more recently in [16], whose authors apply the system to dynamically change an industrial arm manipulator's contribution to cooperative tasks, such as sawing, based on operator fatigue detected from EMG sensors.

A framework for hand gesture recognition using both EMG and accelerometer signals is proposed in [17]. The authors combine fuzzy K-means clustering and multistream hidden Markov models and demonstrate its efficacy by achieving 93% word accuracy in recognizing Chinese Sign Language, and 97% accuracy when distinguishing between 18 gestures for manipulating a virtual Rubik's cube in real time.

The particular sensor used in this thesis was Thalmic Labs' Myo Armband, whose technical specifications are presented in depth in [18]. The authors of [19] have used the same device for developing a gesture classifier using Convolutional Neural Networks based on frequency-domain features, and evaluated it in part by guiding a 6-DoF JACO industrial manipulator.

1.3 Research questions and contributions

The HQP algorithm presented and implemented in this thesis largely follows the formulation in [7] and [8]. The contribution of this thesis is the incorporation of myoelectric signals into the framework of HQP with the aim of developing an integrated collaborative system that is compatible with any (redundant) industrial manipulator. This is done by answering the following research questions:

1. How to formulate task-functions necessary for operating robots in a shared workspace with humans, in a way that is applicable to different robot models in various scenarios in the framework of HQP?

2. How can operator gestures classified from myoelectric signals best be used for online guidance of robots within the framework of HQP, in terms of execution speed and accuracy?

1.4 Thesis outline

Chapter 1 has introduced the general direction of the thesis, accounted for relevant works in the area, and defined its objectives. The rest of the report is structured as follows.

Chapter 2 presents the theoretical background underlying the work done in this study, including the modeling of robotic manipulators and their control by defining task-functions on the kinematic level. The definition of Quadratic Programs and

their solution via the active-set method is reviewed as a general framework for constrained optimization, and then put in context of robotic control in the form of the HQP algorithm. The basics of electromyography in the control context are also summarized.

Chapter 3 details the application of the previously mentioned methods on a UR10 robot. The implemented formulations of joint limit, trajectory tracking, obstacle avoidance tasks are presented, as well as the design of an open-loop control schema using EMG and IMU signals from the Myo Armband for guiding the manipulator.

Chapter 4 contains in total five experiments carried out to demonstrate and evaluate the efficacy of the implementation of the presented task-functions and HQP algorithm in general.

The report is concluded in Chapter 5 with a review of the work done, discussion of the resulting outcomes and suggestions for further development.

2

Theory

This chapter presents the theoretical background relevant to various parts of the work carried out in the thesis. The main topics in it are a broad review of kinematic modelling and control of robotic manipulator, followed by a similar section about Quadratic Programming and convex optimization, and concludes with the use of electromyography in control applications.

2.1 Robot manipulators

Robots are actuated multi-body systems. They consist of rigid links interconnected by joints, forming a kinematic chain. A kinematic chain is said to be open if there is only one sequence of links connecting the two ends of the chain, which is typically the case for industrial robot manipulators, where one end of the chain is fixed at the base, and the other (commonly termed end-effector) is free to move in space. The mobility of a robot is enabled by the actuation of joints, which can be either prismatic (allowing translational motion between the joined links) or revolute (allowing rotational motion between the links). In an open kinematic chain, each joint provides the system with one degree of freedom (DoF). [20]

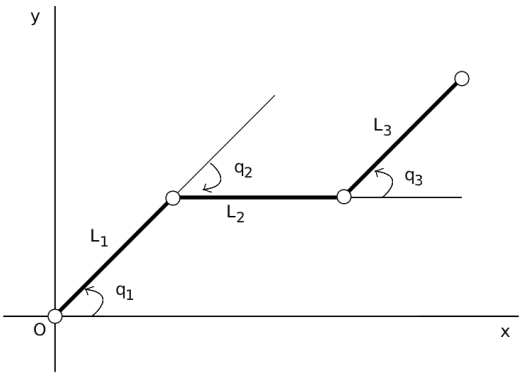


Figure 2.1: A 3-DoF kinematic chain in 2 dimensions

The following subsections present some of the mathematical models and corresponding terminology used for describing the orientation and movement of robotic ma-

nipulators, and how they are used for controlling their actuators in order to execute generic tasks.

2.1.1 Kinematics and differential kinematics

The most elemental problem in robotics modeling is describing the pose (position and orientation) of various parts of the mechanical structure, typically in 3D Cartesian space with respect to the fixed base frame as reference. This can be done using homogeneous transformation matrices, which are mathematical mappings between two frames, consisting of a rotation matrix and a translation vector, expressing the relative orientation and position of the frames, respectively. Knowing the dimensions of the links (L_i in figure 2.1) and the actual states of the variable joints (q_i), it is possible to construct such matrices to represent the pose of various parts of the robotic structure. Using homogeneous transformation matrices, we can write:

$$T_B^A = \begin{bmatrix} & R_B^A & & t_B^A \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

to describe frame B to frame A [20]. The rotation matrix $R \in SO(3)$ is a special orthogonal matrix which accounts for the difference between the orientation of the two frames, and the translation vector $t \in \mathbb{R}^3$ for the difference in position. The advantage of this representation is that the matrices connecting consecutive links in the chain can be multiplied together recursively to derive the description of each part of the manipulator with respect to a common reference frame. This is commonly referred to as forward kinematics. For a robot with n degrees of freedom, the pose of the end effector can be acquired as such:

$$T_n^0 = T_1^0(q_1)T_2^1(q_2)\dots T_n^{n-1}(q_n) \tag{2.2}$$

Homogeneous transformation matrices offer a convenient way to relate the position and orientation of each link in the kinematic chain to the previous one, and to compute more complex relationships by systemic composition of the elementary ones. While the representation and operability of different positions in space is fairly straight forward, this is not the case for orientation. Rotation matrices are just one way to describe orientation; other alternatives worth mentioning are Euler angles and unit quaternions [20] [21] [22]. The mathematical definitions of these representations and their relationships to one another can be found in Appendix A.

The twin problem of forward kinematics, called inverse kinematics, is about finding the joint configuration that correspond to a given end-effector pose. The problem solution is fundamental for controlling a robot through its joints in order to produce the desired end-effector motion. While the derivation of forward kinematics is fairly straight-forward, solving the inverse is much less trivial; there is generally no closed-form solution to the problem, and depending on the structure of the robot and the

specific end-effector pose, there may be infinite, multiple or even no solutions to the problem.

As forward kinematics refers to the relationship between joint states and end-effector pose, differential kinematics is the relationship between joint and end-effector velocities. The matrix expressing this relationship is called the geometric Jacobian $J(q) \in \mathbb{R}^{6\times n}$:

$$v_e = \begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix} = J(q)\dot{q} \tag{2.3}$$

where $\dot{p}_e$ and ω_e refer to the end effector linear and angular velocities, respectively. It can be derived similarly to the direct kinematics equations by recursively computing the contributions of each frame with respect to the previous ones in the chain. Alternatively, one can compute the analytical Jacobian, $J_A(q) \in \mathbb{R}^{6\times n}$, by simply differentiating the forward kinematics function with respect to the joint variables:

$$\dot{x}_e = \begin{bmatrix} \dot{p}_e \\ \dot{\phi}_e \end{bmatrix} = \begin{bmatrix} \frac{\partial p_e}{\partial q} \\ \frac{\partial \phi_e}{\partial q} \end{bmatrix} \dot{q} = J_A(q)\dot{q} \tag{2.4}$$

The two Jacobians are typically not the same, as ω_e represents the angular velocity with respect to the base frame, while $\dot{\phi}_e$ is the time derivative of the parameters used to describe end effector orientation, most commonly Euler angles, without a clear physical interpretation in itself [20] [21].

The equations presented in this section are sufficient for creating an elementary kinematic control law for robotic manipulators, that is to say to generate joint movement in order to send the end-effector along a pre-defined trajectory, or towards a static goal point, referred to as Closed Loop Inverse Kinematics (CLIK), presented in Appendix B.

2.1.2 Task-function based control

A task-function $e(q) \in \mathbb{R}^m$ is a differentiable vector function specifying some property of the robot that is to be controlled to a reference value [7]. This can be written without loss of generality as:

$$e(q) = 0 \tag{2.5}$$

Differentiating this function provides an expression for the evolution of the task with respect to the configuration space of the kinematic structure:

$$\dot{e}(q) = \frac{\partial e}{\partial q}\dot{q} = J_e\dot{q} \tag{2.6}$$

Note that the task Jacobian denoted as J_e here maps the joint velocities to the derivative of the task values (whatever they may represent), whereas the analytical Jacobian in the previous section map to the derivative of the end-effector pose specifically. From the preceding definitions, one can recognize the analytical Jacobian as an example of a task Jacobian, namely to the task defined as the distance of the end-effector pose from a static reference. In what now follows we will show that the control law defined by the CLIK algorithm is equivalent to solving this single task in the least-squares sense.

Consider now imposing the following ordinary differential equation (ODE) as the reference behaviour $\dot{e}^*$ for the task:

$$\dot{e}^*(q) = -\lambda e(q) \tag{2.7}$$

where λ is a positive real constant. Following this dynamic would mean that the value of the task function $e(q)$ decays towards 0 exponentially [23] [7]. The goal of driving the task to its reference value can then be achieved finding the control inputs that minimize the difference between the task velocity $\dot{e}$ and $\dot{e}^*$, which is given by solving the following quadratic optimization problem (QP):

$$\arg\min_{\dot{q}} \quad ||J_e\dot{q} - \dot{e}^*||^2 \tag{2.8}$$

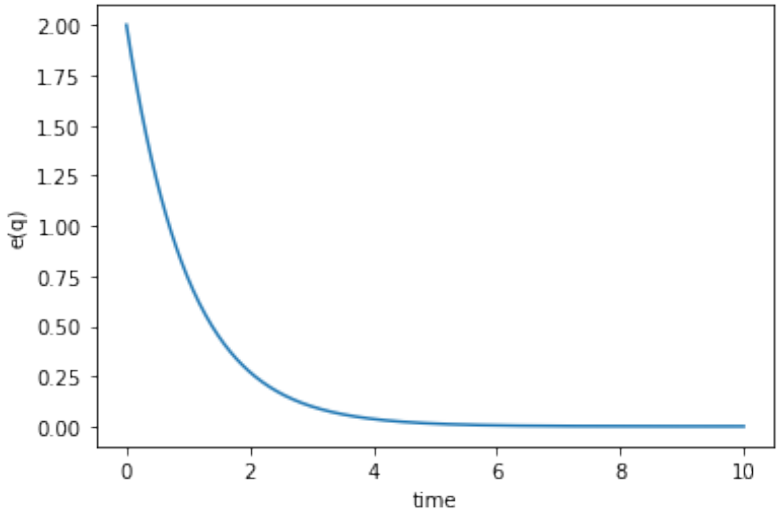


Figure 2.2: Task-function value over time following reference behaviour given in (2.7). $e_0 = 2$, $\lambda = 1$

This can in fact be found as the product of the task Jacobian pseudoinverse and reference velocity:

$$\dot{q}^* = J_e^\dagger \dot{e}^* \tag{2.9}$$

This is due to the algebraic properties of the pseudoinverse. If the number of DoF of the manipulator is insufficient to follow the imposed task dynamic from the current configuration, i.e. the system $J_e\dot{q} = \dot{e}^*$ has no solution, it is said to be singular. In this case, $\dot{q}^*$ returned by (2.9) is the unique vector with the smallest magnitude that minimizes $||J_e\dot{q} - \dot{e}^*||$.

On the other hand, if the number of DoF exceeds the amount necessary to follow the desired dynamic, the system is said to be kinematically redundant. In this case, (2.9) yields the unique vector with the smallest magnitude that satisfies $J_e\dot{q} = \dot{e}^*$. Furthermore, the set of all viable solutions to (2.8) can be expressed as:

$$\dot{q}^* \in \{J_e^\dagger \dot{e}^* + Pz\} \tag{2.10}$$

where $z \in \mathbb{R}^n$ is an arbitrary vector, and $P = I - J_e^\dagger J_e$ is a projector on the null-space of J_e. The main benefit of the task-function approach comes from leveraging the redundancy of the kinematic system expressed by z in order to satisfy multiple hierarchically ordered objectives, by finding solutions to secondary tasks within the null-space of higher priority tasks in a recursive fashion.

In many cases, it is more convenient to define the desired value of some feature as a range rather than as a specific value. An example could be specifying the physical joint limits of the manipulator, or a minimal distance between the links and some external object in the workspace. This can be achieved with inequality tasks (once again, without loss of generality):

$$e(q) \leq 0 \tag{2.11}$$

In general, solving inequality tasks is not as straight-forward as taking the pseudo-inverse of an equality task; they require QP solvers [8][23]. An algorithm for solving a prioritized ensemble of equality and inequality tasks will be presented in section 2.3, as one of the main topics of this thesis.

2.2 Quadratic programming

A Quadratic Program is an optimization problem that can be written in standard form as

$$\begin{aligned} \min_x f(x) = \quad & \frac{1}{2}x^T Q x + p^T x \\ \text{subject to} \quad & Ax = b \\ & Gx \leq h \end{aligned} \tag{2.12}$$

where $x \in \mathbb{R}^n$ denotes the optimization variables[1]. The matrix $Q \in \mathbb{R}^{n \times n}$ and vector $p \in \mathbb{R}^n$ define the objective function to be minimized. The matrix-vector pairs of ($A \in \mathbb{R}^{m_{eq} \times n}$, $b \in \mathbb{R}^{m_{eq}}$) and ($G \in \mathbb{R}^{m_{in} \times n}$, $h \in \mathbb{R}^{m_{in}}$) define m_{eq} equality and m_{in} inequality constraints, respectively. If the constraints are linear and Q is positive semi-definite (an assumption we will be using moving forward), the problem is a convex quadratic program. Convex optimization problems have a number of convenient properties that lend themselves to efficient solution methods; in particular, any local minimum is also the global minimum [24].

2.2.1 Karush–Kuhn–Tucker conditions

In general, the Karush-Kuhn-Tucker (KKT) conditions are a set of necessary conditions for locally optimal solutions of constrained optimization problems, and for convex quadratic programs in particular, they are also sufficient conditions for global optimality [24]. What this means is that solving a QP is equivalent to satisfying the KKT conditions, and as such they are at the base of several numerical optimization algorithms.

The Lagrangian of (2.12) is defined as:

$$\mathcal{L}(x, \lambda, \mu) = \frac{1}{2} x^T Q x + p^T x + \lambda^T (Ax - b) + \mu^T (Gx - h) \tag{2.13}$$

The KKT conditions can then be stated as follows:

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = Qx^* + p + A^T \lambda^* + G^T \mu^* = 0 \tag{2.14}$$

$$Ax^* - b = 0, \quad Gx^* - h \leq 0 \tag{2.15}$$

$$\mu_i^* G_i x^* = 0, \quad \mu_i^* \geq 0, \quad i = 1, ..., m_{in} \tag{2.16}$$

Where x^* is an optimal point at which $f(x)$ is at a minimum (within the specified constraints). The vectors λ and μ are Lagrange multipliers (also called dual variables) associated with the equality and inequality constraints of the problem, respectively. If the constraints are linearly independent, the optimal dual variables λ^*, μ^* corresponding to the primal vector x^* are unique. The values of μ^* can furthermore give information about the state of inequality constraints at x^*; if a constraint is inactive ($G_i x^* < h_i$), the associated multiplier $\mu_i = 0$, and if it is active ($G_i x^* = h_i$), then $\mu_i > 0$. This corollary will be utilized in the algorithm presented in the next section for solving QP problems with inequality constraints.

[1] In the previous section, x has been used to denote the end-effector position, and n the degrees of freedom of the kinematic structure. This is no longer the case here, as the concepts presented in this section are general to constrained optimization and not necessarily linked to applications in robotics.

2.2.2 Active-set method

As mentioned, the KKT conditions can be leveraged to find numerical solutions to optimization problems. For a QP with only equality constraints, the conditions can be written as a system of linear equations, whose solution yields the optimal primal and dual variables [24]:

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix} \tag{2.17}$$

Problems with inequality constraints require a bit more work, but can be solved along the same lines by observing that active inequality constraints are functionally equivalent to equality constraints, while inactive ones are inconsequential. Therefore, if we know the optimal active set, the solution is no different than for the equality-only case in (2.17). This leads us to the problem of finding the set of active constraints at the optimal solution.

Let $\mathbb{A}$ denote the set of active inequality indices at x, and $\bar{\mathbb{A}}$ its complement set:

$$\mathbb{A}(x) = \{i : G_i^T x = h_i\} \tag{2.18}$$

and substitute $x^* = x - c$ to get the linear system to solve for the iterative algorithm:

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} -c \\ \lambda^* \end{bmatrix} = \begin{bmatrix} p + Gx \\ Ax - h \end{bmatrix} \tag{2.19}$$

The algorithm to find the optimal active set can be summarized as follows [25]:

1. Start with some initial guess of $\mathbb{A}$ and a feasible point x_k with respect to it.
2. Solve the system (2.19) for $c = x - x_k$ where x is the solution to the main problem and c is the desired step.
3. Check the feasibility of the new point. One of the following three cases may be true.
4. If $c = 0$ and some Lagrangian multiplier(s) is/are < 0, it means that the new point is not strictly at the limit of the previously active constraints, and the solution could be further improved. This can be done by removing the constraint the most negative multiplier from $\mathbb{A}$ and restarting from 2.
5. If $c \neq 0$, compute a step length α and set $x_{k+1} = x_k + \alpha p$. If this step results in the violation of some constraint (i.e. the main system yielding negative Lagrangian multiplier(s) for previously inactive constraint(s)), add them to $\mathbb{A}$. Restart from 2.

6. If $c = 0$ and the Lagrangian multipliers of the active inequality constraints are all ≥ 0, the optimal active set has been found, and solving for x gives the true optimal solution to the main problem.

Other numerical optimization algorithms for inequality-constrained QPs exist, e.g. interior point methods. The main advantage of active-set methods is that the algorithm can return feasible solutions at each iteration. This can be important in real-time applications, when there is no time to find the optimal solution before applying a control update.

2.3 Solving a Stack of Tasks with Hierarchical Quadratic Programming

As mentioned in section 2.1.2, the redundancy of the robotic structure can be utilized in order to generate optimal motion with respect to multiple objectives by iteratively solving each task within the null-space of previous ones, in order of decreasing priority. This section will present the Hierarchical Quadratic Programming algorithm as a framework for doing so.

To recap, we want to control a robotic manipulator by first defining a configuration- and reference-dependent task-functions, which are essentially just systems of linear equalities or inequalities. These task-functions are then differentiated in order to acquire the task Jacobians and their reference dynamics, which serve as constraints in the optimization problem whose solution is the vector of input variables $\dot{q}^*$ to be sent to the manipulator in order to produce the desired behaviour.

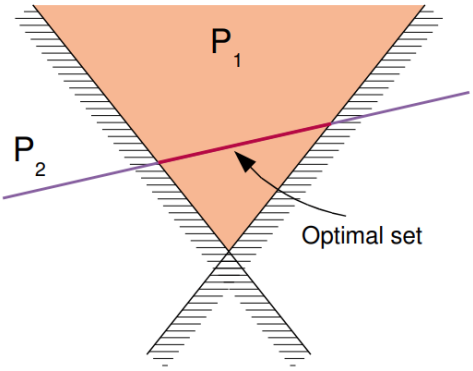
Let us begin by reformulating and expanding the notation for solving hierarchically ordered task-functions using Quadratic Programming, first introduced in (2.8).

We define a stack of tasks as an ordered set of systems of linear equalities or inequalities, given by the aforementioned Jacobian matrices and their reference vectors. At each level of priority $k \in \{1, ..., p\}$, 1 being the most important and p being the least important task, lies a matrix-vector pair formed by A_k, b_k, or G_k, h_k, depending on the type of the task at hand. In order to solve the problem, we iterate through the stack in order of decreasing hierarchy, i.e. increasing k. At each level, we solve the problem (2.20):

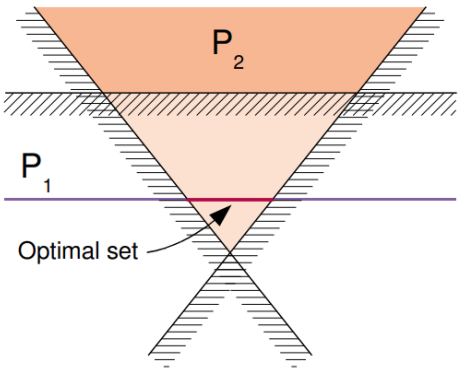
$$\begin{aligned} \underset{\dot{q},\, w_k}{\arg\min} \quad & \frac{1}{2}\dot{q}^T\dot{q} + \frac{\kappa}{2}w_k^T w_k \\ \text{s. t.} \quad & \bar{A}_k\dot{q} - w_k = \bar{b}_k \\ & \bar{G}_k\dot{q} - w_k \leq \bar{d}_k \end{aligned} \tag{2.20}$$

Here the matrices $\bar{A}_k$, $\bar{b}_k$, $\bar{G}_k$, $\bar{h}_k$ have been introduced, as well as the slack variable vector w_k. Together, they form a convex solution set S_k for the problem. This set, or equivalently these matrices is / are built iteratively, by first initializing $S_0 \in \mathbb{R}^n$,

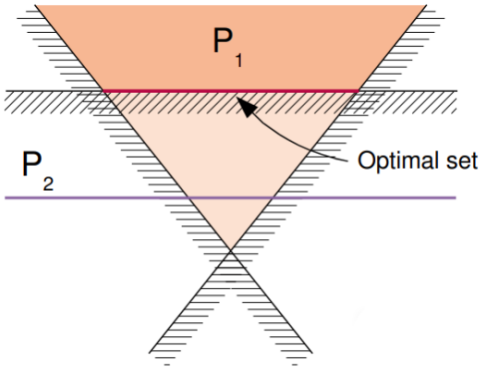
i.e. no constraints, and then appending each task's linear system as the algorithm moves its way through the stack in order of increasing k. The role of the slack variable vector $w_k \in \mathbb{R}^{m_k}$ is to relax the imposed constraints if necessary, and to serve as a measure on their violation.



(a) The two systems are compatible and the optimal solution is simply their intersection; priority does not matter



(b) Equality has higher priority than inequality. The optimal solution satisfies the equality and all possible inequalities while minimizing the distance from unfeasible ones.



(c) Inequality has higher priority than equality. The optimal solution satisfies all inequalities while minimizing the distance from the equality's solution set.

Figure 2.3: Illustration of the resulting solutions for a prioritization problem with an equality and inequality task. [7]

Figure 2.3 gives a visual representation of the prioritization mechanism of the HQP algorithm in 2 dimensions and with 2 tasks.

The way this is achieved is by taking the optimal vector of slack variables w_k^* after having solved each iteration of the problem and using them (now as a locked constant) to build S_{k+1}. By doing so, the optimal solution of input variables $\dot{q}_k^*$ after each level will be as good as they can be (in the least-squares sense) without violating the higher priority tasks in the hierarchy further than they previously were,

before including the latest/less important task into the QP. The way this is done is shown in pseudoalgorithm form in Algorithm 1 [7]:

Algorithm 1: Hierarchical Quadratic Programming

Result: Joint velocities $\dot{q}$ as control input at time t

Input: Stack of tasks in decreasing priority: J_k, e_k^* for $k = 1, ..., p$
current robot configuration: q

Let A_i, b_i denote the equality and G_i, h_i the inequality constraint systems for the corresponding tasks. Initialize the working constraint set $\bar{A}_0$, $\bar{b}_0$, $\bar{G}_0$, $\bar{h}_0$ as empty.

for $k = 1$ **to** p **do**

Solve the QP in the form of (2.20) using the working set, and then update it accordingly:

$$\bar{A}_{k+1} \leftarrow \begin{bmatrix} \bar{A}_k \\ A_k \end{bmatrix}, \bar{b}_{k+1} \leftarrow \begin{bmatrix} \bar{b}_k \\ A_k \dot{q}_k^* \end{bmatrix}$$

$$\bar{G}_{k+1} \leftarrow \begin{bmatrix} \bar{G}_k \\ G_k \end{bmatrix}, \bar{h}_{k+1} \leftarrow \begin{bmatrix} \bar{h}_k \\ G_k \dot{q}_k^* \end{bmatrix}$$

end

After solving the final QP including all tasks in the stack, send the solution $\dot{q}^*$ as the control input to the manipulator and restart for the next timestep.

By going through the stack one task at a time and performing these matrix updates, the algorithm ensures that the hierarchy between tasks is respected, i.e. the solution set for each task is within the null-space of the previous ones [8]. Note that $\bar{b}_{k+1} \leftarrow A_k \dot{q}_k^*$ and $\bar{h}_{k+1} \leftarrow G_k \dot{q}_k^*$ is equivalent to $\bar{b}_k + w_k^*$ and $\bar{h}_k + w_k^*$, respectively. Since the problem is convex, as long as it is feasible, one can ensure that the subsequent tasks are solved within the null-space of higher priority ones − see (2.10) − by storing the slack variables after each stage in the stack.

Lastly, the parameter κ in (2.20) is worth noting. Theoretically, the solution set S_k is always a non-empty convex polytope, i.e. a feasible solution should always exist. In practice, however, numerical instability can occur when the constraints are ill-conditioned [8]. As such, the parameter κ balances the algorithm's stability with the optimality of the resulting solutions; higher κ reduces the error and induces instability, and vice versa.

2.4 Electromyography

Electromyography (EMG) is the measurement of electrical activity in muscles, produced in response to impulse signals from the associated motor neurons [26]. Its main application has historically been monitoring neuromascular abnormalities and identifying related diseases by inserting electrodes in the form of fine needles directly into the muscles. Alternatively, the measurements can be taken by placing electrodes on the surface of the skin over the muscles. The drawbacks of this approach include inability to access deep tissue, increased difficulty to distinguish between activity

from distinct muscles (cross-talk), as well as extra susceptibility to measurement noise; hair and fat between the electrodes and target muscles, changes in body temperature, and slight displacements of the sensors can all affect the recordings [27].

Nevertheless, thanks to advances in sensor hardware and the non-intrusiveness of the technique, surface EMG has been a widely used source of input for biomedical control systems, with applications including assistive robots, wheelchairs, and dexterous prosthetics [13]. Due to the aforementioned noise present in surface EMG measurements, the raw data requires significant pre-processing to generate robust, actionable signals as input to a given system.

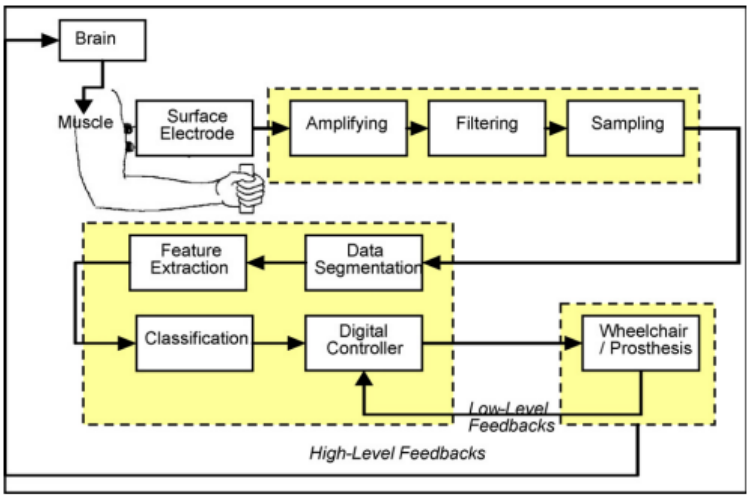


Figure 2.4: Schematic diagram of pattern based myoelectric control system [13]

A single channel of raw EMG signal is collected by one pair of surface electrodes on the subject's body, reflecting the difference in electric potential between the two, given at the sampling rate of the instrument after A/D conversion. This data is then segmented into separate time windows, typically containing 25-200 sample points per channel.

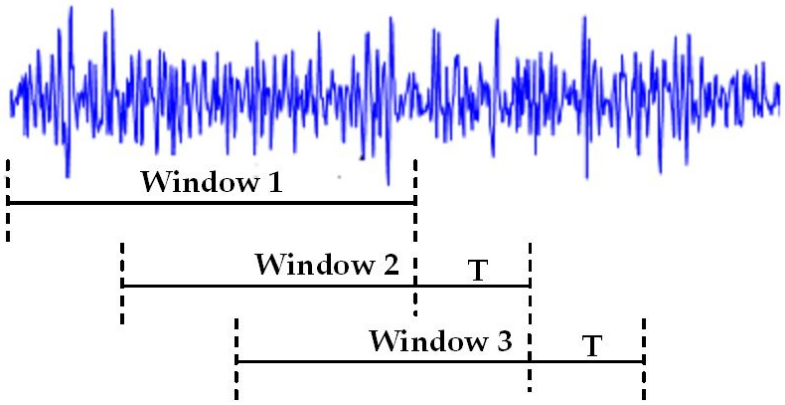


Figure 2.5: Segmentation of EMG signal into overlapping time windows [26]

Each time segment is then used to extract a set of features to give a more compact representation of signal activity in that window. Features can either stem from properties in the time-domain, e.g. mean absolute value, variance, zero crossing; or frequency-domain such as Fourier transform, mean frequency, and spectral density [12].

The feature vectors are finally mapped to a set of distinct gestures performed by the user and corresponding control signals by a classifier.

3

Method

This chapter details how the methods described in the previous section were applied in this thesis work. It begins with a broad overview of the implemented software framework, followed by the derivation of first a number of closed-loop task-functions, then task-functions for open-loop control using the Myo armband.

3.1 General software framework

The algorithm described in section 2.3 was implemented using Python within the framework of Robot Operating System (ROS). ROS is a widely used collection of modeling tools, control algorithms and other pieces of software for robotic development. It allows the integration and reusability of algorithms in a variety of robotic applications.

The controller was tested on and written mainly for an UR10 robot, but with the goal of easy adaptation for other systems with ROS-compatibility. Furthermore, the code was written with the goal of ease of interpretability in mind, rather than computational efficiency. The algorithm based on the solution of expanding QPs is known to be inefficient, but the controller was still easily able to function on a consumer-grade laptop for a 6-DoF robot with any combination of tasks presented in this chapter at a sampling frequency of 25Hz.

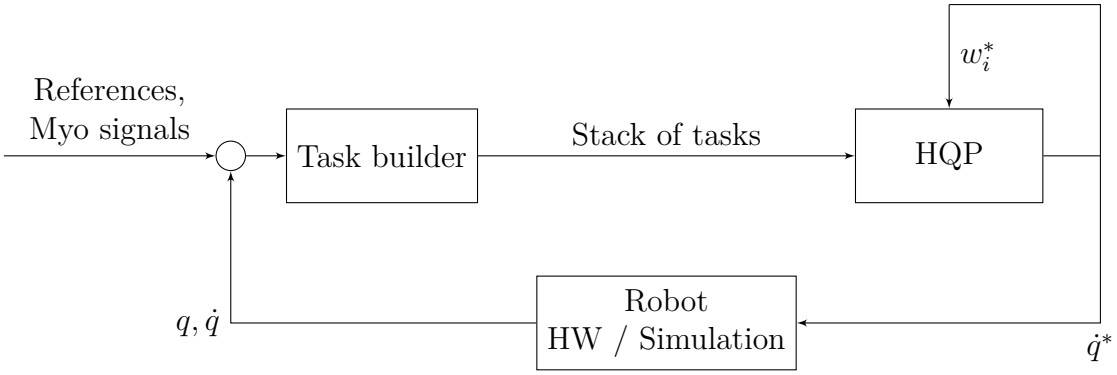


Figure 3.1: High-level diagram of the implemented software architecture.

The schema of the implemented control software is shown in figure 3.1. As shown in section 2.3, the tasks considered by the HQP algorithm are defined by their goal values and convergence dynamic. In this work, the convergence dynamic was set to exponential according to (2.7), as it is commonly the choice. As such, only the reference values and priority levels between different tasks need to be set by the user before starting the controller. The numerical representation of the tasks also depends on the current configuration of the robot, e.g. forward kinematics calculations for avoidance purposes need to be updated during run-time.

The HQP block receives the linear systems and their respective priorities, prepared by the task builder block, and progressively solves the stack in decreasing hierarchy as in section 2.3. After solving the final optimization problem including the bottom-most task, the control input $\dot{q}^*$ is sent to the physical or simulated robot, the slack variables are reset, and the process is repeated.

3.2 Closed-loop tasks

3.2.1 Joint limits

Setting bounds on the joints' positions and velocity for the solver to consider is important for a number of reasons. First and foremost, the solution must comply with the actual physical capabilities of the robot; it is often the case for example that joints can only rotate one full revolution from their base state in either direction. Joint velocities are similarly limited by the power of the actuators, as well as being important for safety reasons if the robot is to share the workspace with humans.

With the joint velocities being the control variables, setting bounds on them is fairly trivial:

$$I\dot{q} \leq b_u \tag{3.1}$$
$$-I\dot{q} \leq -b_l \tag{3.2}$$

where I is the n-dimensional identity matrix and b_u, $b_l \in \mathbb{R}^n$ are vectors specifying the upper- and lower- bounds on the velocity of each joint respectively.

With Δt denoting the sampling time of the controller, joint positions $q_t \in \mathbb{R}^n$ at time t can be expressed simply using the joint velocity measurements:

$$q_t = q_{t-1} + \dot{q}\Delta t \tag{3.3}$$

Using this definition, the task-functions in (3.4) can be used to enforce joint position bounds using joint velocities as the control variables in a similar way:

$$\Delta t I \dot{q} \leq b_u - q_{t-1} \tag{3.4}$$
$$-\Delta t I \dot{q} \leq -b_l + q_{t-1} \tag{3.5}$$

3.2.2 Trajectory tracking

The implementation of the trajectory tracking task essentially mirrors the CLIK algorithm described in Algorithm 2, but without considering the orientation of the end-effector, i.e. $x_e \in \mathbb{R}^3$ simply denotes the position of the end-effector in the base frame.

$$e(q) = x_{ref}(t) - x_e(q) \tag{3.6}$$

Differentiating the task-function with respect to time and imposing exponential convergence results in the constraint:

$$J_v \dot{q} = \dot{x}_{ref}(t) + \frac{\lambda}{2}(x_{ref}(t) - x_e(q)) \tag{3.7}$$

where the task Jacobian $J_v \in \mathbb{R}^{3\times n}$ is equivalent to the robot's translational Jacobian, i.e. the upper half of either the geometric or analytical Jacobian, mapping the joint velocities to end-effector velocities, and λ is a positive gain parameter.

It is worth noting that the derivation of reference trajectories for robotic manipulators is an extensive field of study in itself, and was not deeply considered in this thesis.

3.2.3 Obstacle avoidance

Obstacles in the workspace were represented as static spherical objects, defined by their central point c (in the base coordinate frame) and radius r. The goal is to keep the Euclidean distance between the end-effector and the obstacle centre greater than the obstacle radius plus a safety constant. In standard task-equation form:

$$||x_e(q) - c|| \geq r + \epsilon \tag{3.8}$$

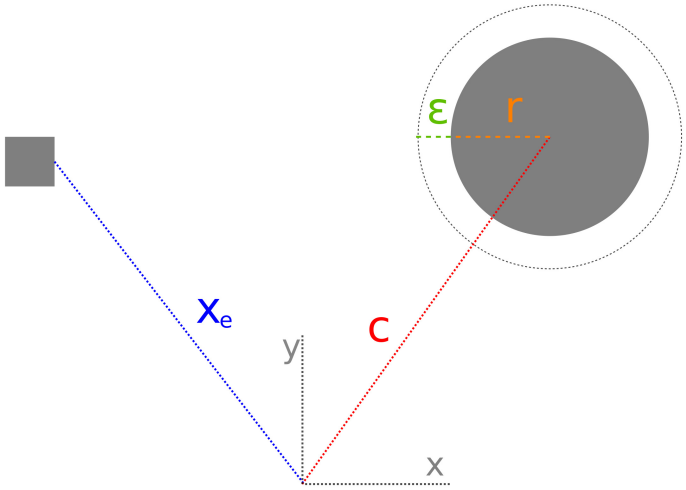


Figure 3.2: 2D representation of the avoidance task. c and x_e are vectors, r and ϵ are scalars.

To make further calculations easier, we can square both sides of the inequality. Reorganizing the terms gives the task-function

$$e(q) = (r + \epsilon)^2 - (x_e(q) - c)^T (x_e(q) - c) \tag{3.9}$$

Differentiation with respect to time gives the task Jacobian:

$$\dot{e}(q) = -2(x_e(q) - c)^T (\dot{x}_e(q) - \dot{c}) = -2(x_e(q) - c)^T J_v \dot{q} \tag{3.10}$$

Where J_v is once again the translational Jacobian. Reorganizing the terms, the final constraint in standard form is written as:

$$-2(x_e(q) - c)^T J_v \dot{q} \leq -\lambda \left((r + \epsilon)^2 - (x_e(q) - c)^T (x_e(q) - c) \right) \tag{3.11}$$

One advantage with this formulation is that the resulting inequality constraint has a dimension of just 1. This is beneficial as the computational cost of the algorithm grows rapidly with the number of inequality constraints involved. The same concept could be applied to more points in the kinematic chain to properly ensure that the whole robot body, and not just the end-effector, stays away from objects in the workspace.

3.2.4 End-effector orientation

As mentioned in Section 2.1.1, there are several ways to represent orientation. While rotation matrices are convenient for deriving forward kinematics functions, they are not as suitable for control purposes because of the high number of parameters they require to express error signals. Euler angles on the other hand are prone to numerical instability; there exist representational singularities where they are not uniquely defined for the given orientation. To this end, unit quaternions were chosen in this thesis to implement the task for orientation control, as they can sufficiently express orientation error with four parameters without the drawbacks of Euler angles.

Let Q_r be the reference orientation and Q_e the actual orientation of the end effector. The quaternion expressing the error between the two orientations can be computed as:

$$\Delta Q = Q_r * Q_e^{-1} \tag{3.12}$$

Since Q and $-Q = \begin{bmatrix} -w & -\phi \end{bmatrix}^T$ can both represent the same orientation, it is necessary to regulate this product by enforcing its scalar element to always be positive for numerical stability. With that in mind, one can recognize that $\Delta\phi = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ if and only if the two orientations are aligned, thus one can emit the scalar element and only use the vector part of the difference quaternion as the task function.

$$e_O = \Delta\phi = w_e(q)\phi_r - w_r\phi_e(q) - \phi_r \times \phi_e(q) \tag{3.13}$$

The final task constraint is similar to that of (3.7):

$$J_O\dot{q} = \omega_r(t) + \frac{\lambda}{2}\Delta\phi(q) \tag{3.14}$$

Where J_O is the lower half of the geometric Jacobian, connecting joint velocities to end-effector angular velocities, and ω_r is the desired angular velocity. If the reference orientation is defined as a time-dependent trajectory, it can be computed from Q_r via (A.20), and in the case of static reference orientation it is obviously 0.

3.3 Tasks using Myoelectric signals for open-loop control

Open-loop control was integrated into the framework using the Myo armband. It is a wearable sensor device developed by Thalmic Labs. It can measure raw EMG activity in forearm muscles in 8 channels at the sampling rate of 200 Hz, and distinguish between 6 gestures through a built-in classifier. It is also equipped with

an inertial measurement unit (IMU) for measuring orientation at the rate of 50 Hz [18].

Figure 3.3: Myo armband and its recognized gestures (resting gesture not included) [28]

Ideally, we would like a scheme that maps the readily available signals from the device to commands altering each of the six degrees of freedom of the end-effector in the workspace. This turned out to be difficult for the following reasons:

1. the recognizable gestures are mutually exclusive binary signals

2. therefore, the ability to bidirectionally change any one of the parameters that describe the pose of the end-effector would require two of them

3. it follows that three translational parameters would exhaust all available gestures, including the neutral rest pose.

The IMU signal on the other hand is continuous, and may be expressed in various forms, just like the orientation of any rigid body; see Appendix A. However, the sensor has a fairly significant innate drift, the magnitude of which varies with the motion of the device during use. The drift is consequential enough to render orientational control as a direct proportional mapping to the IMU values infeasible. Lastly, the EMG classifier is prone to falsely identifying hand gestures when the user is moving their arm around without performing any; a clear source of error when relying on both signals simultaneously.

On grounds of these limitations of the Myo armband, a more limited scheme was developed. The system is structured into two tasks, which will be detailed below. Thus far, tasks have been derived by first defining a desired task-function $e(q)$, and then differentiating it to derive the task Jacobian. Since these open-loop tasks depend not only on q but also on Myo signals, this approach would be futile as there is no guarantee for convergence. Tasks in the following sections simply refer to dif-

ferentiated systems constituting the constraints that produce the desired behaviour in the manipulator, without explicitly defining e.

3.3.1 EMG-based orientational control

The first task maps gestures to orientational commands. In particular, the operator may rotate the end-effector along one axis of the world frame in positive and negative directions by waving in or out, respectively, and along another by either closing or opening their fist.

With J denoting the geometric Jacobian of the manipulator, the task is defined as:

$$J\dot{q} = \begin{cases} \begin{bmatrix} 0 & 0 & 0 & -\alpha & 0 & 0 \end{bmatrix}^T & \text{if gesture = FIST} \\ \begin{bmatrix} 0 & 0 & 0 & \alpha & 0 & 0 \end{bmatrix}^T & \text{if gesture = OPEN} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & \beta & 0 \end{bmatrix}^T & \text{if gesture = WAVE IN} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & -\beta & 0 \end{bmatrix}^T & \text{if gesture = WAVE OUT} \end{cases} \tag{3.15}$$

where α and β are constants affecting the rotational velocity. In this case, the accessible axes of rotation are x and y, but this could be trivially changed to some other combination of two axes, in order to suit the situation.

Notably, the whole Jacobian is used, although the first three rows are constant regardless of the EMG signal. This is to ensure that the end-effector position is unaffected while the user is manipulating the orientation through gestures. Conversely, when this is not the case, the task changes to preserving the current orientation while allowing other tasks to alter the position. With J_ω denoting the lower half of the geometric Jacobian:

$$J_\omega\dot{q} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \quad \text{if gesture = REST or PINCH} \tag{3.16}$$

3.3.2 IMU-based translational control

The second task maps forearm stance to forward-backward translation. "Forward" is defined as the direction of the z axis-of the end-effector frame (as per ROS convention). See figure 3.4. The unit vector expressing this direction numerically can namely be found in the third row of the homogeneous matrix given by solving the forward kinematics problem.

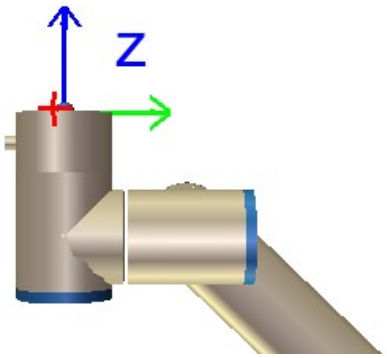


Figure 3.4: End-effector frame axes

In order to express forearm posture, the reference frame of the IMU needs to be calibrated beforehand. With the armband worn on the right arm (status LED bar facing the hand), a neutral IMU quaternion value is calibrated by holding the forearm parallel to the ground, with the palm facing to the left, as shown on figure 3.5. The figure shows the approximate coordinate frame in this stance, but as mentioned, the IMU has a strong drift, and the specific placement of the armband can also have a significant impact on the internal orientation of the IMU axes.

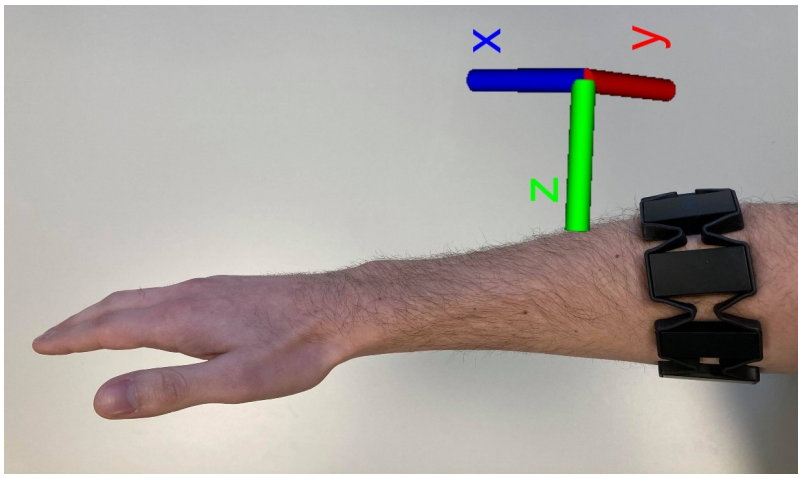


Figure 3.5: Neutral arm stance used for calibration with IMU reference axes

After calibration, the translational commands can be triggered by pointing the arm up- or downwards. Let Q_{mr} be the calibrated IMU reference quaternion and Q_{myo} the IMU quaternion value at current time. The deviation is then measured by their inverse product:

$$\Delta Q = \begin{bmatrix} \Delta Q^w & \Delta Q^x & \Delta Q^y & \Delta Q^z \end{bmatrix}^T = Q_{mr} * Q_{myo}^{-1} \tag{3.17}$$

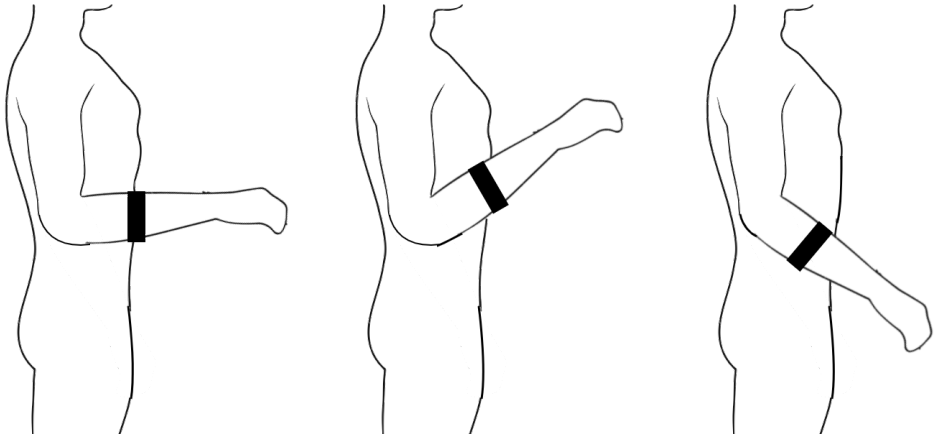


Figure 3.6: 2D sketches showing neutral, forward and backward arm stances

The task itself is similar to the previous one, with J once again denoting the manipulator's geometric Jacobian, and $z \in \mathbb{R}^3$ the forward unit vector:

$$J\dot{q} = \begin{cases} \begin{bmatrix} \lambda z & 0 & 0 & 0 \end{bmatrix}^T & \text{if } \Delta Q^x > 0.15 \\ \begin{bmatrix} -\lambda z & 0 & 0 & 0 \end{bmatrix}^T & \text{if } \Delta Q^x < -0.15 \end{cases} \tag{3.18}$$

The threshold value of 0.15 was chosen heuristically after iterative testing, corresponding to roughly 30° deviation from the calibration pose. As in (3.16), the task is changed to keeping the position, but not the orientation of the end-effector fixed when the user is not actively triggering either command. With J_v denoting the upper half of the Jacobian:

$$J_v\dot{q} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \quad \text{if } -0.15 \leq \Delta Q^x \leq 0.15 \tag{3.19}$$

As mentioned, the pattern recognition model is prone to produce false positives. To alleviate this, the system defaults the EMG task to (3.16) if (3.18) is active, i.e. orientational control is disabled if the forearm is detected to be out of range of the calibration pose.

Prior to arriving at this system, an alternative scheme was considered, with rotation commands mapped to IMU deviation thresholds with respect to each of the three coordinate axes, as it would have enabled arbitrary rotation, but had to be discarded because of the unreliability of the sensor: as the internal frame of the IMU drifts, the calibrated reference position shifts around, and the movements that initially correspond to (almost) elemental rotations cease to do so.

The proposed system should still allow the operator to guide the end-effector to any position, but not pose, in the workspace, by appropriately sequencing rotational and translational commands. The limitations of the system are further discussed after the relevant experiments in the next chapter.

4

Results

This section presents the experiments carried out to demonstrate and evaluate the implementation of the tasks presented in the previous chapter, and the HQP algorithm described in 2.3 for controlling a UR10 robot. The application of the controller on a physical manipulator in real life was not possible because of complications due to the Covid-19 pandemic during the writing of this thesis, as originally intended. The tests were instead performed in URSim, a virtual simulation environment for Universal Robots' manipulators.

4.1 UR10

The UR10 robot is an industrial manipulator developed by Universal Robots. It weights 33.5 kilograms, has a maximum reach of 1.3 metres and a maximum payload of 10 kilograms. It has 6 degrees of freedom through revolute joints, and supports various end-effector accessories.

Figure 4.1: UR10 industrial robot

4.2 Scenario 1

The first scenario consists of following a simple straight trajectory between the points $\begin{bmatrix}0.5 & -0.5 & 0.2\end{bmatrix}^T$ and $\begin{bmatrix}0.3 & 0.3 & 0.4\end{bmatrix}^T$ in the span of 16 seconds or 400 sampling periods, with a constant velocity profile. More explicitly, the reference vectors at timestep i are defined as:

$$x_{ref,i} = \begin{bmatrix}0.5\\-0.5\\0.2\end{bmatrix} + \frac{i}{400}\begin{bmatrix}0.3-0.5\\0.3+0.5\\0.4-0.2\end{bmatrix} \tag{4.1a}$$

$$\dot{x}_{ref,i} = \frac{1}{400}\begin{bmatrix}0.3-0.5\\0.3+0.5\\0.4-0.2\end{bmatrix} \tag{4.1b}$$

At the same time, the robot is under joint velocity and position inequality constraints. The joint bounds were set more stringently than practically necessary for safety purposes to demonstrate the behaviour of the controller.

Priority	Task name	Parameters
1	Velocity limit, upper	$b_u = \begin{bmatrix}0.14 & 0.14 & 0.14 & 0.14 & 0.14 & 0.14\end{bmatrix}^T$
2	Velocity limit, lower	$b_l = \begin{bmatrix}-0.14 & -0.14 & -0.14 & -0.14 & -0.14 & -0.14\end{bmatrix}^T$
3	Position limit, upper	$b_u = \begin{bmatrix}2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5\end{bmatrix}^T$
4	Position limit, lower	$b_l = \begin{bmatrix}-2.5 & -2.5 & -2.5 & -2.5 & -2.5 & -2.5\end{bmatrix}^T$
5	Trajectory tracking	(4.1), $\lambda = 1.5$

Table 4.1: Scenario 1 Stack of Tasks

The joint position limits are respected throughout the scenario, with joint number 3 extending the furthest in the allowed range. The upper joint velocity limit is observable from timestep 200 onwards for joint number 1, and shortly for joint 5. During this period, the end-effector continues in the designated direction, but is unable to close the gap to the reference in the y-direction during the time span of the simulation.

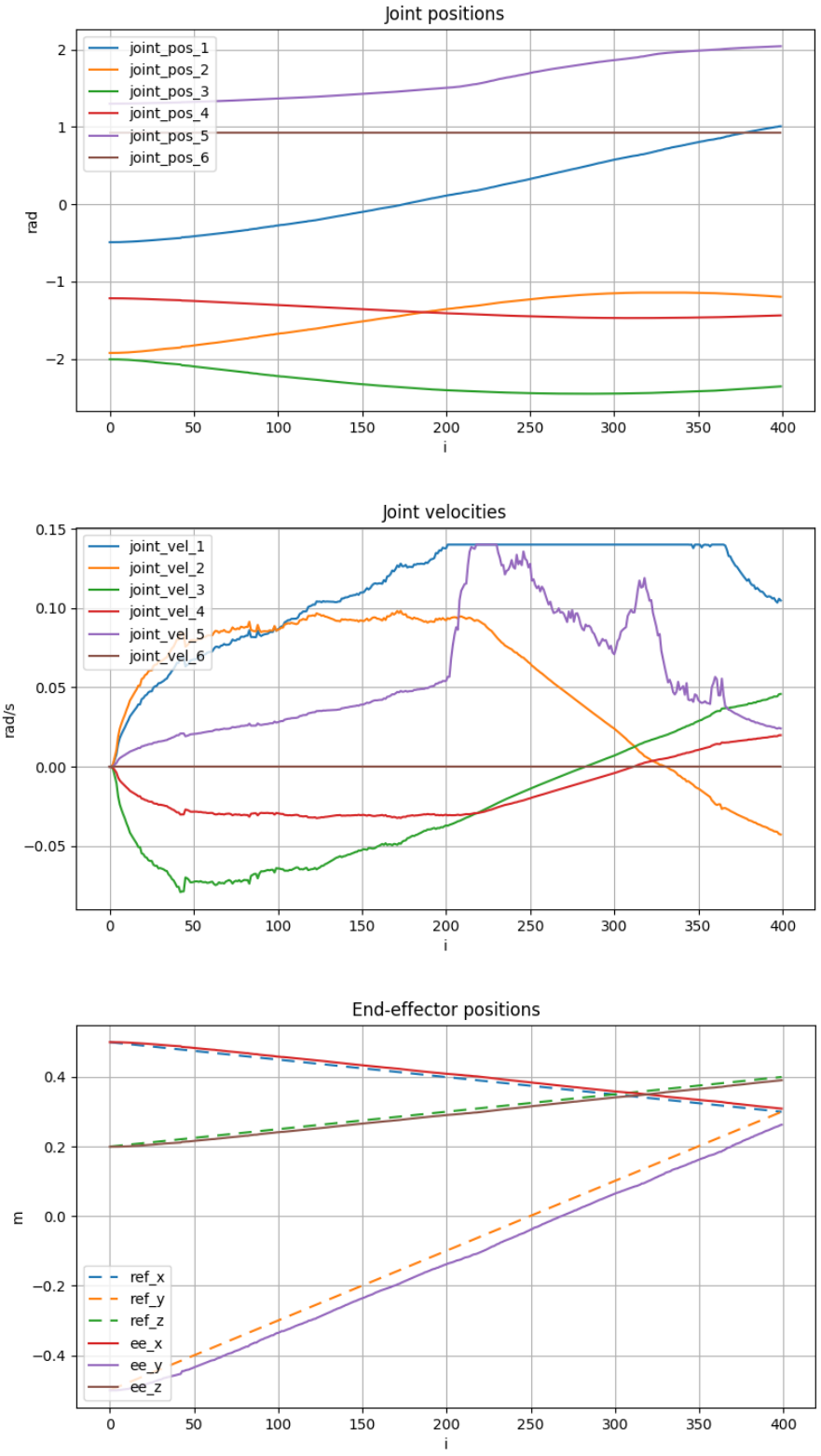


Figure 4.2: Results of scenario 1

4.3 Scenario 2

In the second scenario, the desired path of the end-effector is not defined explicitly in the form of time-dependent reference variables. Instead, the trajectory tracking task is used to attract the end-effector towards a constant goal point:

$$x_{ref,i} = \begin{bmatrix} 0.75 \\ 0.5 \\ 0.2 \end{bmatrix}, \quad \dot{x}_{ref,i} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \forall i \tag{4.2}$$

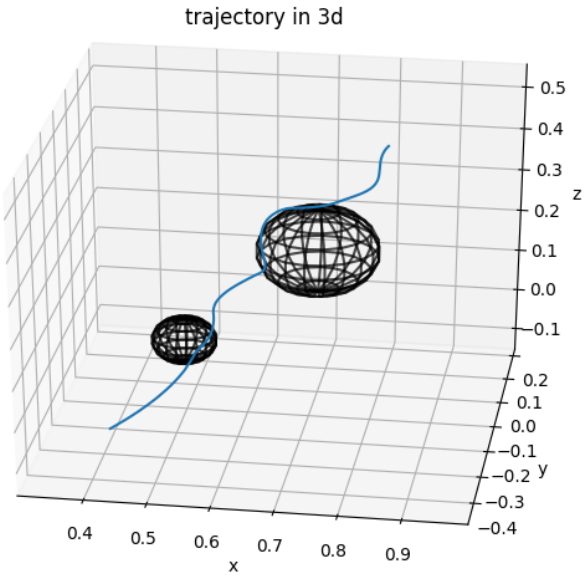
Similarly, the quaternion $\begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}^T$ is used as a target orientation. In the initial configuration of the robot, the end-effector pose is:

$$x_0 = \begin{bmatrix} 0.5 \\ -0.75 \\ 0.2 \end{bmatrix}, \quad Q_0 = \begin{bmatrix} 0.45 \\ 0 \\ 0.89 \\ 0 \end{bmatrix} \tag{4.3}$$

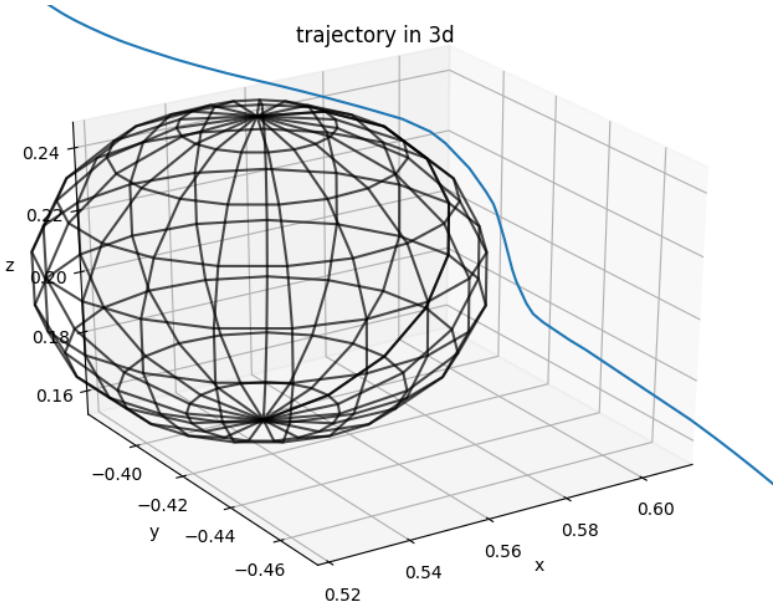
Between the start and goal points are two obstacles to be avoided, all while conforming to joint position and velocity limit (which are more realistic this time, compared to the previous scenario):

Priority	Task name	Parameters
1	Velocity limit, upper	$b_u = \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{bmatrix}^T$
2	Velocity limit, lower	$b_l = \begin{bmatrix} -0.3 & -0.3 & -0.3 & -0.3 & -0.3 & -0.3 \end{bmatrix}^T$
3	Position limit, upper	$b_u = \begin{bmatrix} 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}^T$
4	Position limit, lower	$b_l = \begin{bmatrix} -3 & -3 & -3 & -3 & -3 & -3 \end{bmatrix}^T$
5	Obstacle avoidance 1	$c = \begin{bmatrix} 0.55 & -0.4 & 0.2 \end{bmatrix}^T \quad r = 0.05$
6	Obstacle avoidance 2	$c = \begin{bmatrix} 0.7 & 0.0 & 0.2 \end{bmatrix}^T \quad r = 0.1$
7	Trajectory tracking	(4.2), $\lambda = 0.8$
8	End-effector orientation	$Q_r = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}^T \quad \lambda = 0.8$

Table 4.2: Scenario 2 Stack of Tasks



(a) Birds-eye view of the path taken by the end-effector.



(b) Closeup of the path taken around the second obstacle.

Figure 4.3: 3D plots of the end-effector path in scenario 2.

The effects of approaching each obstacle, and the activation of the corresponding tasks are observable by the abrupt changes in the joint velocity profile at timesteps 34 and 84, accompanied by temporary divergences from the goal pose for the end-effector. After the second obstacle is passed around timestep 115, the end-effector smoothly locks to its setpoints; orientation takes slightly longer to converge to than position as it has a lower priority in the stack (observe that joints 4 and 6 in the wrist are most active in this phase).

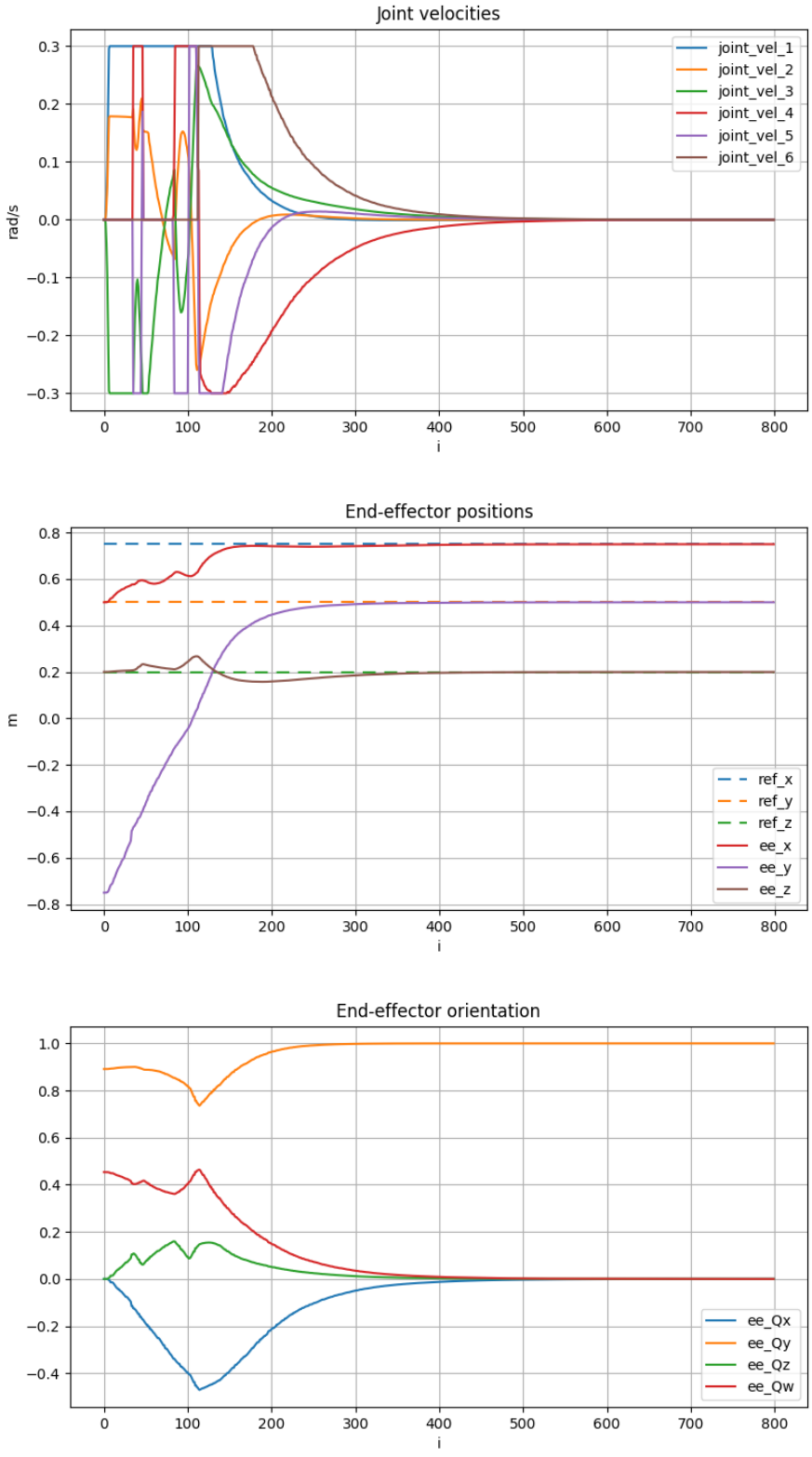


Figure 4.4: Results of scenario 2

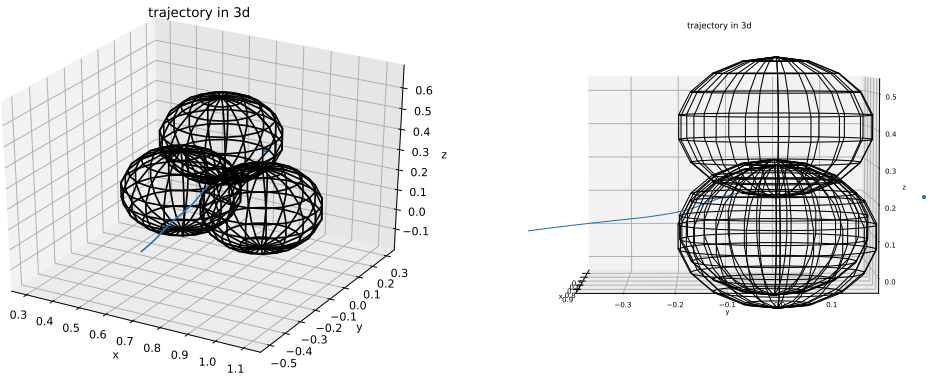
4.4 Scenario 3

The third scenario is similar to the previous one; the end-effector is to move to a reference position without relying on a predefined trajectory while maintaining orientation. There are three closely grouped obstacles in the workspace, collectively forming a concave shape.

Priority	Task name	Parameters
1-4	Joint limits	Same as in 4.2.
5	Obstacle avoidance 1	$c = \begin{bmatrix} 0.6 & 0.1 & 0.1 \end{bmatrix}^T \quad r = 0.2$
6	Obstacle avoidance 2	$c = \begin{bmatrix} 0.9 & 0.1 & 0.1 \end{bmatrix}^T \quad r = 0.2$
7	Obstacle avoidance 3	$c = \begin{bmatrix} 0.75 & 0.1 & 0.3 \end{bmatrix}^T \quad r = 0.2$
8	Trajectory tracking	$x_{ref} = \begin{bmatrix} 0.55 & -0.4 & 0.2 \end{bmatrix}^T \quad \lambda = 0.8$
9	End-effector orientation	$Q_r = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}^T \quad \lambda = 0.8$

Table 4.3: Scenario 3 Stack of Tasks

As the end-effector is attracted towards its goal position by the trajectory task, it ends up at the intersection of the three objects after around 100 timesteps, where it effectively enters a deadlock state. Intuitively, this could be resolved by backtracking and moving around the structure - however, this would require the generated joint velocity inputs to be locally suboptimal: the way to minimize the slack variables associated with the trajectory task is in fact to stay stuck at the very edge of the obstacle structure, as opposed to maneuvering around. This is a demonstrative example of the general issue with local optima that the HQP algorithm is inherently susceptible to. Alternative formulations of the avoidance and trajectory tracking tasks are proposed as potential solutions and discussed further in Section 5.



(a) Birds-eye view of the path taken by the end-effector.

(b) Side-view of the end-effector path.

Figure 4.5: 3D plots of the end-effector path in scenario 3.

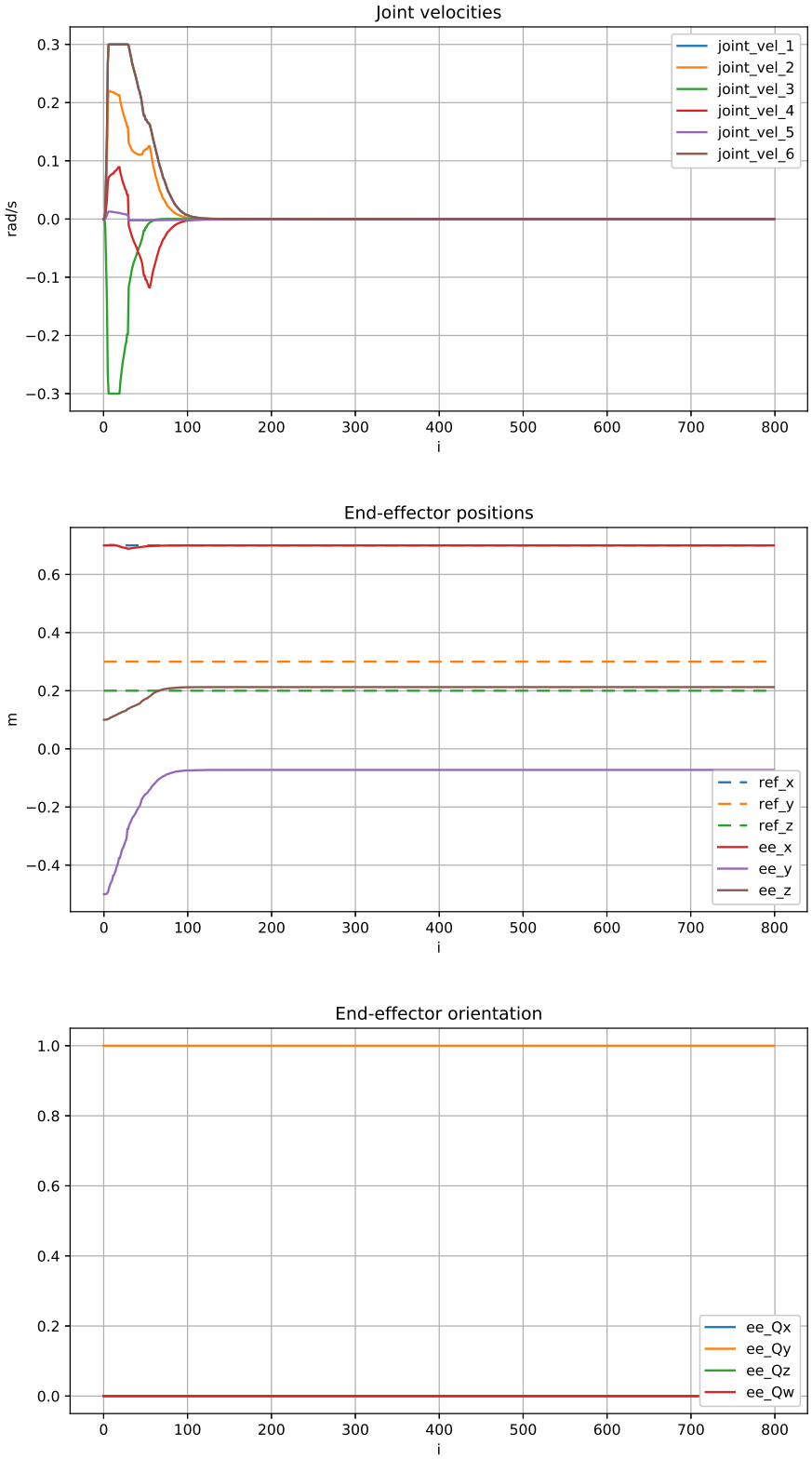


Figure 4.6: Results of scenario 3

4.5 Scenario 4

The fourth scenario is a fairly elementary one with the purpose of demonstrating the capabilities of the open-loop control scheme built around the Myo armband. The first four tasks related to joint limits are the same as previously, while the closed-loop trajectory and orientation tracking tasks are replaced by their open-loop counterparts.

The initial position of the end-effector is at $\begin{bmatrix}0.7 & -0.2 & 1.0\end{bmatrix}^T$, with orientation $\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}^T$. The goal is to move the end-effector down along the z-axis by 60 centimetres with the same orientation to $\begin{bmatrix}0.7 & -0.2 & 0.4\end{bmatrix}^T$; the simulation ends when the Euclidean distance from the target pose is less than 0.05. There is a single obstacle between the start- and goal points.

Priority	Task name	Parameters
1	Velocity limit, upper	$b_u = \begin{bmatrix}0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3\end{bmatrix}^T$
2	Velocity limit, lower	$b_l = \begin{bmatrix}-0.3 & -0.3 & -0.3 & -0.3 & -0.3 & -0.3\end{bmatrix}^T$
3	Position limit, upper	$b_u = \begin{bmatrix}3 & 3 & 3 & 3 & 3 & 3\end{bmatrix}^T$
4	Position limit, lower	$b_l = \begin{bmatrix}-3 & -3 & -3 & -3 & -3 & -3\end{bmatrix}^T$
5	Obstacle avoidance	$c = \begin{bmatrix}0.7 & -0.2 & 0.7\end{bmatrix}^T \quad r = 0.1$
6	Myo gesture recognition	$\alpha = \beta = 0.2$
7	Myo orientation tracking	$\lambda = 0.03$

Table 4.4: Scenario 4 Stack of Tasks

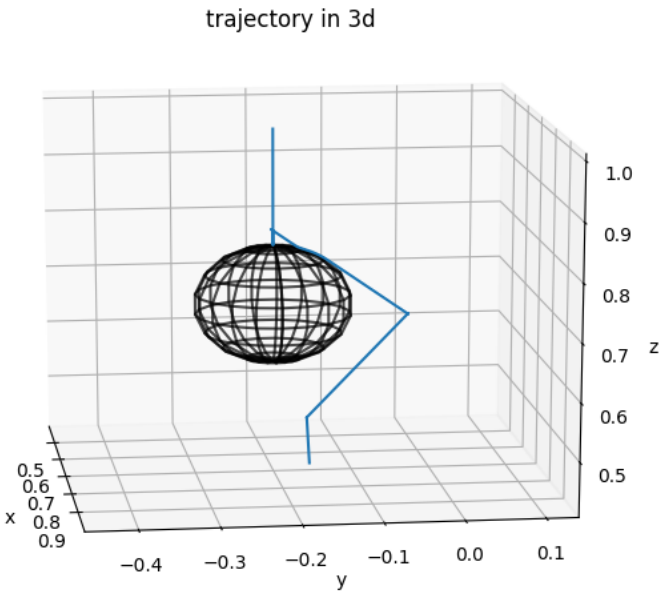


Figure 4.7: 3D plot of the path taken by the end-effector.

The scenario took just over a minute to complete. Interaction with the obstacle can be observed after timestep 200, when the end-effector is automatically stopped as it is directed straight into the object, and shortly after timestep 500 as it moves to its side along its tangent. The first one is due to avoidance, and the second is operator control.

Around timesteps 800-1000, the robot is redirected towards the goal point after having moved to the side of the obstacle. Here a number of short discontinuities can be observed as the armband periodically fails to recognize the gesture that is performed continuously by the user, causing all joint movements to halt momentarily.

Naturally, the scenario could have been completed faster by increasing the α, β, and λ parameters of the open-loop tasks. As it were, the longest phase in the process was the aforementioned reorientation towards the goal at the side of the obstacle, lasting about 13 seconds, with the other similar manoeuvres taking anywhere between 4 to 8 seconds.

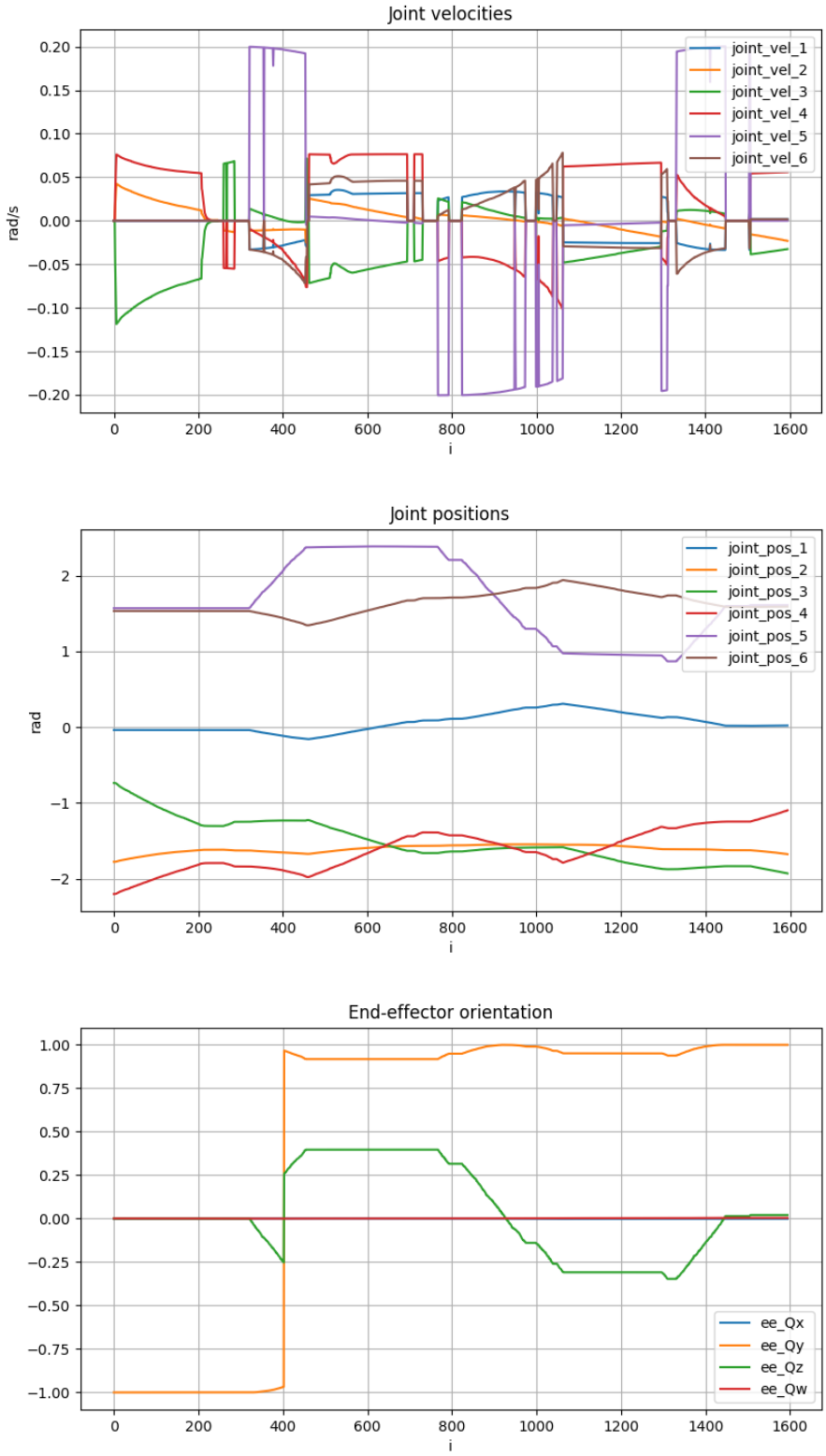
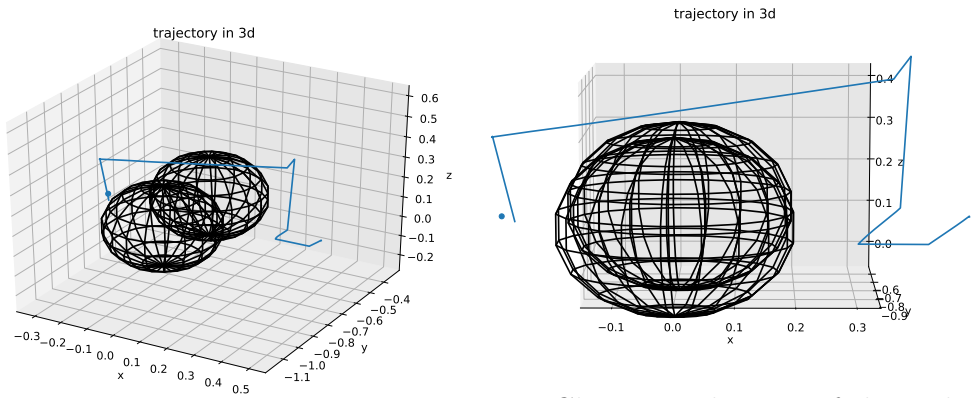


Figure 4.8: Results of scenario 3

4.6 Scenario 5

The fifth presented experiment is a slightly more complicated open-loop scenario than the last. While the setup is intuitively not much more different than the previous one, it took considerably more attempts to successfully complete, thus giving a better highlight of the limitations of the system.



(a) Birds-eye view of the path taken by the end-effector.

(b) Close-up side-view of the end effector path and goal position, market by the dot.

Figure 4.9: 3D plots of the end-effector path in scenario 5.

The starting- and goal-poses of the experiment, written in Cartesian coordinates and quaternions relative the base frame were as such:

$$x_0 = \begin{bmatrix} 0.50 & -0.75 & 0.10 \end{bmatrix}^T \tag{4.4}$$

$$Q_0 = \begin{bmatrix} 0.0 & 0.1 & 0.0 & 0.0 \end{bmatrix}^T \tag{4.5}$$

$$x_n = \begin{bmatrix} -0.30 & -0.75 & 0.1 \end{bmatrix}^T \tag{4.6}$$

$$Q_n = \begin{bmatrix} 0.0 & 0.1 & 0.0 & 0.0 \end{bmatrix}^T \tag{4.7}$$

$$\tag{4.8}$$

Put in words, the end-effector starts facing towards the floor, has to move 80 centimetres in negative x direction, avoiding the obstacles in the way, and finish by resuming the same downward-facing orientation. The experiment ran for a maximum of 2000 samples, or 80 seconds. Also worth noting that the parameters of the open-loop tasks were raised relative the previous scenario, allowing more speed/responsiveness to detected commands.

Priority	Task name	Parameters
1	Velocity limit, upper	$b_u = \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{bmatrix}^T$
2	Velocity limit, lower	$b_l = \begin{bmatrix} -0.3 & -0.3 & -0.3 & -0.3 & -0.3 & -0.3 \end{bmatrix}^T$
3	Position limit, upper	$b_u = \begin{bmatrix} 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}^T$
4	Position limit, lower	$b_l = \begin{bmatrix} -3 & -3 & -3 & -3 & -3 & -3 \end{bmatrix}^T$
5	Obstacle avoidance 1	$c = \begin{bmatrix} 0.0 & -0.6 & 0.1 \end{bmatrix}^T \quad r = 0.2$
6	Obstacle avoidance 2	$c = \begin{bmatrix} 0.0 & -0.9 & 0.1 \end{bmatrix}^T \quad r = 0.2$
6	Myo gesture recognition	$\alpha = \beta = 0.3$
7	Myo orientation tracking	$\lambda = 0.05$

Table 4.5: Scenario 5 Stack of Tasks

Despite over a dozen attempts, the author could not fully meet the objectives of the experiment as in the last scenario. The data presented here is from the closest attempt, where the final position (figure 4.9) and orientation (figure 4.10, last subfigure) are both close to the target, but not within the (arbitrary) limit of 0.05 total Euclidean distance.

Admittedly, the manipulator never came close enough to the obstacles in the simulated workspace to activate the corresponding tasks, but we can see the effects of the velocity limit tasks around timesteps 1500, demonstrating that the lower elements in the stack of tasks are functioning as intended.

An obvious qualitative issue with the system is that the conducted movement is by necessity jerky and fragmented. A key constraint when designing the system was the robustness of the armband signals. As it is, the operator cannot move and change the direction of the end-effector simultaneously, as one would naturally do when using their hands, for example.

To resolve this, one should begin at the signal processing level, and improve the broader control system software stack from the bottom up. The improvement should ideally involve removing all of the built-in implementations by Thalmic Labs, and creating new IMU and EMG filters, as well as a new pattern recognition model. Simply using a different mapping function for the open-loop tasks within the HQP algorithm could not suffice, in the author's opinion; even here, temporary cut-offs in the pattern recognition signals can be observed, for example.

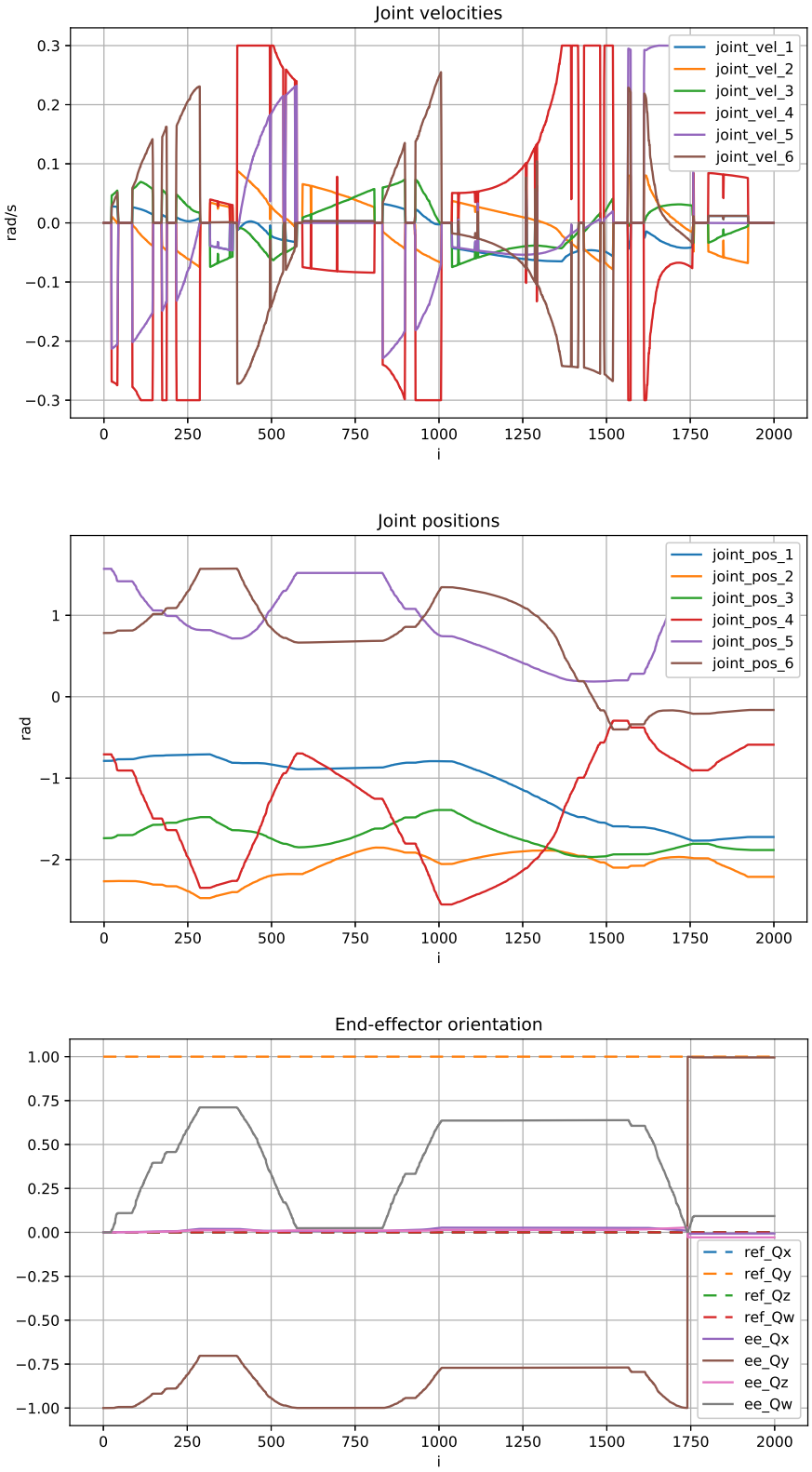


Figure 4.10: Results of scenario 5

4.7 Scenario 6

The last experiment is somewhat of a mixture between the scenarios presented in 4.4 and 4.6. The goal is to move the end-effector from one point to another while maintaining orientation.

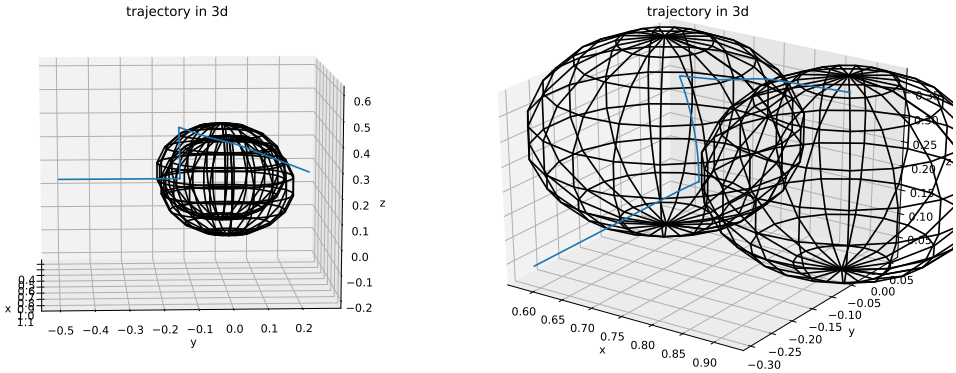
$$x_0 = \begin{bmatrix} 0.75 & -0.5 & 0.2 \end{bmatrix}^T \tag{4.9}$$

$$Q_0 = \begin{bmatrix} 0.0 & 0.1 & 0.0 & 0.0 \end{bmatrix}^T \tag{4.10}$$

$$x_n = \begin{bmatrix} 0.75 & 0.3 & 0.2 \end{bmatrix}^T \tag{4.11}$$

$$Q_n = \begin{bmatrix} 0.0 & 0.1 & 0.0 & 0.0 \end{bmatrix}^T \tag{4.12}$$

Once again, the trajectory task does not have a predefined path, it simply acts as an attractor towards the goal point. There are two closely aligned obstacles in the way of the shortest path, forming a concave trap. This experiment was first conducted in closed loop in reference, and for the most part the system can eventually find a way around them and complete the scenario, but it takes a long time to do so, and in some cases it stays stuck forever; the outcomes vary depending on the exact initial pose (and numerical irregularities), but the whole process would take at minimum half a minute to finish.



(a) Birds-eye view of the path taken by the end-effector.

(b) Close-up view of the end-effector around the obstacles.

Figure 4.11: 3D plots of the end-effector path in scenario 6.

In order to demonstrate the HQP framework's ability to handle a semi-autonomous scenarios, the Myo armband tasks were included in the stack of tasks, presented in Table 4.6.

Priority	Task name	Parameters
1	Velocity limit, upper	$b_u = \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{bmatrix}^T$
2	Velocity limit, lower	$b_l = \begin{bmatrix} -0.3 & -0.3 & -0.3 & -0.3 & -0.3 & -0.3 \end{bmatrix}^T$
3	Position limit, upper	$b_u = \begin{bmatrix} 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}^T$
4	Position limit, lower	$b_l = \begin{bmatrix} -3 & -3 & -3 & -3 & -3 & -3 \end{bmatrix}^T$
5	Obstacle avoidance 1	$c = \begin{bmatrix} 0.0 & -0.6 & 0.1 \end{bmatrix}^T \quad r = 0.2$
6	Obstacle avoidance 2	$c = \begin{bmatrix} 0.0 & -0.9 & 0.1 \end{bmatrix}^T \quad r = 0.2$
6	Myo gesture recognition	$\alpha = \beta = 0.3$
7	Myo orientation tracking	$\lambda = 0.05$
9	Trajectory tracking	$x_{ref} = \begin{bmatrix} 0.75 & 0.3 & 0.2 \end{bmatrix}^T \quad \lambda = 0.3$
10	End-effector orientation	$Q_r = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}^T \quad \lambda = 0.5$

Table 4.6: Scenario 6 Stack of Tasks

This way, the operator could keep their arm in a neutral pose and let the system guide itself without input. When the end-effector arrives at the obstacles and the (closed-loop) system is stuck in a local optima (around timestep 250), the operator temporarily takes over, moving the end-effector "backwards" (in positive z-direction, roughly speaking) using the Myo armband, until it is sufficiently away from the deadlock state (around timestep 550). After that, the operator returns their arm to the neutral stance and lets the trajectory-tracking task complete its course.

While this approach worked, it requires the operator not to divert from the neutral arm pose as to not give open-loop inputs unless necessary. It would be intuitively more elegant to do this with the Myo tasks at the bottom of the hierarchy. With the current implementation of the algorithm, this would not be possible, since any command coming from the Myo tasks that would increase the end-effector position from the target point would be ignored, despite the system being locked in a local optimum.

As a proposal for future work in relation to this, one may implement a way to keep track of the values of (the slack variables of) each task between sampling steps, and remove or move around tasks in the stack when a deadlock state / local optimum error is detected.

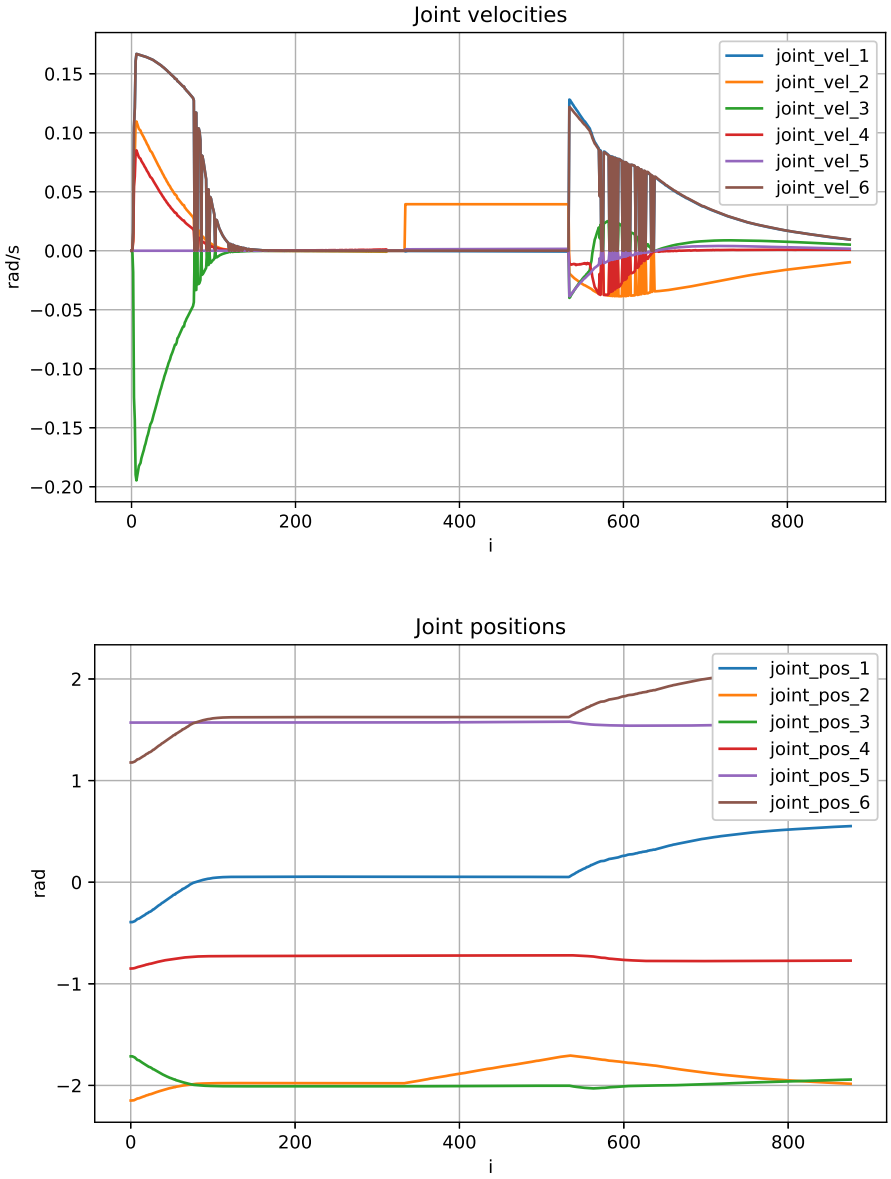


Figure 4.12: Results of scenario 6

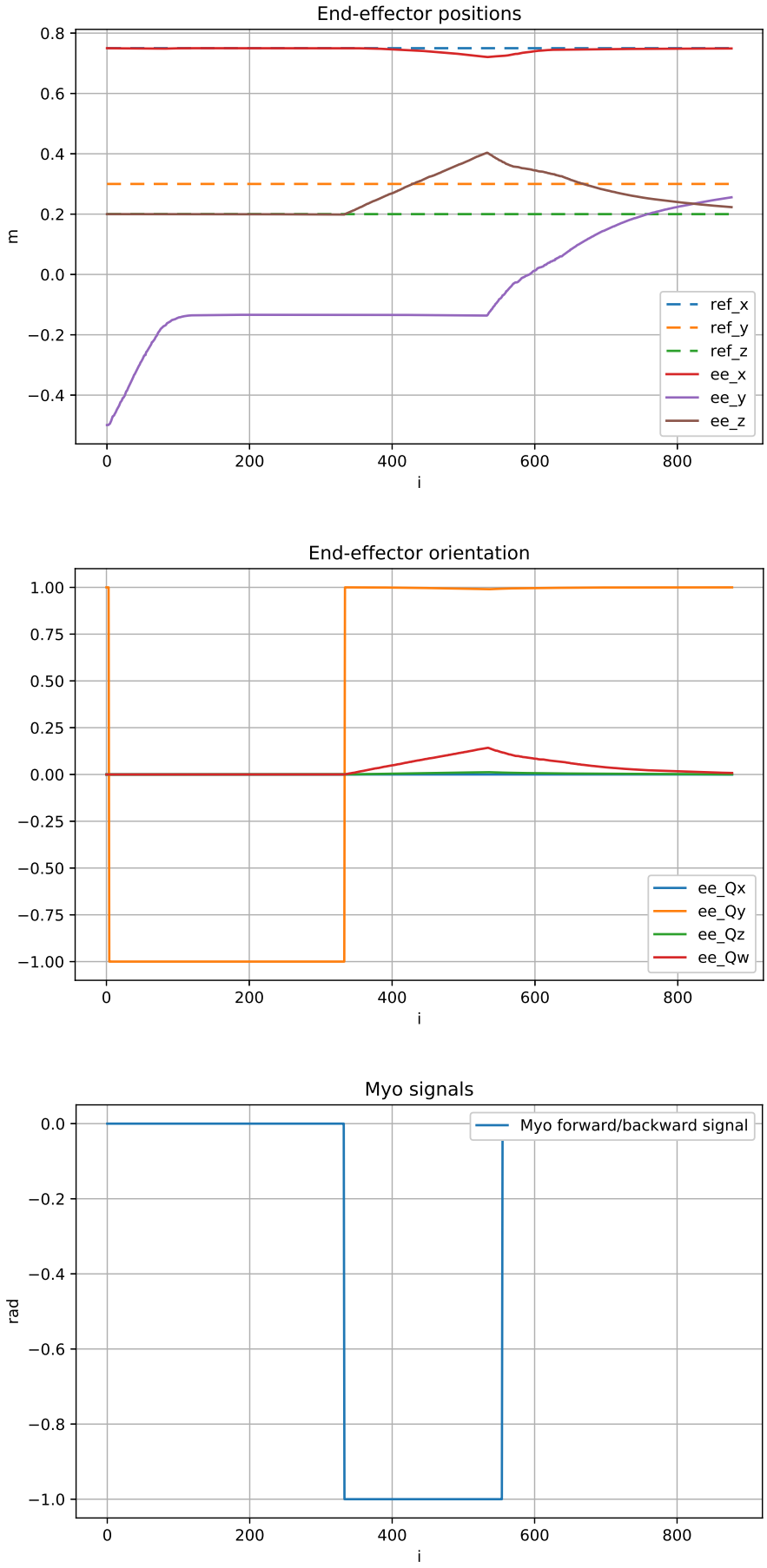


Figure 4.13: Results of scenario 6

5

Conclusion

In this thesis, a HQP controller has been implemented and investigated. The controller returns the optimal joint velocity inputs to satisfy an arbitrary number of linear equality or inequality tasks ordered in a strict hierarchy. The core algorithm and a handful of basic task-functions have been analytically derived and developed in Python within the framework of ROS in a robot-agnostic manner. HQP-compliant task-functions have been implemented to make use of myoelectric signals from a biological sensor that allow users online guidance of robots. The functionality of the implementation has been tested and demonstrated in six scenarios on a simulated UR10 robot.

5.1 Discussion of results

Admittedly, the initial planned scope of the project was wider than what was realized. A number of features never came to be developed due to time constraints, and the tasks that have been presented were only implemented in one viable form; their objective evaluation is therefore difficult as there is no alternative references to measure performance against.

This is especially relevant for the evaluation of electromyographic control scheme. One of the research questions of this thesis was "how can operator gestures classified from myoelectric signals best be used for online guidance of robots within the framework of HQP, in terms of execution speed and accuracy". In order to meaningfully answer this question, different users were meant to perform different scenarios using the Myo armband with different control layouts. Due to the Covid-19 pandemic, setting up physical testing and gathering subjects was not possible; the two last experiments presented in section 4.4 was done by the author using real sensor inputs, but a simulated workspace.

Many of the issues with the current implementation of the open-loop schema stems from the limitations of the Myo armband hardware and built-in firmware, in the author's opinion. These have been detailed in sections 3.3 and 4.6.

Nonetheless, the conducted experiments confirm that the implemented controller is indeed viable and capable of generating motion in real time, accounting for several

constraints simultaneously and enforcing strict hierarchy between them by exploiting the redundancy of the mechanical structure. By placing the velocity limits at the top of the stack hierarchy, the framework is suitable for safely operating robots in a shared environment with humans.

In the next section, some suggestions for potential improvements and extentions still missing are proposed for future research and development.

5.2 Future work

As already already described in section 4.6, the main limiting factors for the current open-loop scheme are mainly related to the out of the box signal quality of the Myo armband, namely:

- a limited number of total distinct gestures for the purpose of robotic control
- occasional loss of signal during persistently maintained gestures from the gesture recognition system
- a strong innate drift in the IMU, rendering proportional and/or continuous use of the signal infeasible

In order to meaningfully alleviate these issues, the preprocessing of the armband's raw signals, and the development of a pattern recognition model that is capable of reliably recognizing a wider range of gestures is suggested. Ideally, one would like to be able to manipulate all six degrees of freedom of the end-effector in a continuous and natural manner, as well as having at least two more input options for e.g. handling grasping commands with a gripper tool.

The obstacle avoidance task could be improved in several ways. The presented formulation could be extended to support moving objects and other geometric primitives than spheres fairly simply. As demonstrated in section 4.4, the HQP is unavoidably susceptible to local optima; in order to reliably navigate crowded workspaces, more sophisticated task-function formulations for obstacle avoidance are necessary. Strategies based on artificial potential fields and dynamic trajectory generation, amongst others, have been subject to extensive research in recent years. Most importantly self-avoidance, i.e. ensuring that different links of the robot do not collide with each other, is a crucial task still missing from this framework.

A previously unmentioned problematic phenomenon, which occurs mainly during open-loop experiments, is the manipulator essentially getting tangled up in itself. This is due to the fact that both the trajectory tracking and the open-loop tasks are defined using the end-effector frame's pose as control parameters, and nothing else. The joint bound tasks make sure that their positions and velocities stay within the given limits, but other than that, the system can find any solution within the optimization space; since the solution is always locally optimal without considering future states, the robot can end up in idiosyncratic configurations as it repeatedly

solves the stack of tasks over time. A simple but effective method to alleviate this, which has not been implemented in this work, is by defining a constant "preferred configuration" task, generally placed at the bottom of the stack, in order to gently pull the manipulator towards a neutral stance whenever possible.

Similarly to collision avoidance, there are several advanced approaches to trajectory generation in literature that could help the stability of the system and make the manipulator motion more graceful.

Finally, there are some suggestions for improving the performance of the HQP algorithm in general. The current implementation of the algorithm uses the Python package quadprog [29] as the underlying numerical solver. A more application-tailored solver, such as the one suggested in [10] could improve the computational efficiency of the algorithm.

Lastly, the current controller implementation has two shortcomings which could be improved upon with slight modifications to the code, without necessarily implementing an entire new numerical solver.

First, the HQP controller currently takes each individually defined task and assigns to them a separate priority level. This is only the case due to ease of implementation and not a technical necessity (in fact, it increases computational complexity without much benefit). Slight modifications in the algorithm code would allow different tasks, regardless of type (equality, greater/lesser than inequalities) to be handled as hierarchically equal.

Second, as already mentioned, is the problem of getting stuck in local optima. This is inherent to any controller based on convex optimization and its solution is nontrivial, but various approaches to alleviating it exist in the literature already which could be incorporated into this work.

Bibliography

[1] C. Samson and B. Espiau. "Application of the Task-function Approach to Sensor-based Control of Robot Manipulators". In: *IFAC Proceedings Volumes* 23 (1990), pp. 269–274.

[2] Y. Nakamura and H. Hanafusa. "Optimal Redundancy Control of Robot Manipulators". In: *The International Journal of Robotics Research* 6 (1987), pp. 32–42.

[3] B. Siciliano and J. -. E. Slotine. "A general framework for managing multiple tasks in highly redundant robotic systems". In: *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*. 1991, 1211–1216 vol.2. DOI: 10.1109/ICAR.1991.240390.

[4] Tan Fung Chan and R. V. Dubey. "A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators". In: *IEEE Transactions on Robotics and Automation* 11.2 (1995), pp. 286–292. DOI: 10.1109/70.370511.

[5] "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 7.12 (1977), pp. 868–871. DOI: 10.1109/TSMC.1977.4309644.

[6] O. Khatib. "The Potential Field Approach And Operational Space Formulation In Robot Control". In: 1986.

[7] Oussama Kanoun. "Contribution à la planification de mouvement pour robots humanoïdes". In: (Oct. 2009).

[8] Oussama Kanoun, Florent Lamiraux, and Pierre-Brice Wieber. "Kinematic Control of Redundant Manipulators: Generalizing the Task-Priority Framework to Inequality Task". In: *Robotics, IEEE Transactions on* 27 (Sept. 2011), pp. 785–792. DOI: 10.1109/TRO.2011.2142450.

[9] S. Kim et al. "Continuous Task Transition Approach for Robot Controller Based on Hierarchical Quadratic Programming". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1603–1610. DOI: 10.1109/LRA.2019.2896769.

[10] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. "Hierarchical quadratic programming: Fast online humanoid-robot motion generation". In: *The International Journal of Robotics Research* 33 (May 2014), pp. 1006–1028. DOI: 10.1177/0278364914521306.

[11] Arash Ajoudani et al. "Progress and prospects of the human–robot collaboration". In: *Autonomous Robots* 42.5 (2018), pp. 957–975.

[12] Maria Hakonen, H. Piitulainen, and A. Visala. “Current state of digital signal processing in myoelectric interfaces and related applications”. In: *Biomed. Signal Process. Control.* 18 (2015), pp. 334–359.

[13] Mohammadreza Oskoei and Huosheng Hu. “Myoelectric Control Systems - A survey”. In: *Biomedical Signal Processing and Control* 2 (Oct. 2007), pp. 275–294. DOI: 10.1016/j.bspc.2007.07.009.

[14] Zeeshan O Khokhar, Zhen G Xiao, and Carlo Menon. “Surface EMG pattern recognition for real-time control of a wrist exoskeleton”. In: *Biomedical engineering online* 9.1 (2010), p. 41.

[15] K. Kiguchi and Y. Hayashi. “An EMG-Based Control for an Upper-Limb Power-Assist Exoskeleton Robot”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.4 (2012), pp. 1064–1071. DOI: 10.1109/TSMCB.2012.2185843.

[16] L. Peternel, N. Tsagarakis, and A. Ajoudani. “A Human–Robot Co-Manipulation Approach Based on Human Sensorimotor Information”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25.7 (2017), pp. 811–822. DOI: 10.1109/TNSRE.2017.2694553.

[17] X. Zhang et al. “A Framework for Hand Gesture Recognition Based on Accelerometer and EMG Sensors”. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 41.6 (2011), pp. 1064–1076. DOI: 10.1109/TSMCA.2011.2116004.

[18] Paolo Visconti et al. “Technical Features and Functionalities of Myo Armband: An Overview on Related Literature and Advanced Applications of Myoelectric Armbands Mainly Focused on Arm Prostheses”. In: *International Journal on Smart Sensing and Intelligent Systems* 11 (June 2018), pp. 1–25. DOI: 10.21307/ijssis-2018-005.

[19] U. Côté Allard et al. “A convolutional neural network for robotic arm guidance using sEMG based frequency-features”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 2464–2470. DOI: 10.1109/IROS.2016.7759384.

[20] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Jan. 2011. ISBN: 978-1-84628-641-4. DOI: 10.1007/978-1-84628-642-1.

[21] James Diebel. “Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors”. In: *Matrix* 58 (Jan. 2006).

[22] Ricardo Campa and Karla Camarillo-Gómez. “Unit Quaternions: A Mathematical Tool for Modeling, Path Planning and Control of Robot Manipulators”. In: Sept. 2008. ISBN: 978-953-7619-06-0. DOI: 10.5772/6197.

[23] Marcus Johansson. “Online Whole-Body Control using Hierarchical Quadratic Programming : Implementation and Evaluation of the HiQP Control Framework”. In: 2016.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

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

[26] Guanglin Li. "Electromyography Pattern-Recognition-Based Control of Powered Multifunctional Upper-Limb Prostheses". In: Aug. 2011. ISBN: 978-953-307-382-8. DOI: 10.5772/22876.

[27] Rubana Chowdhury et al. "Surface Electromyography Signal Processing and Classification Techniques". In: *Sensors (Basel, Switzerland)* 13 (Sept. 2013), pp. 12431–66. DOI: 10.3390/s130912431.

[28] Karen Tatarian et al. "Stepping-stones to Transhumanism: An EMG-controlled Low-cost Prosthetic Hand for Academia". In: Sept. 2018. DOI: 10.1109/IS.2018.8710489.

[29] *quadprog*. URL: https://pypi.org/project/quadprog/.

A

Appendix 1

All formulas taken or derived from [20].

A.1 Rotation matrices

Generic form:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{A.1}$$

Orthogonality property:

$$R^T R = R R^T = I \iff R^{-1} = R^T \tag{A.2}$$

Elementary rotations around the base axes of the reference frame by angle θ:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{A.3}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{A.4}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

Composition of successive rotations with respect to fixed frame:

$$R_{total} = R_2 R_1 \tag{A.6}$$

Relationship of derivative to angular velocity ω with respect to fixed reference frame:

$$\dot{R}(t) = S(\omega)R(t) \tag{A.7}$$

$$S(\omega) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{A.8}$$

A.2 Euler angles

A set of three angles $\phi = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}^T$ represent consecutive elementary rotations. The axes of these rotations can be defined in different ways; the representation must be interpreted in the context of the convention used. Presented here is ZYX or roll-pitch-yaw convention:

Conversion to rotation matrix:

$$R(\phi) = R_z(\alpha)R_y(\beta)R_x(\alpha) \tag{A.9}$$

Conversion from rotation matrix:

$$\alpha = \text{atan2}(r_{21}, r_{11}) \tag{A.10}$$

$$\beta = \text{atan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \tag{A.11}$$

$$\gamma = \text{atan2}(r_{32}, r_{33}) \tag{A.12}$$

The system above degenerates when $\cos(\beta) = 0$; in this case, the Euler angle representation cannot be found analytically as it is not unique (representation singularity).

A.3 Unit quaternions

Generic form

$$Q = \begin{bmatrix} w \\ \phi \end{bmatrix} = \begin{bmatrix} w & x & y & z \end{bmatrix}^T \tag{A.13}$$

Unitary property:

$$w^2 + x^2 + y^2 + z^2 = 1 \tag{A.14}$$

There are different mathematically equivalent formulas to convert (A.1) to (A.13), which can help with numerical stability when one conversion would involve a denominator close to 0. One of them is presented here, assuming $w \geq 0$:

$$\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \pm \frac{1}{2\sqrt{r_{11} + r_{22} + r_{33} + 1}} \begin{bmatrix} r_{11} + r_{22} + r_{33} + 1 \\ r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \tag{A.15}$$

Conversion to rotation matrix

$$R(w,\phi) = \left(w^2 - \phi^T\phi\right) I + 2wS(\phi) + 2\phi\phi^T = R(-w,-\phi) \tag{A.16}$$

Definition of quaternion inverse:

$$Q^{-1} = \begin{bmatrix} -w \\ \phi \end{bmatrix} = \begin{bmatrix} w \\ -\phi \end{bmatrix} \tag{A.17}$$

If $Q \iff R$, then $Q^{-1} \iff R^T$.

Definition of quaternion product:

$$Q_1 \otimes Q_2 = \left[w_1 w_2 - \phi_1^T\phi_2 \left[w_1\phi_2 + w_2\phi_1 + \phi_1 \times \phi_2\right]\right] \tag{A.18}$$

If $Q_1 \iff R_1$, $Q_2 \iff R_2$:

$$Q_1 \otimes Q_2 \iff R_1 R_2 \tag{A.19}$$

Relationship of derivative to angular velocity ω (quaternion propagation):

$$\dot{Q}(t) = \frac{1}{2} E(Q(t))\omega \tag{A.20}$$

$$E(Q) = \begin{bmatrix} -\phi^T \\ wI - S(\phi) \end{bmatrix} \tag{A.21}$$

B

Appendix 2

All formulas taken or derived from [20].

B.1 CLIK

Algorithm 2: Closed Loop Inverse Kinematics

Result: Joint velocities $\dot{q}$ as control input at time t
Input: End-effector reference trajectory: $x_{ref}(t)$, gain constant $\alpha > 0$
for $t = t_0$ **to** t_n **do**
    Read joint states q from sensors
    Solve forward kinematics to compute end-effector pose: $x_e(q)$
    Calculate error term $\Delta x = x_{ref} - x_e$
    Calculate control input $\dot{q} = J_A^{\dagger}(q)(\dot{x}_{ref} + \alpha \Delta x)$
end

B.2 Matrix pseudoinverse

For a given matrix $A \in \mathbb{R}^{m \times n}$, the pseudoinverse $A^{\dagger}$ can be expressed as follows:

If $m > n$ and $\text{rank}(A) = n$; the rows, but not the columns of A are linearly independent:

$$A^{\dagger} = (A^T A)^{-1} A^T \tag{B.1}$$

If $m > n$ and $\text{rank}(A) = m$; the columns, but not the rows of A are linearly independent:

$$A^{\dagger} = A^T (A A^T)^{-1} \tag{B.2}$$

If A is invertible, then B.1 = B.2 = A^{-1}. If A is rank deficient, it may be constructed using singular value decomposition:

$$A = U \Sigma V^T \iff A^{\dagger} = U \Sigma^{\dagger} V^T \tag{B.3}$$