

Trajectory Planning and Subject-Specific Control of a Stroke Rehabilitation Robot using Deep Reinforcement Learning

by

Arash Hashemi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2021

© Arash Hashemi 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

There are approximately 13 million annual new stroke cases worldwide. Research has shown that robotics can provide practical and efficient solutions for expediting post-stroke patient recovery. Assistive robots provide automatic limb training, which saves a great deal of time and energy. In addition, they facilitate the use of data acquisition devices. The data is beneficial in terms of quantitative evaluation of the patient progress.

This research focused on the trajectory planning and subject-specific control of an upper-extremity post-stroke rehabilitation robot. To find the optimal rehabilitation practice, the manipulation trajectory was designed by an optimization-based planner. A linear quadratic regulator (LQR) controller was then applied to stabilize the trajectory. The integrated planner-controller framework was tested in simulation. To validate the simulation results, hardware implementation was conducted, which provided good agreement with simulation.

One of the challenges of rehabilitation robotics is the choice of the low-level controller. To find the best candidate for our specific setup, five controllers were evaluated in simulation for circular trajectory tracking. In particular, we compared the performance of LQR, sliding mode control (SMC), and nonlinear model predictive control (NMPC) to conventional proportional integral derivative (PID) and computed-torque PID controllers. The real-time assessment of the mentioned controllers was done by implementing them on the physical hardware for point stabilization and circular trajectory tracking scenarios. Our comparative study confirmed the need for advanced low-level controllers for better performance. Due to complex online optimization of the NMPC and the incorporated delay in the method of implementation, performance degradation was observed with NMPC compared to other advanced controllers. The evaluation showed that SMC and LQR were the two best candidates for the robot.

To remove the need for extensive manual controller tuning, a deep reinforcement learning (DRL) tuner framework was designed in MATLAB to provide the optimal weights for the controllers; it permitted the online tuning of the weights, which enabled the subject-specific controller weight adjustment. This tuner was tested in simulation by adding a random noise to the input at each iteration, to simulate the subject. Compared to fixed manually tuned weights, the DRL-tuned controller presented lower position-error.

In addition, an easy to implement high-level force controller algorithm was designed by incorporating the subject force data. The resulting hybrid position/force controller was tested with a healthy subject in the loop. The controller was able to provide assist as needed when the subject increased the position-error.

Future research might consider model reduction methods for expediting the NMPC optimization, application of the DRL on other controllers and for optimization parameter adjustment, testing other high-level controllers like admittance control, and testing the final controllers with post-stroke patients.

Acknowledgements

Foremost, I would like to thank my supervisor, Prof. John McPhee, for his gracious support, encouragement and invaluable insight throughout my Master's degree. I am indebted to his guidance. His careful editing contributed tremendously to the production of this thesis.

I would like to thank my committee members, Prof. Katja Mombaur and Prof. James Tung for their suggestions and intellectual contributions to the thesis improvement.

Finally, I would like to thank all the members of Motion Research Group for their help, and advice. Being part of this team was, and will always be, my honour.

Dedication

This thesis is dedicated to my mother without whom I would not be able to start my education and reach this level. No words can describe my gratitude for her sacrifices and unconditional love.

Table of Contents

List of Tables	xii
List of Figures	xiii
1 Introduction	1
1.1 Motivation and Goals	1
1.2 Thesis Organization	2
1.3 Thesis Contributions	3
2 Background and Literature Review	4
2.1 Overview	4
2.2 Stroke and Rehabilitation	4
2.3 Rehabilitation Robotics	5
2.3.1 Rehabilitation Robotics Challenges	5
2.3.2 Control Strategies for Rehabilitation Robotics	6
2.4 Optimization Techniques for Solving Optimal Control Problems	8
2.4.1 Indirect Methods	8
2.4.2 Direct Methods	10
2.4.3 Dynamic Programming	12
2.5 Reinforcement Learning	14
2.5.1 Q-learning	14

2.5.2	Value Function Approximation Methods	16
2.5.3	Policy Gradient Methods	17
2.5.4	Actor-Critic Methods	18
2.6	Conclusion	19
3	Robot and Human Arm Modeling	20
3.1	Overview	20
3.2	Rehabilitation Robot	20
3.2.1	Hardware and Design	20
3.2.2	Software	22
3.2.3	Robot Kinematics	22
3.2.4	Robot Dynamics	25
3.3	Human Arm Model	28
3.4	Human-Robot Interaction Model	29
3.5	Conclusion	30
4	Human-Robot Interaction Trajectory Planning	32
4.1	Overview	32
4.2	Trajectory Planning	32
4.2.1	Problem Formulation	33
4.2.2	Simulation Results	36
4.3	Trajectory Stabilization	40
4.3.1	Linear Quadratic Regulators (LQR)	41
4.3.2	Simulation Results	43
4.4	Experimental Implementation	44
4.5	Conclusion	45

5	Comparative Study of the Rehabilitation Robot Control Algorithms	47
5.1	Overview	47
5.2	Controller Design	47
5.2.1	Proportional-Integral-Derivative (PID) Controllers	48
5.2.2	Computed-Torque PID Controllers	49
5.2.3	Sliding Mode Control (SMC)	51
5.2.4	Nonlinear Model Predictive Control (NMPC)	54
5.3	Simulation Results	60
5.4	Experimental Results	64
5.4.1	Comparison Criteria and Implementation	65
5.4.2	Tuning Process	65
5.4.3	Point Stabilization Results	65
5.4.4	Tracking Results	71
5.5	Conclusion	75
6	Deep Reinforcement Learning Tuning of the Model-based Controllers	77
6.1	Overview	77
6.2	Deep Deterministic Policy Gradient (DDPG)	78
6.2.1	Problems of DDPG	81
6.2.2	Twin-Delayed Deep Deterministic Policy Gradient (TD3)	82
6.3	Implementation	82
6.3.1	Algorithm Validation	83
6.3.2	Tuner Structure	84
6.4	Simulation Results	86
6.5	Conclusion	87

7	Subject in the Loop Experimental Implementation	88
7.1	Overview	88
7.2	Experimental Considerations	89
7.3	Implicit Force Control	89
7.4	Experimental Results	91
7.5	Conclusion	94
8	Conclusion and Future Work	95
8.1	Thesis Summary	95
8.2	Recommendations and Future Work	97
	References	99
	APPENDICES	114
A	Stability Proof of Sliding Mode Control	115
B	Controller Parameters	117
C	TD3 Networks and Hyperparameters Information	119

List of Tables

3.1	Hardware Specification	21
3.2	Link Lengths	22
3.3	Link Mass and Moment of Inertia	26
4.1	Optimization Inequality Constraints	36
4.2	LQR Weights	45
5.1	RMSE Simulation	61
5.2	Control Point Results	67
5.3	Control Tracking Results	72
B.1	Controller Parameters in Simulation	117
B.2	Controller Parameters in Hardware Implementation	118
C.1	Neural Network Parameters	119
C.2	TD3 Hyperparameters	119

List of Figures

2.1	Robot example	13
2.2	RL scheme	15
3.1	Stroke rehabilitation robot (top view)	21
3.2	Stroke rehabilitation robot kinematics	24
3.3	Control-oriented model validation. The end-effector X and Y positions in the workspace are shown.	28
3.4	Human-robot interaction model [49]	30
4.1	End-effector desired path in the workspace.	37
4.2	Robot state trajectories in the jointspace.	38
4.3	Input trajectories. The left plot depicts the robot torques and the right plot shows the muscle activations.	39
4.4	Trajectory error results. The top row highlights the collocation error between the real dynamics (dx/dt) and the dynamics (f) at the collocation points used. It depicts the resulting error due to the discretization. As shown, this error is considerably low which suggests that the application of the direct-collocation was successful.	40
4.5	Schematics of integrated planner and controller. The planner runs offline and provides the desired values that the controller uses to stabilize the trajectory. The resulting robot torques are then applied to the robot manipulator to advance its state forward in time.	43
4.6	LQR control results in simulation. X and Y are end-effector positions. \dot{X} and \dot{Y} are end-effector velocities. Also, τ_1 and τ_2 are robot motor torques.	44

4.7	Experimental implementation of the trajectory stabilizer on the robot. . .	46
5.1	PID block diagram	49
5.2	Computed-Torque PID block diagram	51
5.3	SMC block diagram	54
5.4	MPC block diagram	56
5.5	Maple-based NMPC scheme	59
5.6	Comparison between the iteration turnaround time of CASADI and Maple NMPC	60
5.7	End-Effector position/velocity simulation comparison (2 ms sampling time)	62
5.8	Robot torques simulation comparison (2 ms sampling time)	63
5.9	NMPC implementations with increased sampling time (10 ms sampling time)	64
5.10	Robot schematics for point stabilization	67
5.11	Point stabilization results	68
5.12	Point stabilization torque/current results	69
5.13	PID point inputs	70
5.14	PID low inputs	70
5.15	PID end-effector positions with lower inputs	71
5.16	Robot schematics for tracking	72
5.17	End-effector X position result for tracking	73
5.18	End-effector Y position result for tracking	73
5.19	End-effector X velocity result for tracking	74
5.20	End-effector Y velocity result for tracking	74
5.21	Robot torques and current results for tracking	75
6.1	Actor-Critic schematics	78
6.2	The TD3-controlled pendulum	83
6.3	The whole-episode tuning strategy	84
6.4	The iteration-based tuning strategy	85

6.5	The controller-tuner scheme	85
6.6	The SMC adaptive weights	86
6.7	The SMC position comparison between fixed vs adaptive weights	87
7.1	The hybrid position/force control scheme	90
7.2	The local-global force relation.	90
7.3	SMC subject in the loop results for the circular trajectory. The top left plot shows the end-effector position P and the desired value P_d . The top right plot depicts the subject force in the global XY coordinates. The bottom plots show the subject $u_{subject}$, controller $u_{controller}$, and final torques u_{robot} .	92
7.4	LQR subject in the loop results for the circular trajectory. The top left plot shows the end-effector position P and the desired value P_d . The top right plot depicts the subject force in the global XY coordinates. The bottom plots show the subject $u_{subject}$, controller $u_{controller}$, and final torques u_{robot} .	93
7.5	LQR subject in the loop testing for tracking the planner trajectory. The top left plot shows the end-effector position P and the desired value P_d . The top right plot depicts the subject force in the global XY coordinates. The bottom plots show the subject $u_{subject}$, controller $u_{controller}$, and final torques u_{robot} .	94

Chapter 1

Introduction

There are over 13 million annual new cases of stroke incidence worldwide [76], and 33% of the survivors have reported limited or no use of their upper-limb. Extensive research has been dedicated to finding practical approaches to aid the recovery of the post-stroke patients. Interestingly, robotics has been proven to be one of the feasible solutions [68]. This thesis focused on an end-effector-based robot for post-stroke upper-extremity rehabilitation.

1.1 Motivation and Goals

This work examined the trajectory planning, comparative control implementation, and automatic controller tuning of our experimental setup. The rehabilitation practice is one of the most critical aspects of patient's improvement; it determines the progress of their motor recovery. Motivated by this fact, the goal of the first phase of the thesis was to design a systematic approach for obtaining the optimal practice trajectory. Controller synthesis was then required to maintain the robot on the desired trajectory. Various options could be chosen for this purpose. Needless to say, each choice had its own advantages and drawbacks. The goal of the second phase of the thesis was to evaluate and find the most suitable controller among five well-known algorithms in the literature. The best controller candidates included weights that required extensive manual tuning, which is a time-consuming process and does not result in optimal weight settings. This issue motivated the third phase of the thesis to focus on an automatic framework for judicious tuning of the parameters. Finally, to incorporate the subject in the control loop, the fourth phase concentrated on designing an intuitive and easy to implement high-level force controller.

1.2 Thesis Organization

- Chapter 1 states the thesis motivations and goals. It also presents the project organization and contributions.
- Chapter 2 provides the background. A detailed review of rehabilitation robotics is presented and the challenges and control strategies are discussed. A comprehensive section is dedicated to optimization techniques and their positive and negative aspects. This information is useful when reading Chapters 4 and 5, where the optimization techniques were extensively utilized. Finally, a summary of reinforcement learning and its progress during recent years is presented. This part helps with the grasp of the main DRL algorithms, namely Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3), in Chapter 6.
- Chapter 3 discusses the rehabilitation robot and human arm model. The low-fidelity robot model was used in Chapters 4 and 5 for model-based controllers. The high-fidelity model, used for controller testing in the aforementioned chapters, is also presented. The integrated human-robot interaction model is then provided. This model is utilized in Chapter 4 for trajectory planning.
- Chapter 4 outlines the trajectory optimization framework. The first section of this chapter discusses the optimization method and the planner framework. The remaining section outlines the trajectory stabilizer for maintaining the robot on the desired trajectory. Simulation results are presented to evaluate the integrated planner-controller. Lastly, experimental results are provided to validate the simulation.
- Chapter 5 introduces the comparative study of the controller algorithms on the robot. The mathematical formulation of five controllers is presented. In addition, two methods of implementing symbolic-based model predictive control are examined. Later in this chapter, comparative simulation results are presented. The final section of the chapter focuses on experimental results of the controllers and providing the best candidates.
- Chapter 6 discusses the formulation and implementation of the DRL algorithms. In addition, the application of the tuner on the SMC candidate controller is presented.
- Chapter 7 describes the process for considering the subject in the control loop for experimental evaluation. The high-level force controller is then presented. The experimental results for the hybrid position/force controller is provided.

- Chapter 8 outlines the conclusion of this research by reviewing the summary of the work. The limitations and recommended future work are also discussed.

1.3 Thesis Contributions

- Development of a novel optimization-based trajectory planning framework with hybrid human-robot cost function for calculating the rehabilitation manipulation practice.
- Designing a LQR trajectory stabilizer to be integrated with the planner.
- Evaluation of the integrated planner-controller in simulation and real-time experiments.
- Conducting a comparative study for finding the best controller candidates among five well-known controllers.
- Development of a DRL tuner to be used with model-based controllers.
- Development of an easy to implement high-level force controller.
- Conducting subject-in-the-loop real-time tests by applying the final hybrid position/force controller.

Chapter 2

Background and Literature Review

2.1 Overview

In this chapter, the literature review for the explored pathways in this thesis is presented. In addition, the foundation of the ideas, discussed in the subsequent chapters, is outlined.

2.2 Stroke and Rehabilitation

Stroke is defined as “rapidly developing clinical signs of local (or global) disturbance of cerebral function, with symptoms lasting 24 hours or longer or leading to death, with no apparent cause other than of vascular origin” by The World health Organization (WHO) [105]. As the leading cause of adult disability [112], and one of the prevalent causes of global death [106], this disease has brought about physiological, psychological, social, and economical impacts. A great portion of post-stroke patients live with long-term disabilities which hamper the normal routine of their life and even interfere with their casual daily activities. Given that more than two-thirds of post-stroke patients have upper limb impairment [104], and considering that most daily activities are associated with utilizing the upper limb, it is crucial to find methods that can help with the motor recovery of post-stroke patients. Physiotherapists and chiropractors have been tremendously helpful in terms of mitigating the negative effects of stroke. Having said that, there are practical approaches that can increase the productivity of the rehabilitation even further.

2.3 Rehabilitation Robotics

Robot-assisted rehabilitation can improve the motor recovery process [51, 111, 136]. Generally speaking, rehabilitation involves employing daily repetitive practices, which are done by patients and supervised/aided by post-stroke physiotherapists. Robots can show utility by automating this process, hence saving a considerable time and effort. Moreover, rehabilitation robots are (or can be) equipped with force/torque sensors, encoders, Electromyography (EMG), etc. The data can then be utilized to quantitatively analyze the patient progress. For instance, EMG data could be mapped back to muscle activation which can then be used to determine the progress of a particular muscle, or a group of muscles. The positive effect of robot-assisted rehabilitation in the motor recovery of post-stroke patients has been demonstrated in [15, 142, 68, 51]. It is worthwhile to note that robot-assisted rehabilitation does not remove the need for physiotherapists, and chiropractors. The choice of practices which are directly associated with motor recovery, requires clinical insight and should be selected adaptively. The robot operates as a tool to aid the performance of the repetitive movements. Similar to other approaches, there are barriers for the general employment of robotic rehabilitation. These challenges are studied in the next section.

2.3.1 Rehabilitation Robotics Challenges

Similar to other branches of robotics, rehabilitation robotics possesses important challenges that need to be addressed for a productive motor recovery [149].

Safety

One of the cardinal factors to consider when designing, controlling, and working with rehabilitation robots is the safety. As we enter the subbranch of Human-Robot Interaction (HRI), the robots should be able to perform the given task successfully and at the same time, minimize the probability of imposing any damage upon patients. On the design side, this can be achieved by selecting the appropriate materials, adopting the appropriate design strategy, and choosing the suitable electrical hardware. Low-power motors can ensure that damage to the subject is almost prohibited even when exerting the maximum input. Also, using online protective devices, like emergency push buttons, is a necessity, as they enable emergency shut down of setup whenever unstable robot behavior is observed. There are strategies on the software side as well that have proven helpful. Restricting the controllers

to a maximum torque/current can be achieved on the software side by adopting saturation functions or current limits. Interestingly, the limit on the hardware and motor power calls for better high-level and low-level controllers that motivates research in this field.

User-Interface

As mentioned before, rehabilitation practices often include repetitive tedious activities which are not always compelling to stick to. Adding the fact that the majority of post-stroke patients are elderly, we can conclude that we are in need of mechanisms to make these practices more engaging and fun for the patients. Developing appropriate user-interfaces for robots by taking advantage of gamification and Virtual Reality (VR) can be a suitable mechanism. In addition, Artificial Intelligence (AI) can be integrated in these interfaces for patient language processing, emotion recognition, etc.

Ethical Aspects

Working with a robotic device that imposes artificial interaction with a subject can have ethical and social impacts, especially on the elderly population because of their specific physical and mental levels; the robot intelligence and autonomy can potentially cause alienation in patients. With this regard, the ethical constraints and possible repercussions of the HRI should be considered. On top of that, the legal aspects of interaction should be clarified. It should be noted that the User-Interface and Ethical Aspects topics are beyond the scope of this thesis and are only mentioned for the sake of completeness; hence, these topics are not investigated in other chapters.

2.3.2 Control Strategies for Rehabilitation Robotics

There are various categorizations of control strategies for rehabilitation robots in the literature. Here, the focus is mostly on the ones presented in [89, 100]. Generally, control strategies are divided into **High-Level** and **Low-Level**. High-level control strategies are “explicitly designed to provoke motor plasticity”. On the other hand, low-level strategies strive to “control the force, position, impedance, or admittance factors of high-level control strategies”.

High-Level Control Strategies

1. **Assistive Control Strategies:** These approaches help the patient in terms of completing the repetitive practices, and are divided into *Impedance-based*, *Counterbalanced-based*, *EMG-based*, and *Performance-based Adaptive* controllers. Impedance control which is a force controller strategy, applies a restoring force when the patient deviates from the desired trajectory. Counterbalanced-based methods provide passive, or active limb weight counterbalance, hence make the exercise easier and more focused on following the desired trajectory. In this regard, we used a passive upper-limb weight counterbalance during subject testing in Chapter 7. EMG-based methods take advantage of patient sEMG signals to determine/trigger the assistance. Finally, performance-based adaptive controllers use the performance data of the patient, including the sum of deviations from the desired trajectory to adapt the assistance (force, impedance parameters, etc.) or to reset the practice (trajectory, time, admittance parameters, etc.).
2. **Challenged-based Control Strategies:** These methods include *Resistive*, *Error-Amplified*, and *Constraint-Induced*. Resistive approach, which is the primary strategy in challenged-based approaches, adopts an opposite mechanism to assistive strategies and resists patient motion (instead of accommodating it). Error-amplified methods increase the deviation error. It has been shown that this approach can lead to faster improvements [109]. Finally, constraint-induced methods limit the use of the non-affected limb so that the affected-limb does most of the practice without extra help.
3. **Haptic Simulation Strategies:** As mentioned in the previous section, the User-Interface is an essential part of robot-assisted rehabilitation. Haptic Simulation methods utilize haptic devices and tactile sensors, along with virtual reality (VR) technology to create an immersive and realistic environment for post-stroke patients and improve their engagement.
4. **Non-contacting Coaching Strategies:** These approaches are different from other categories in that there is no contact-based HRI. The patient performs a predefined practice and a system monitors his/her movement. AI-based vision systems are highly effective in this application. By detecting posture and identifying joint positions, the deviation from the desired joint trajectory can be calculated and the corrective visual signal can be provided for the patient to correct the practice.

The aforementioned strategies can be modified and combined for particular applications.

Low-Level Control Strategies

Low-level control strategies assist the employment of high-level controllers, and are commonly used in general control tasks. These approaches are divided into position, force, or hybrid position-force controllers. Various categories can be assigned to low-level controllers like model-free versus model-based, linear versus nonlinear, optimization-based versus non-optimization-based, etc. In order to address the safety issues, impedance/admittance controllers are also implemented along with the low-level ones. As many of the low-level strategies take advantage of optimization, it is imperative to know about various optimization techniques, their advantages, and drawbacks.

2.4 Optimization Techniques for Solving Optimal Control Problems

Optimal control problems include solving for some *decision variables*, such that a performance index is minimized (for cost functions) or maximized (for reward functions). Depending on the problem formulation, the output of the optimizer may be the unknown parameters (function optimization), or the input function (functional optimization). In the case of the former, the problem is usually referred to as “optimal control” and in the case of the latter, it is referred to as “trajectory optimization” [118]. Generally speaking, optimization methods for optimal control divide into three major categories: indirect, direct, and dynamic programming. All three methods, or their modified versions, are actively used in optimal control and sequential decision-making problems. Needless to say, each approach has its own positive and negative aspects. In what follows, each approach and its potential application is discussed [118]:

2.4.1 Indirect Methods

In these methods, the problem is “first optimized, then discretized”. Calculus of variations is utilized to derive the Karush-Kuhn-Tucker (KKT) optimality conditions. The optimization problem then turns into a multiple-point boundary value (BV) problem. Analytical or (in most cases) numerical recipes are then exploited to solve the resulting BV problem. The optimal control is turned into solving a system of nonlinear algebraic equations. Various classes of methods for solving **Nonlinear System of Equations** and **Differential Equations and Function Integration** have been studied in indirect methods.

Two main numerical approaches for solving differential equations are **Time-Marching** and **Collocation** methods. Time-marching methods divide into *Multiple-Step* and *Multiple-Stage* categories. In multi-step methods, the solution of the current time-step is acquired by using the solution of n previous time-steps. Depending on the formulation of steps and the choice of n , different well-known numerical integration methods are obtained. The simplest multi-step method is the *forward Euler* approach; only the previous time-step's solution is utilized for finding the current time-step's solution ($n = 1$):

$$\begin{aligned} x(t_{k+1}) &= x(t_k) + \Delta t_k f(x_k, t_k) \\ \Delta t_k &= t_{k+1} - t_k \end{aligned} \tag{2.1}$$

In Eq. 2.1, $x(t_k)$ and $f(x_k, t_k)$ are the solution and the derivative of the solution at time t_k , respectively. In multi-stage approaches, the solution interval $[t_0, t_f]$ is divided into K subintervals $[\tau_n, \tau_{n+1}]$, at which the integral is approximated by a quadrature:

$$\begin{aligned} \int_{t_0}^{t_f} f(x(t), t) dt &\approx \Delta t \sum_{n=1}^K \beta_n f(x_n, \tau_n) \\ x(t_f) &\approx x(t_0) + \Delta t \sum_{n=1}^K \beta_n f(x_n, \tau_n) \\ \Delta t &= t_f - t_0 \end{aligned} \tag{2.2}$$

β_n are the problem coefficients and are found by calculating the states $x(\tau_i)$ and $f(x(\tau_i), \tau_i)$ at each subinterval points. In collocation methods, the interval is again divided into subintervals; however, the states themselves are approximated by piece-wise polynomials of degree K . The accuracy of the solution depends on the choice of K :

$$X_{approx}(t) = \sum_{n=0}^K c_n (t - t_0)^n \quad t \in [t_0, t_f] \tag{2.3}$$

The resulting coefficients c_n are then found. This is done by equating the approximate states to the states at t_0 . Also, the derivative of the approximate states should match the derivative of the states (the right-hand side of the differential equation):

$$\begin{aligned} X_{approx}(t_0) &= x(t_0) \\ \dot{X}_{approx}(\tau_j) &= f(\tau_j) \end{aligned} \tag{2.4}$$

As seen later in the chapter, collocation methods have the advantage of solving for all the unknowns simultaneously. In addition, the dynamics is simulated in *parallel* since the states are found at the same time with the coefficients. On the contrary, in time-marching methods, the inputs and states are found sequentially.

2.4.2 Direct Methods

In direct methods, instead of checking for first-optimality conditions and then solving the resulting system of nonlinear equations, the controls (control parametrization) or states/controls (state-control parametrization) are approximated directly. In other words, the direct methods are “discretize first and then optimize” classes of approaches. Instead of reaching a system of nonlinear algebraic equations, the discretization leads to a nonlinear optimization problem or a nonlinear program (NLP). Here, we briefly mention the direct methods for solving optimal control problems and discuss the advantages and disadvantages of each method. It is worthwhile to note that the aforementioned methods have indirect versions as well. These versions are not described since they were not applied in this thesis.

Direct Single-Shooting Methods

Single-shooting is a *control parametrization* approach where only the controls are approximated by functions:

$$u(t) = \sum_{n=0}^K c_n \phi_n(t) \tag{2.5}$$

where $u(t)$ is the control function, c_n are the unknown parameters, and ϕ_n are the known function approximators. The states and the cost function are found by integrating (simulating) the dynamics forward (model roll-out) using a time-marching approach. An initial guess for the unknown parameters is used and modified, by evaluating the terminal cost and the cost function, until the minimum is acquired. While simple and intuitive, single-shooting methods suffer from several numerical issues that lead to the ill-conditioning of the Hamiltonian [118, 132]. One of these issues is that this method is too sensitive to the initial guess and can go unstable when it is far from the answer. This problem is mitigated by using a modified version of single-shooting which is called *multiple-shooting*.

Direct Multiple-Shooting Methods

In this method, the interval $[t_0, t_f]$ is divided into subintervals $[\tau_j, \tau_{j+1}]$ where the direct single-shooting algorithm is performed in each of them. Each subinterval has its own unknown parameters. The continuity condition in Eq. 2.6 enforces that the last state of each interval is the same as the first state of the next interval.

$$x(\tau_{j-}) - x(\tau_{j+}) = 0 \tag{2.6}$$

Multiple shooting method alleviates the numerical issues of single-shooting; by decreasing the integration interval, the error of initial conditions decreases. Despite this improvement, there are other fundamental issues with shooting methods that make the next method, *Direct Collocation*, favorable in many robotic applications. The following briefly mentions these issues [132]:

- Since the states are not part of the decision variables, in problems with state constraints, each constraint needs to be simulated again, hence making the optimization slower.
- Using the time-marching approach in single-shooting methods enforces a sequential solution (finding u and rolling out the dynamics and cost). This inhibits the optimizer to calculate the dynamics and the decision variables in parallel, hence hindering the use of parallel computing in large and nonlinear problems.

Direct Collocation

Direct collocation resolves all the aforementioned issues by considering the states in the decision variables (state-control parametrization). The problem is discretized by approximating the state and control using appropriate functions. This process is also called “Transcription”. Various choices can be implemented for the state-control function approximators and that leads to the multiple existing transcription methods. One of these methods, namely *Trapezoidal Transcription* is discussed in Chapter 4, where the direct collocation was implemented for trajectory planning. The procedure is similar to the collocation approach for integration in Section 2.4.1. Due to having the states as decision variables, adding state constraints is more computationally tractable. In addition, states and inputs are calculated in parallel. Although the size of the problem increases by adding the states to the decision variables, the resulting nonlinear program is sparse usually and appropriate sparse solvers, like SNOPT [52], can be exploited for this matter.

2.4.3 Dynamic Programming

First introduced by Bellman [12], dynamic programming (DP) provides a new perspective on solving optimization problems and has been in the spotlight for optimal control problems in the last two decades [124]. As in the other two methods, the aim is to minimize a cumulative cost (or maximize a cumulative reward). The major requirement in DP is that the problem should be represented as a *multi-stage optimization problem*. The more concrete mathematical requirement for this process is that it should have the *Markov Property*. This property states that: given the state x_k and action a_k at the k th stage, the next stage state x_{k+1} can be calculated independent of the previous history of states and actions. The process with this property is called a *Markov Decision Process* (MDP). Dynamic Programming is a great candidate for sequential decision making fields, including control, machine learning, shortest path problems (SPP), etc. The core idea of DP which is called **The Principle of Optimality** is fairly simple and highly intuitive. This principle states that the tail portion of the main problem should also be optimal. In other words, if we split the main N -stage problem into multiple k -stage problems, the solution for all of those k -stage problems should also be optimal. This notion can be readily followed by intuition. If the solution of the k -stage problem is not optimal, there exists a better solution that can replace it. DP assigns a value metric to each state at each stage, $J_k(x_i)$, where $k = 0, \dots, N - 1$ and $i = 1, \dots, n_s$ with N being the number of stages and n_s being the number of states. Roughly speaking, the value metric of a state at a particular stage k shows how much cost (or reward) is accumulated starting from that state onward until the end of the sequence; the value shows how “good” it is to be in a particular state. This is different from the immediate cost $g_k(x_i, u_j)$ that is received by taking action u_j at state x_i .

More elaboration is presented with an example, as the understanding of this concept is crucial for following the subsequent sections. Suppose we have a robot in a grid (Fig 2.1). Starting from the initial condition, the goal is to control the robot to the desired point (green square) with N moves (stages). The robot can employ “up”, “down”, “left”, and “right” commands. In this example, entering each square has an immediate cost $g_k(x_i)$. The “cost to go” for each square, after employing i moves and ending up at that square, is the summation of all the costs that are accumulated from that square after commanding $N - i$ moves.


-1	-1	1
-1	-1	-1
	-1	-1

Figure 2.1: Robot example

The *cost to go function*, which is a common term in the nomenclature of optimal control, is substituted with *value function* in the nomenclature of AI. Value functions are discussed in the Reinforcement Learning section. The solution of the optimal control is found implicitly by finding the optimal costs to go $J_k^*(x_i)$. The solution is divided into backward and forward passes. The backward pass includes starting from the tail section backwards to the initial stage; this part is demonstrated below:

The last stage:

$$J_N^*(x_i) = g_N(x_i) \quad \text{for all } x_i \tag{2.7}$$

and for $k=0, \dots, N-1$:

$$J_k^*(x_i) = \min_{u_j} [g_k(x_i, u_j) + J_{k+1}^*(f_i(x_i, u_j))]$$

The cost to go in the last stage is similar to the immediate cost since the terminal stage is reached. For other stages, the optimal cost to go is found by the Bellman update (Eq. 2.7). This equation is solved iteratively until the initial state is reached. Note that the dynamics (transition) model of the system is required to find the next state cost to go $J_{k+1}^*(f_i(x_i, u_j))$. DP is able to work with both discrete and continuous dynamics. With few modifications, it can also handle stochastic problems. As the optimal costs to go are obtained, the forward pass is initiated to find the optimal inputs.

Initial Step:

$$u_0^* = \arg \min_{u_0} [g_0(x_0, u_0) + J_1^*(f_0(x_0, u_0))]$$

Subsequent Steps ($k = 1, 2, \dots, N - 1$): (2.8)

$$u_k^* = \arg \min_{u_k} [g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k^*, u_k))]$$

where u_k^* is the optimal input at stage k . The presented algorithm is the *Exact Dynamic Programming*. In subsequent sections, the approximate version of this algorithm is presented.

2.5 Reinforcement Learning

Reinforcement Learning (RL) is an optimization tool for solving real-world problems. It is considered as approximate DP ([16]) since it uses Bellman equation and the same DP formulation at its core. During recent years, researchers have implemented approximations on the original formulation of DP to make it practical for higher-space problems and problems related to physical systems, in particular. The schematics of RL is depicted in Fig. 2.2 [128]. At each iteration, the RL agent interacts with the environment by choosing an action a_t , which causes the transition of the environment from its previous state s_t to its next state s_{t+1} ; an immediate reward r_t is then obtained. The goal of RL is to modify the agent so that it produces actions that maximize the cumulative reward. In this section, the minimization of cost is substituted with maximization of the reward and hence, the term “value function” is used instead of cost-to-go function. RL is a semi-supervised learning algorithm. Unlike supervised-learning methods, optimal targets, or labels in the field of supervised learning, are not available before training. The generated sub-optimal output **during** the training process is used instead.

2.5.1 Q-learning

DP algorithm is model-based in the sense that a dynamics model is required to go from state-dependent J_k^* to optimal inputs u_k^* . Inspired from DP, a model-free algorithm

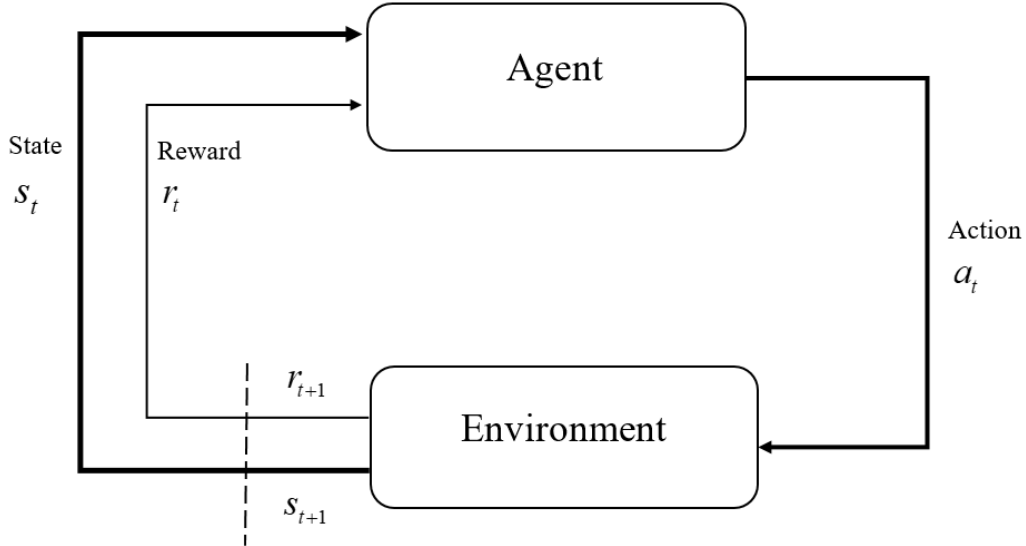


Figure 2.2: RL scheme

called “Q-learning” was introduced [46]. In this algorithm, instead of defining a state-dependent cost to go, state-action value functions $q(s, a)$ which are dependent on both the state and the action, are defined. Loosely speaking, $q(s, a)$ determines how good it is to be in state s and execute action a . Using this method, there is no need for a dynamics model and state-action pairs are sufficient for training the agent. The Bellman update in this case is represented as Eq. 2.9, where $Q_k(x_i, a_j)$ is the k th-stage state-action value function at (x_i, a_j) , and $V_{k-1}(x_{i+1})$ is the $(k - 1)$ th-stage state value function at x_{i+1} ; α_k determines the effect of Q values of the previous time-step on the current Q value, and γ determines the discount factor on the Bellman update. Similar to DP, Q-learning is an iterative algorithm. With this regard, Q values for each state-action pairs can be stored in a table and get updated according to Eq. 2.9.

$$\begin{aligned}
Q_k(x_i, a_j) &= (1 - \alpha_k)Q_{k-1}(x_i, a_j) + \alpha_k[r_k + \gamma V_{k-1}(x_{i+1})] \\
V_{k-1}(x_{i+1}) &= \max_a Q_{k-1}(x_{i+1}, a) \\
i &= 1..n_s \quad ns = \text{number of states} \\
j &= 1..n_a \quad na = \text{number of actions} \\
k &= 1..N \quad N = \text{number of stages}
\end{aligned} \tag{2.9}$$

The subtle differences between DP and Q-learning has made the latter a strong model-free algorithm which has been useful for solving problems in a variety of topics. Having said that, storing Q values in a table is not a practical strategy in more complex problems with higher state-action spaces. In addition, solving the maximization is more difficult. These issues called for modifications and approximations to this algorithm, especially for real-world problems. There are two major perspectives for applying approximations to the original RL formulation. **Value Function Approximation** and **Policy Approximation**. Since both of these perspectives were applied in the DRL algorithms presented in Chapter 6, they are briefly introduced in the following section.

2.5.2 Value Function Approximation Methods

In order to make RL a feasible solution for large MDPs, approximations are applied to it during recent years. One perspective is to approximate the value functions, or Q values, and then find the actions implicitly. A *Parameterized* value function approximation is a method in which the value/ Q function is defined as a function of a set of parameters w and the best w that can describe $v(s)$ or $q(s, a)$, are found by solving an optimization problem:

$$\begin{aligned}
\hat{v}(s, w) &\approx v(s) \\
\hat{q}(s, a, w) &\approx q(s, a)
\end{aligned} \tag{2.10}$$

where $\hat{v}(s, w)$ and $\hat{q}(s, a, w)$ are the parameterized approximate state value functions and state-action value functions, respectively. There are a myriad of function approximations that can be utilized for this purpose. Linear function approximators (like least square methods), nonlinear function approximators (like neural networks), decision trees, and Fourier/wavelet bases just to name a few. Generally speaking, C2-continuous function approximators are preferred. Due to their mathematical features and their success in

approximating complex function in recent years, deep neural networks (DNNs) are preferred mostly in the literature. Value functions can be approximated by finding the optimal parameters of a neural network. This can be done by structuring the problem as an optimization. Incremental methods, like *Gradient Descent*, are usually utilized to solve the optimization problem. Combining DNN-based value function approximation with Q-learning has resulted in a phenomenal success in the field of RL. The outcome, which is called deep Q networks (DQNs), has been able to present human-level performance in Atari games [98, 99]. After the optimal value function is approximated, the action (policy) can be obtained by:

$$a^* = \underset{a}{\operatorname{arg\,max}}[\hat{q}(s, a, w)] \tag{2.11}$$

Another popular method for finding the actions is called “epsilon-greedy” in which a random number is generated and compared to a predefined parameter ϵ . If the number is greater than ϵ , then Eq. 2.11 is used for finding the action. If not, a random action is taken, within the bounds of the action space. This approach allows for more exploration and circumvents getting stuck in local maximums.

2.5.3 Policy Gradient Methods

Another idea for making RL practical for complex problems is to explicitly approximate the final action (policy), instead of deriving it from the approximated value function with an ϵ -greedy approach. Policy-based RL methods have better convergence features. They are more sample-efficient in high-dimensional state-action spaces, and can learn stochastic policies, which is necessary in many applications like games. It is also sometimes a good idea to employ a stochastic policy to improve robustness, for instance, for control applications. On the other hand, policy-based methods are more prone to convergence to local optimum, and they often have high variance. The policy $\pi_\theta(s)$ is now a function of parameter θ ; the optimal θ is found similar to value function approximation approaches. By defining an optimization metric $J(\theta)$, the parameters are adjusted in the direction that maximizes J and provides optimal policy. The formulation of an one-step MDP is shown in Eq. 2.12:

$$\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\
&= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) R_{s,a} \\
\Delta_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \Delta_\theta \log \pi_\theta(s, a) R_{s,a} \\
&= \mathbb{E}_{\pi_\theta}[\Delta_\theta \log \pi_\theta(s, a) r]
\end{aligned} \tag{2.12}$$

J is defined as the expectation of the one-step reward for different policies. $d(s)$ is the probability distribution of the states in stochastic problems. $R_{s,a}$ is the received reward at the state s after taking the action a . In order to find the gradient of J , the “policy gradient trick” is used:

$$\Delta_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \frac{\Delta_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} = \Delta_\theta \log \pi_\theta(s, a) \tag{2.13}$$

The gradient of an expectation in Eq. 2.12 is itself an expectation using this simple mathematical trick; this equation can be expanded to multi-step MDPs (Eq. 2.14). Using stochastic gradient descent, the parameters are adjusted at each iteration. This way, we can replace $Q^{\pi_\theta}(s, a)$ with r_t , which is the immediate reward of that iteration. This approach leads to an algorithm called REINFORCE [145]. Another approach is to use Value Function Approximation techniques to approximate $Q^{\pi_\theta}(s, a)$. This method is the topic of the next section.

$$\Delta_w J(w) = \mathbb{E}_{\pi_w}[\Delta_w \log \pi_w(s, a) Q^{\pi_w}(s, a)] \tag{2.14}$$

2.5.4 Actor-Critic Methods

Policy-gradient methods have their own drawbacks, like high-variance gradient approximations. Instead of using the immediate reward, action value functions can be approximated directly. Actor-critic methods [67] use a **Critic** to approximate the $Q^{\pi_\theta}(s, a)$ and an **Actor** to approximate the π_θ . In other words, the actor maps states to actions and the critic updates this mapping to produce actions that increase the Q values. Neural network function approximators can be utilized for both the actor and the critic. With neural networks, the critic updates the action-value function parameters w and the actor updates policy parameters θ . Actor-critic removes the drawbacks of previous methods but

adds the approximation error due to the application of neural networks. The approximate policy gradient algorithm is as follows:

$$\begin{aligned}
 Q_w(s, a) &\approx Q^{\pi_\theta}(s, a) \\
 \nabla_\theta J(\theta) &\approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)] \\
 \Delta\theta &= \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)
 \end{aligned}
 \tag{2.15}$$

There have been many variations of actor-critic algorithms in the literature [17, 53, 110]. The two main DRL algorithms in this thesis, namely DDPG and TD3, also employ an actor-critic scheme. We discuss the details of these algorithms in Chapter 6.

2.6 Conclusion

In this chapter, we presented the background and literature on rehabilitation robotics, optimization techniques, and reinforcement learning. The information helps with the grasp of next chapters where we use some of the aforementioned approaches.

Chapter 3

Robot and Human Arm Modeling

3.1 Overview

In this chapter, the modeling of the rehabilitation robot and the human arm is discussed. Developing a good model is of utmost importance in model-based control. The model should be detailed enough to capture the system's inherent dynamics, and computationally tractable to enable fast simulations and real-time experimental implementation. To this end, both control-oriented and high-fidelity models were developed.

3.2 Rehabilitation Robot

Our rehabilitation robot was developed by Quanser Inc., the Toronto Rehabilitation Institute (TRI), and the Motion Research Group (MoRG) at University of Waterloo to expedite upper extremity motor recovery. It is an end-effector-based robot and operates in the horizontal plane. The patient does the repetitive practices by grabbing the end-effector and moving it in the desired manipulation direction. The robot helps the patient for this purpose by providing assistive/resistive forces on the user's hand.

3.2.1 Hardware and Design

The robot is a 2 degrees of freedom (DOF) fully actuated planar parallelogram mechanism and comprises four Aluminum links (Fig. 3.1). It is equipped with two DC motors

and two optical encoders, which are connected to the actuated joints driving l_1 and l_2 by disc-and-timing belt mechanisms. In addition to the original hardware from Quanser, the robot is equipped with a six axis force/torque sensor (ATI Industrial Automation F/T Sensor: Nano25) on the end-effector. The specification of the motors, encoders, and the force sensor is shown in Table 3.1.

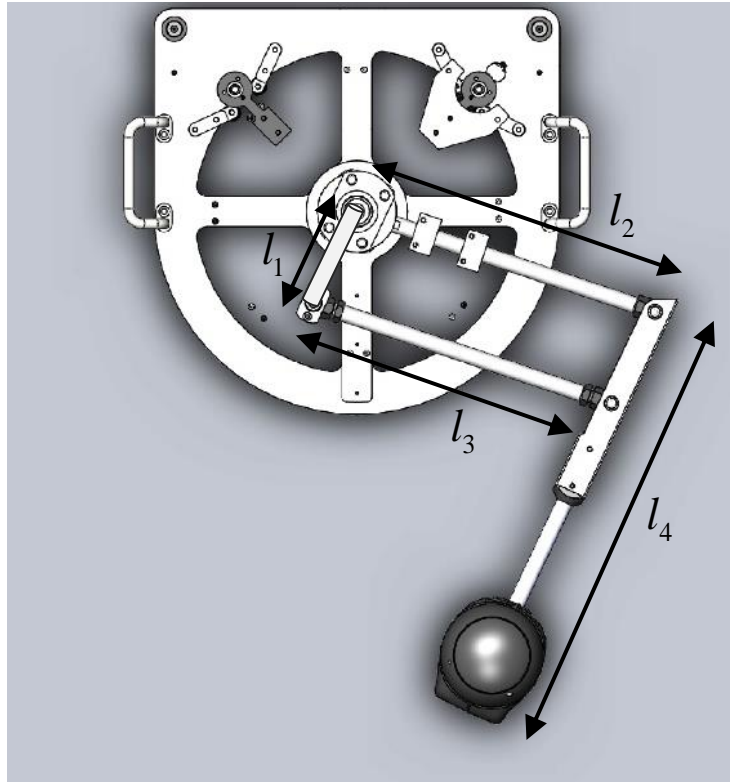


Figure 3.1: Stroke rehabilitation robot (top view)

Table 3.1: Hardware Specification

Motor Torque Constant	$K_T = 0.115 \text{ Nm/Amp}$
Gear Ratio	r=16:307
Motor Rate	115 mN-m
Motor Encoder Resolution	4000 count/revolution
Force Sensor Limit	250 N (Horizontal) , 1000 N (Normal)
Force Sensor Resolution	1/24 N (Horizontal) , 1/48 N (Normal)

3.2.2 Software

The robot uses QUARC real-time control software, which generates code from Simulink models and runs them on the Windows target in real-time. The software utilizes a Q8 Quanser Data Acquisition (DAQ) card. The encoder data is read by including a HIL Read block in the Simulink model. The force sensor data is acquired using a National Instruments (NI) card, and a HIL Analog Read block. The motor current is calculated by dividing the controller torque by the torque constant. The HIL Write block is then utilized for sending the command to the motors. In order to transfer the data between multiple Simulink models, the QUARC communication API is taken advantage of by incorporating Server/Client blocks. Moreover, QUARC stream functions enable the data transfer between MATLAB scripts and Simulink models. Before deploying the software on the robot, it is built in Simulink. The generated C code is then sent to the RAM for implementation in the external mode. Depending on the application, some files can run in the normal simulation mode. This mode is usually used when there are multiple Simulink models. The normal simulation models calculate the controller torque and send them to the primary model in the external mode for robot deployment.

3.2.3 Robot Kinematics

Before delving into the dynamic modeling of the robot, the forward and inverse kinematics is presented. Trigonometry was used to derive the kinematic equations. The link length information, which was required for deriving the kinematics, is represented in Table 3.2.

Table 3.2: Link Lengths

l_1	0.100 m
l_2	0.310 m
l_3	0.310 m
l_4	0.375 m

Forward Kinematics

The forward kinematics of the robot transformed the robot joint-space to the end-effector work-space. Eq. 3.1 represents the forward kinematics of the robot.

$$\begin{aligned}
P_R &= \begin{bmatrix} x_e \\ y_e \end{bmatrix} \\
\theta_R &= \begin{bmatrix} \theta_{R1} \\ \theta_{R2} \end{bmatrix} \\
\begin{bmatrix} x_e \\ y_e \end{bmatrix} &= \begin{bmatrix} l_2 \cos(\theta_{R1}) + l_4 \cos(\theta_{R2}) \\ l_2 \sin(\theta_{R1}) + l_4 \sin(\theta_{R2}) \end{bmatrix}
\end{aligned} \tag{3.1}$$

where P_R is the end-effector position in the workspace, and θ_R is the column matrix of the driven joint angles. The relation between velocity and acceleration levels in the jointspace and workspace is found in Eq. 3.2. J_R is the robot geometric Jacobian.

$$\begin{aligned}
\dot{P}_R &= J_R \dot{\theta}_R \\
\ddot{P}_R &= \dot{J}_R \dot{\theta}_R + J_R \ddot{\theta}_R \\
J_R &= \begin{bmatrix} -l_2 \sin(\theta_{R1}) & -l_4 \sin(\theta_{R2}) \\ l_2 \cos(\theta_{R1}) & l_4 \cos(\theta_{R2}) \end{bmatrix}
\end{aligned} \tag{3.2}$$

Inverse Kinematics

A trigonometric approach was utilized to find the closed-form inverse kinematic solution of the robot. The corresponding points and vectors are shown in Fig 3.2, from the top view. P_b is the origin of the base coordinate frame of the robot, and P_e is the end-effector position with respect to the origin. \vec{E} connects the P_b to P_e . P_m is the position of the intersection point of link 2 and link 4 with respect to the origin. m_1 , m_2 , and m_3 are scalars. Eq. 3.3 is used to find the magnitude of \vec{E} and relate its value to scalars m_1 , m_2 .

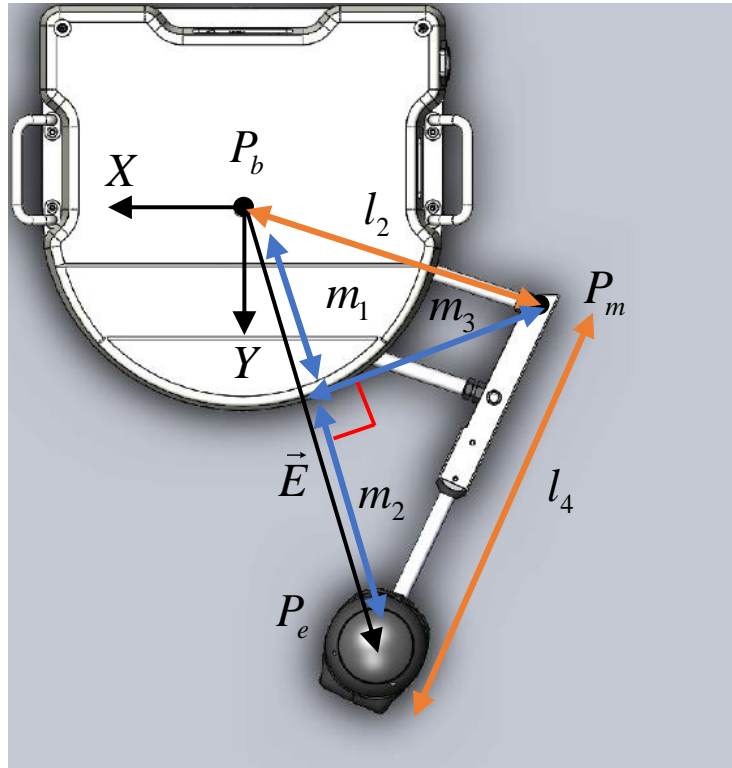


Figure 3.2: Stroke rehabilitation robot kinematics

$$\begin{aligned}
 |E| &= \sqrt{\vec{E} \cdot \vec{E}} \\
 m_1 + m_2 &= |E|
 \end{aligned}
 \tag{3.3}$$

Using the Pythagorean theorem in the two right triangles:

$$\begin{aligned}
m_1^2 + m_3^2 &= l_2^2 \\
m_2^2 + m_3^2 &= l_4^2 \\
m_1^2 - m_2^2 &= l_2^2 - l_4^2 \\
m_1 - m_2 &= \frac{l_2^2 - l_4^2}{|E|} \\
m_1 + m_2 &= |E| \\
m_1 &= \frac{\frac{l_2^2 - l_4^2}{|E|} + |E|}{2} \\
m_2 &= |E| - \frac{\frac{l_2^2 - l_4^2}{|E|} + |E|}{2} \\
m_3 &= l_2^2 - m_1^2 \\
\hat{n}_x &= \frac{\vec{E}}{|E|} \\
P_m &= P_b + m_1 \hat{n}_x + m_3 (\hat{Z} \times \hat{n}_x)
\end{aligned} \tag{3.4}$$

Finally, the robot angles were calculated as follows:

$$\begin{aligned}
\theta_1 &= \tan^{-1} \frac{P_e(2) - P_m(2)}{P_e(1) - P_m(1)} \\
\theta_2 &= \tan^{-1} \frac{P_m(2) - P_b(2)}{P_m(1) - P_b(1)}
\end{aligned} \tag{3.5}$$

3.2.4 Robot Dynamics

The high-fidelity robot model was designed by a previous research [47] in MapleSim™, which takes advantage of graph-theoretic modeling approaches and symbolic computing to produce fast simulations. This model is utilized for control algorithm testing in simulation. The control-oriented model was developed by setting friction and joint stiffness terms to zero in MapleSim. This model was used in all the model-based controllers. The robot's mass, inertia, and center of gravity (CG) locations were required for modeling and is shown in Table 3.3.

Table 3.3: Link Mass and Moment of Inertia

link i	mass (kg)	moment of inertia (kg.m ²)	CG X	CG Y
1	$m_1=2.578$	$J_1=0.022$	$x_1=-0.005$	$y_1=0$
2	$m_2=3.399$	$J_2=0.061$	$x_2=0.001$	$y_2=0$
3	$m_3=0.062$	$J_3=0.001$	$x_3=0.158$	$y_3=0$
4	$m_4=1.083$	$J_4=0.010$	$x_4=0.274$	$y_4=0.008$

High-Fidelity Model

The high-fidelity model of the robot is as follows:

$$M_R \ddot{\theta}_R + C_R \dot{\theta}_R + K_R(\theta_R - \theta_{R_0}) = -J_R^T f_{RE} - f_{RJ} + \tau_R \quad (3.6)$$

where M_R is the inertia matrix, C_R is the Coriolis or centrifugal matrix, and K_R is the joint stiffness matrix. f_{RE} and f_{RJ} are end-effector and joint frictions, respectively. Friction terms were modeled using a continuous-velocity friction model [26]. Finally, $\tau_R \in \mathbb{R}^2$ is the robot motor torque. The motor current was then obtained by using $I = \tau_R / K_T$ where K_T is the DC motor torque constant.

Control-Oriented Model

The control-oriented model of the robot was derived by omitting friction and joint-stiffness terms in Eq. 3.6. The model is shown in Eqs 3.7-3.10. The parameters in Eq. 3.10 are shown in Table 3.3.

$$M_R \ddot{\theta}_R + C_R \dot{\theta}_R = \tau_R \quad (3.7)$$

$$M_R(q) = \begin{bmatrix} \alpha_1 & \alpha_2 c_{12} + \alpha_3 s_{12} \\ \alpha_2 c_{12} + \alpha_3 s_{12} & \alpha_4 \end{bmatrix} \quad (3.8)$$

$$C_R(q, \dot{q}) = \begin{bmatrix} 0 & (\alpha_2 s_{12} - \alpha_3 c_{12}) \dot{q}_2 \\ (\alpha_3 c_{12} - \alpha_2 s_{12}) \dot{q}_1 & 0 \end{bmatrix} \quad (3.9)$$

$$\begin{aligned}
c_{12} &= \cos(q_1 - q_2) \\
s_{12} &= \sin(q_1 - q_2)
\end{aligned}$$

$$\begin{aligned}
\alpha_1 &= (x_1^2 + y_1^2)m_1 + m_3l_1^2 + (x_4^2 + y_4^2)m_4 + J_1 + J_4 \\
\alpha_2 &= m_3l_1x_3 + m_4l_2x_4 \\
\alpha_3 &= m_3l_1y_3 - m_4l_2y_4 \\
\alpha_4 &= (x_2^2 + y_2^2)m_2 + m_4l_2^2 + (x_3^2 + y_3^2)m_3 + J_2 + J_3
\end{aligned} \tag{3.10}$$

Model Validation

To validate the control-oriented model, a PD controller was applied to this model and the high-fidelity model for tracking a piecewise constant reference trajectory. Both models were extracted from MapleSim and exported to MATLAB for controller implementation. The results in Fig. 3.3 show good agreement between the control-oriented and high-fidelity models.

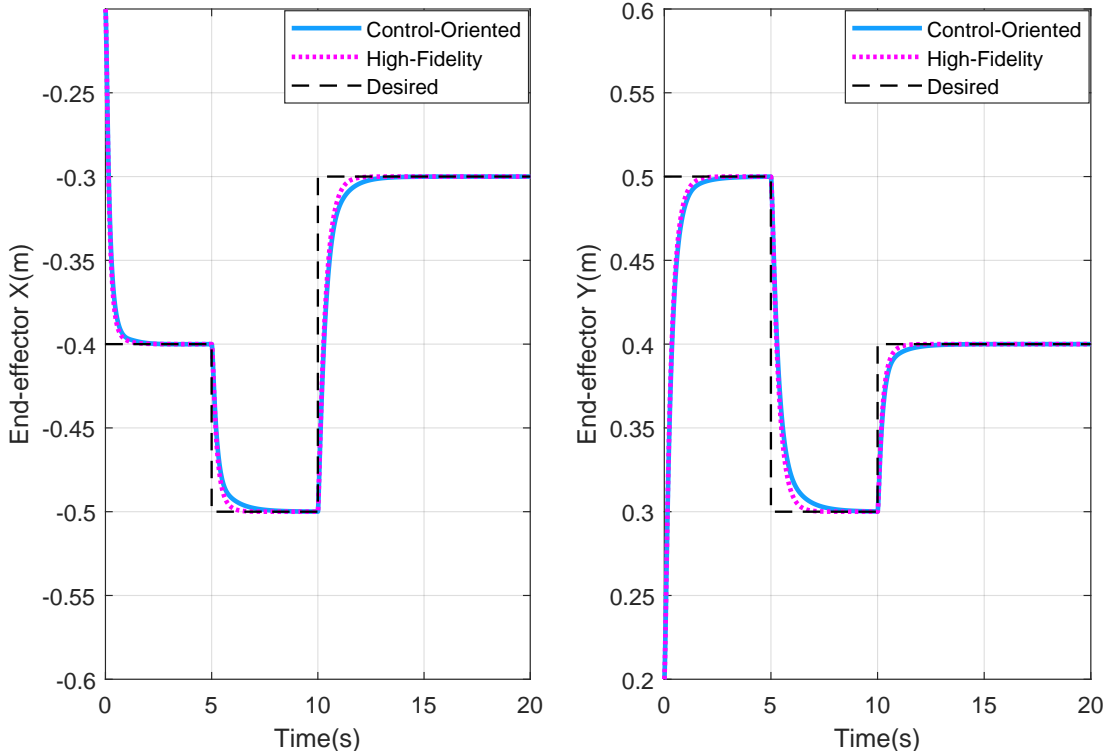


Figure 3.3: Control-oriented model validation. The end-effector X and Y positions in the workspace are shown.

3.3 Human Arm Model

In the subsequent chapter, the trajectory planning of the integrated human-robot interaction model was examined. To this end, a model of the human arm was required. This section examines a planar arm model, developed by [49]; this model is a two-dimensional 2 DOF linkage (Fig. 3.4); it includes one degree of freedom for elbow flexion/extension and one degree of freedom for shoulder rotation. The relation between the human arm joint and the hand position in the work space is shown below:

$$\begin{aligned} \dot{P}_A &= J_A \dot{\theta}_A \\ \ddot{P}_A &= \dot{J}_A \dot{\theta}_A + J_A \ddot{\theta}_A \end{aligned} \quad (3.11)$$

where P_A is the hand position, J_A is the human arm geometric Jacobian matrix, and

$\theta_A \in \mathbb{R}^2$ is the column matrix of human arm elbow and shoulder joint angles. The equation of motion (EOM) of this mechanism is shown in Eq. 3.12.

$$M_A \ddot{\theta}_A + C_A \dot{\theta}_A = \tau_A(u_a) \quad (3.12)$$

In this equation, M_A is the arm inertia matrix, and C_A is the arm Coriolis matrix. $\tau(u_a) \in \mathbb{R}^2$ is the arm joint torque. The arm model is muscle-driven and the $\tau(u_a)$ is a function of muscle activations u_a .

For the muscle dynamics, we used a 2D muscle model developed by [49] which was derived from a 3D model; 29 muscles of this model were lumped into 6 final muscles, namely shoulder and elbow mono-articular and bi-articular muscles. The Thelen muscle model was utilized to represent Hill-type muscle dynamics [134].

3.4 Human-Robot Interaction Model

The human arm interacts with the robot by grabbing the end-effector, and the robot performs rehabilitation practices in the horizontal plane. A passive revolute joint on the end-effector was utilized to integrate the two systems by using the interaction force F_I between the robot and the human arm:

$$M_R \ddot{\theta}_R + C_R \dot{\theta}_R + K_R(\theta_R - \theta_{R0}) = -J_R^T f_{RE} - f_{RJ} + \tau_R - J_R^T F_I \quad (3.13)$$

$$M_A \ddot{\theta}_A + C_A \dot{\theta}_A = \tau_A(u_a) + J_A^T F_I \quad (3.14)$$

Using a kinematic constraint (Eq. 3.15) to equate the position of the end-effector and the arm, and by equating the internal interaction force, the final human-robot interaction model is presented in Eq. 3.16. The robot angles θ_R and angular velocities $\dot{\theta}_R$ were considered as the states and the robot torques τ_R and human arm muscle activations u_a were considered as the inputs.

$$P_R = P_A \quad (3.15)$$

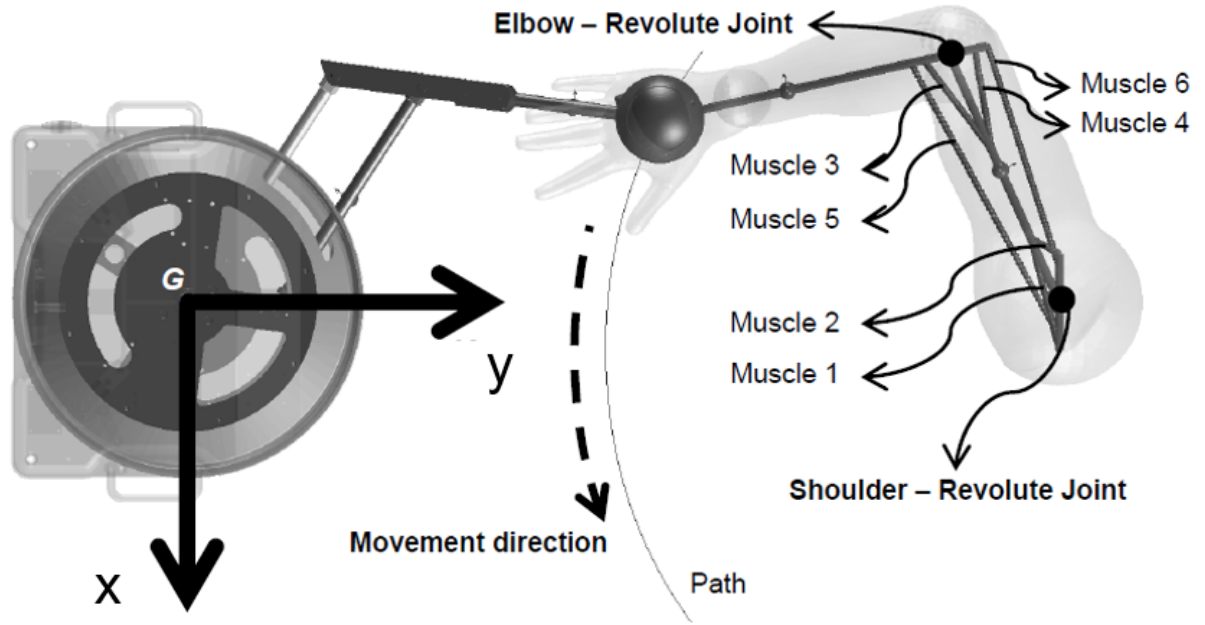


Figure 3.4: Human-robot interaction model [49]

$$\tau_A(u_A) + J_A J_R^{-T} (\tau_R - [M_R \ddot{\theta}_R + C_R \dot{\theta}_R + K_R (\theta_R - \theta_{R_0}) + f_{RJ} + J_R^T f_{RE}]) - M_A \ddot{\theta}_A - C_A \dot{\theta}_A = 0$$

$$\text{states: } x = \begin{bmatrix} \theta_{R_1} \\ \theta_{R_2} \\ \dot{\theta}_{R_1} \\ \dot{\theta}_{R_2} \end{bmatrix} \in \mathbb{R}^4$$

$$\text{inputs: } u = \begin{bmatrix} \tau_R \\ u_a \end{bmatrix} \in \mathbb{R}^8$$
(3.16)

3.5 Conclusion

The kinematic and dynamic model of the robot was presented in this chapter. Friction and joint stiffness terms were removed to derive the control-oriented dynamic model; this model was validated by applying a PD controller to it and comparing the results with the

high-fidelity model. We also provided the human arm and the final human-robot interaction model which was utilized in the next chapter in our planner optimization formulation.

Chapter 4

Human-Robot Interaction Trajectory Planning

4.1 Overview

In recent years, the primary focus within the field of rehabilitation robotics has been hardware design and control [68]. The trajectories, which are the paths followed by the patient's upper/lower limb for rehabilitation exercises, are usually chosen heuristically or manually by general physiotherapy principles [151, 148]. In this chapter, we discuss our approach to the trajectory planning for the HRI model presented in the previous chapter. Briefly, an optimization-based framework was used to systematically calculate the optimal manipulator trajectory for upper limb planar practices; this framework will benefit clinicians as well as rehabilitation robot specialists to set the exercises. Additionally, a trajectory stabilizer was designed to keep the robot on the desired trajectory. The integration of the planner and stabilizer leads to closed-loop tracking with optimal trajectories.

4.2 Trajectory Planning

Trajectory planning is an essential consideration in HRI research [95, 152]. The goal is to obtain some desired robot trajectory that is both feasible for the controller and safe for the human operator. The latter feature becomes even more critical in rehabilitation robotics due to the sensitive characteristics of the problem and patient involvement; an inappropriate rehabilitation trajectory exacerbates the patient's condition and might even

lead to aggressive robot torques. Previous research on model-based rehabilitation/assistive robot trajectory generation has either not considered human dynamic/biomechanical models [152, 119, 141, 125], or has used kinematic level models only [131]. Another line of research has focused solely on trajectory planning for the upper/lower extremity. Here, the research aim is to derive mathematical models that sufficiently describe natural human movements [14]; this approach typically involves consideration of the human skeletal dynamics [103]. Papers that have studied trajectory planning by including muscle dynamics in their human model have utilized only first and second-order approximate muscle models [14, 13]. The primary focus of the aforementioned research area is on human movements and hence, the robot is excluded in the models designed and employed for trajectory planning. Moreover, few studies in this area of literature have taken advantage of optimization techniques for trajectory planning [95, 141, 125, 64]. For instance, extensive research has been conducted on trajectory interpolation approaches [41, 107], in which the target points of the trajectory are set manually by rehabilitation robot specialists and then approximation functions are defined to interpolate between target points.

4.2.1 Problem Formulation

For the trajectory planning phase, the direct-collocation method was used to address the ensuing optimal control problem. In this method, the problem, including the cost function, dynamics (human-robot interaction model presented in Section 3.4), and constraints, are discretized; the discrete optimization problem is subsequently solved, resulting in trajectory points. Interpolation approaches are then exploited to come up with the final continuous trajectories. This is different from using trajectory interpolation as in [41, 107], since the target points in our approach are the output of the discrete optimization, rather than manual setting of the robot specialists. The trajectory planning framework runs offline since no online update is required. Hence, computational issues are less important in this phase. In order to formulate and solve the optimization, **OptimTraj**, an open-source optimization library in MATLAB developed by Kelly [65], was used. This library is capable of solving C2 continuous (smooth) problems with linear and/or nonlinear cost/constraint functions. More specifically, problems with continuous dynamics, boundary constraints, path constraints, integral costs, and boundary costs are under the branch of problems this library can solve. OptimTraj also supports the provision of analytic gradients, with respect to the cost function, for improved convergence times, hence warranting the use of symbolic gradients as produced by Maple®.

Four optimization methods are available in the OptimTraj library: Trapezoidal Direct Collocation, Hermite-Simpson Direct Collocation, Runge-Kutta 4th Order Multiple Shoot-

ing, and Chebyshev-Lobatto Orthogonal Collocation. A GPOPS wrapper is also provided if the user wishes to take advantage of adaptive mesh refinement; in this approach, variable mesh size and approximate polynomial degree is provided [108, 77]. Once the problem has been transcribed according to the methods listed prior, OptimTraj calls MATLAB's *fmincon* as the NLP solver; the interior-point method is used within *fmincon* to solve the final discretized optimization. In order to evaluate the performance, OptimTraj then reports the total time required for NLP (`nlpTime`), the optimum value function (`ObjVal`), the integral of collocation constraint error at each segment (`error`), and the maximum of this error during the entire duration (`maxError`).

In the current study, the Trapezoid transcription method was chosen for problem discretization. In this method, the time histories of the states and control inputs (over a given horizon) are approximated as quadratic splines and linear splines respectively. In other words, the controls and dynamics are assumed to be linear between grid (collocation) points. For the discretized version to be a sufficient representation of the continuous system, the derivative of the continuous and discretized systems must match at each grid point; this is how the corresponding polynomials are constructed. After the nonlinear program is solved at the collocation points, spline approximations are used to find the continuous state-input trajectories. The discrete optimization method is shown below:

$$\begin{aligned} \min_{x_0 \dots x_N, u_0, \dots, u_N} \quad & \sum_{N_0}^{N-1} \frac{h_k}{2} (J_k + J_{k+1}) \\ & h_k = t_{k+1} - t_k \end{aligned} \tag{4.1}$$

subject to:

$$\begin{aligned} x[0] &= x_0 \\ x[N] &= x_f \end{aligned} \tag{4.2}$$

collocation constraints(dynamics):

$$\frac{h_k}{2} (f_{k+1} + f_k) = x_{k+1} - x_k \quad k = 1 \dots N \tag{4.3}$$

inequality constraints:

$$\begin{aligned} x_{min} &\leq x_{i,k} \leq x_{max} & i = 1 \dots n_s & \quad k = 1 \dots N - 1 \\ u_{min} &\leq u_{i,k} \leq u_{max} & i = 1 \dots n_u & \quad k = 1 \dots N - 1 \end{aligned} \quad (4.4)$$

integration:

$$\text{For } t_k \leq t \leq t_{k+1} \quad k = 1 \dots N - 1$$

control (linear spline):

$$\begin{aligned} u(t) &= u_k + (t - t_k)\beta & k = 1 \dots N - 1 \\ \beta_k &= \frac{-1}{h_k}(u_k - u_{k+1}) \end{aligned} \quad (4.5)$$

state (quadratic spline):

$$\begin{aligned} x(t) &= x_k + (t - t_k)f_k + (t - t_k)^2\gamma_k & k = 1 \dots N - 1 \\ \gamma_k &= \frac{-1}{2h_k}(f_k - f_{k+1}) \end{aligned}$$

where J is the cost function, h_k is the stage time-step, x is the HRI state, and u is the HRI input. For the cost function J , robot inputs and input rates along with muscle activation and arm joint accelerations were selected. Robot inputs were chosen to reduce the DC motor currents required for torque generation; this effect improves hardware durability. Robot input rates are also correlated with trajectory smoothness and patient comfort, hence their inclusion. The last two terms were adopted from [13], in which the authors used inverse optimal control to find the cost function minimized by the central nervous system in upper-arm reaching movements. It was suggested that in this case, minimization of mechanical energy expenditure and human arm joint acceleration best describes human-like motor behaviours. Using this hybrid human-robot cost function is a novel approach in optimization-based rehabilitation robot planners and is an advance in determining the optimal trajectory. The mathematical representation of the cost function is shown in Eq. 4.6. τ_R is the robot input torque, and u_a is the arm muscle activation. The cost function itself is quadratic since the squared version of each term is set. However, the nonlinear dynamic constraint in Eq. 4.3 makes the overall optimization non-convex and an NLP.

Boundary constraints were defined on states and inputs (Eq. 4.4), which represented the joint-space limits of the robot angles, hardware limits of robot motors, and physiological limits of human arm musculature (Table 4.1). The final time value was set to 2 seconds;

this was approximately the required time to finish the trajectory. Moreover, the initial and final end-effector positions were set such that internal rotation of the shoulder joint was prohibited; this movement has reportedly been ineffective in patient motor recovery. These positions were mapped to robot initial and final joint-space states by using inverse kinematics.

Table 4.1: Optimization Inequality Constraints

Bounds	Minimum	Maximum
joint angle (θ_R)	-5 rad	5 rad
joint angular velocity ($\dot{\theta}_R$)	-10 rad/s	10 rad/s
robot torque (τ_R)	-10 N.m	10 N.m
muscle activation (u_A)	0	1

$$J = \sum_{k=1}^{N-1} (W_1 \tau_R^2(t_k) + W_2 \dot{\tau}_R^2(t_k)) + W_3 u_A^2(t_k) + W_4 \ddot{\theta}_A^2(t_k) h_k \quad (4.6)$$

Also seen in Eq. 4.6 are the weights associated with each term: W_1 , W_2 , W_3 , and W_4 . These are notable as they can be varied as the rehabilitation period progresses. For example, less weight might need to be provided to muscle activations as the patient’s motor function improves and practices transition from passive to active rehabilitation. In this case, the patient is able to apply more muscle activation, and thus the optimization should put less emphasis on minimizing this value such that the new trajectory is well-suited to the patient’s current status. The weight adjustment can be done by testing multiple weight settings and comparing the experimental and simulated muscle activations.

4.2.2 Simulation Results

The offline NLP computational time was 86.122 seconds. The desired end-effector path is shown in Fig. 4.1. Also, the calculated robot joint trajectories along with the open-loop inputs are presented in Figs. 4.2 and 4.3. The initial spike in the inputs is necessary to overcome the robot inertia at the beginning of the trajectory. Note that as angular velocity increases in the joints, less torque is required from the motor for trajectory-tracking.

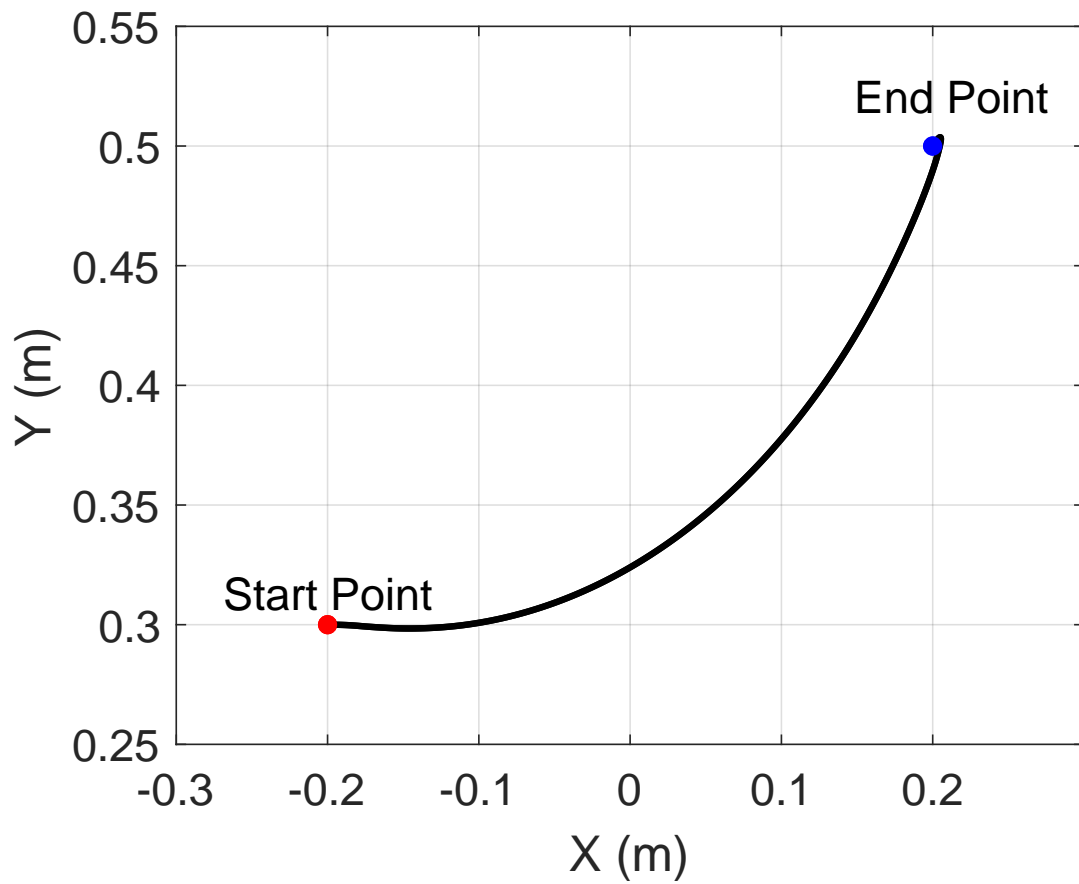


Figure 4.1: End-effector desired path in the workspace.

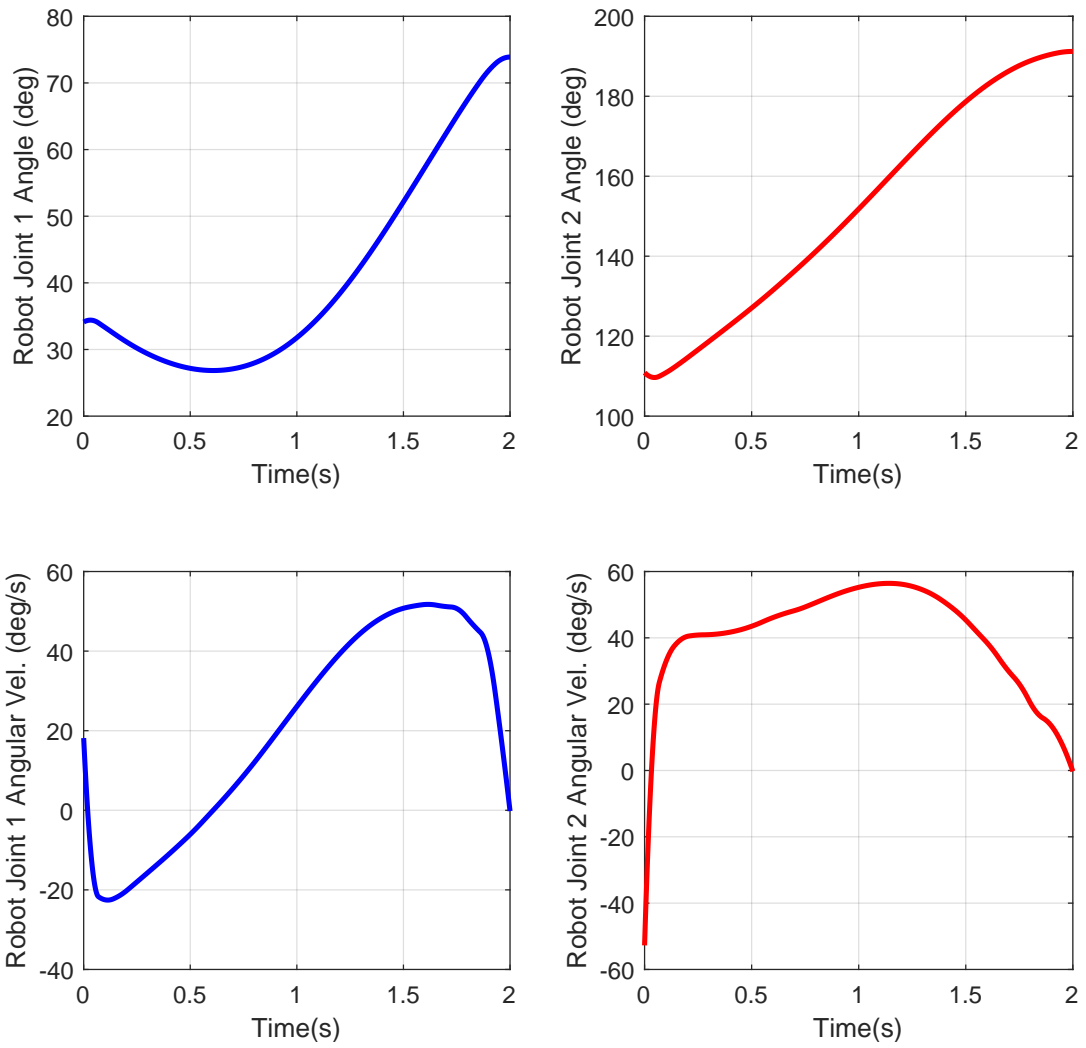


Figure 4.2: Robot state trajectories in the jointspace.

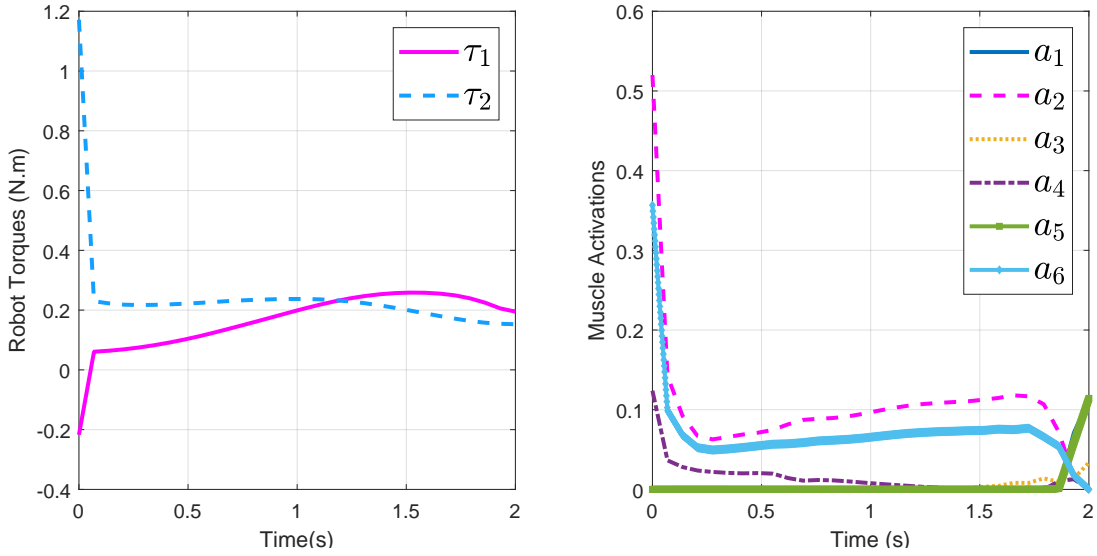


Figure 4.3: Input trajectories. The left plot depicts the robot torques and the right plot shows the muscle activations.

The collocation error for the robot joint angles was used to estimate how much accuracy was lost after the discretization; this was done by checking whether the generated solution satisfied the system dynamics. The error is defined as calculating the difference between the derivative of our state trajectory after interpolation and the collocation (dynamic) constraints between the grid points:

$$\epsilon(t) = \dot{x}(t) - f(x(t), u(t)) \quad (4.7)$$

The error estimate for each optimization segment was acquired by integrating the collocation error at each segment:

$$\eta_k = \int_{t_k}^{t_{k+1}} \epsilon(\tau) d\tau \quad (4.8)$$

As most interest is in regard to the joint angles, it is important to note that the segment error is within the acceptable range, which is 2 deg/s. Fig. 4.4 plots the errors for the robot joint angles.

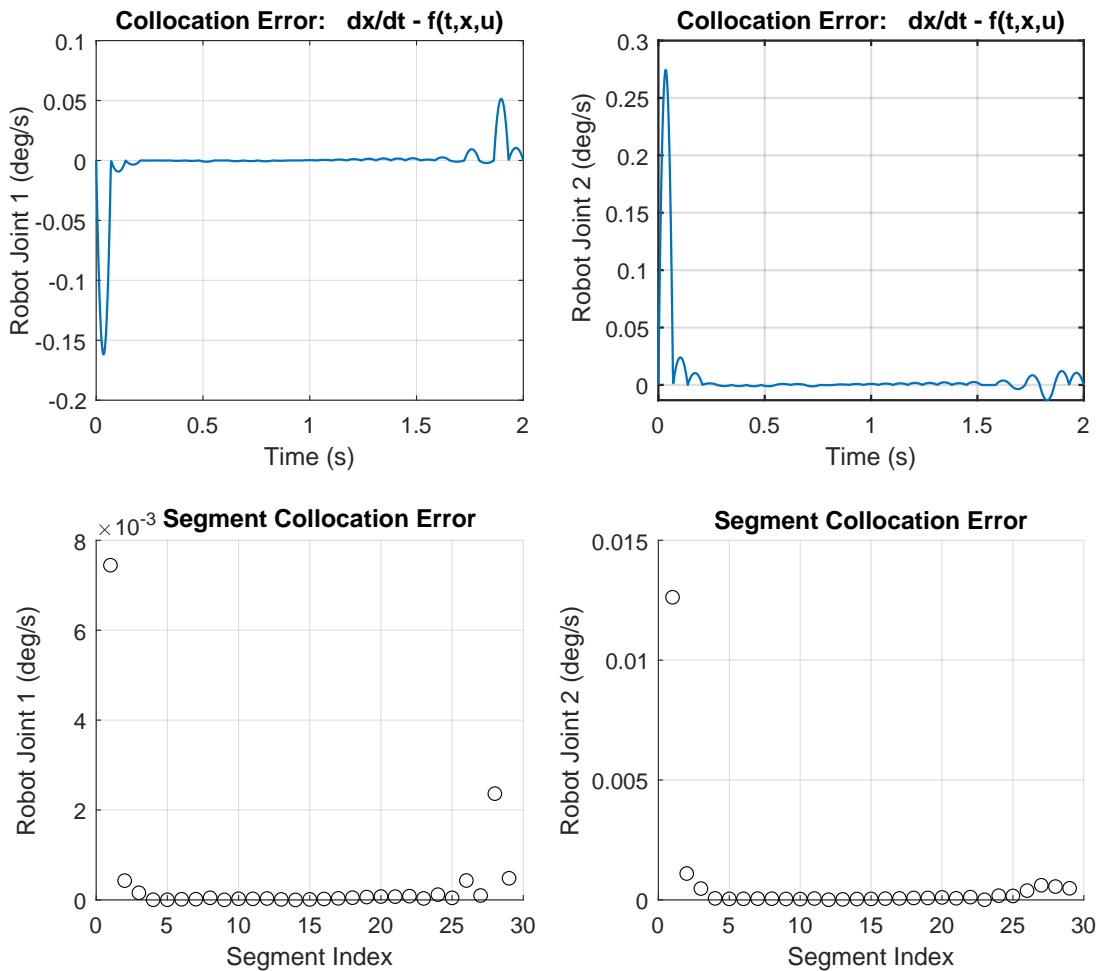


Figure 4.4: Trajectory error results. The top row highlights the collocation error between the real dynamics (dx/dt) and the dynamics (f) at the collocation points used. It depicts the resulting error due to the discretization. As shown, this error is considerably low which suggests that the application of the direct-collocation was successful.

4.3 Trajectory Stabilization

Under ideal conditions, the motion planner discussed in Section 4.2 would provide the desired human/robot state and control trajectories; the former pertaining to the rehabilitation practice/exercise kinematics and the latter being the open-loop inputs necessary

to drive the system’s motors. However, this approach cannot work in a practical (experimental) setting. Concerns such as unmodeled dynamics, disturbances, and sensor noise would degrade the performance of an open-loop controller. On top of that, there is no guarantee that the patient applies the same muscle activations as given by the open-loop values. Consequently, a trajectory stabilizer is required.

The online use of the HRI model for the implementation of the stabilizer on the rapidly-updated robot dynamics would cause computational issues. Instead, we only considered the robot model online and neglected the human arm model for the stabilizer; the stabilizer calculated the robot torques, and the patient contribution was recorded by force sensor data. We should mention that both using the robot model and the HRI model online result in sub-optimal trajectories, because the inputs will not be exactly the same as the simulated planner results.

4.3.1 Linear Quadratic Regulators (LQR)

In the current thesis, an LQR algorithm was utilized as the trajectory stabilizer for closed-loop control during rehabilitation robot operation. LQR controllers are one of the most effective existing optimization-based algorithms and have been implemented in many control fields including robotics [126], autonomous cars [102], and aerospace [78]. Briefly, this approach uses indirect optimization methods and calculus of variations to solve the infinite-horizon problem shown in Eq. 4.10, where Q (semi-positive definite) and R (positive definite) are weight matrices applied to the states and inputs respectively. The same feedback control solution can be found by using continuous dynamic programming equation, also known as the **Hamilton-Jacobi-Bellman** (HJB) equation (Eq. 4.11), with a quadratic cost function:

$$J^*(\tilde{x}) = \tilde{x}^T S \tilde{x}, \quad S = S^T \succeq 0 \quad (4.9)$$

where \tilde{x} is the state error and S is a positive definite matrix. The application of this cost in the HJB results in an algebraic Riccati equation (Eq. 4.12) from which the optimal S can be obtained:

$$J = \int_0^\infty [\tilde{x}^T Q \tilde{x} + u^T R u] dt, \quad Q = Q^T \succeq 0 \quad R = R^T \succ 0 \quad (4.10)$$

$$\forall x, \quad 0 = \min_u [\tilde{x}^T Q \tilde{x} + u^T R u + \frac{\partial J^*}{\partial x}](Ax + Bu) \quad (4.11)$$

$$0 = SA + A^T S - SBR^{-1}BTS + Q \quad (4.12)$$

where A and B are linear dynamics matrices. The solution of the Riccati equation leads to the feedback control solution:

$$u = -R^{-1}BTSx = -K\tilde{x} \quad (4.13)$$

A caveat of LQR is that it only works for linear system dynamics. Hence, the application of this algorithm to nonlinear systems requires linearization of the dynamics $f(\cdot)$ about the desired points as demonstrated in Eq. 4.14.

$$\begin{aligned} \dot{\tilde{x}} &= \left. \frac{\partial f}{\partial \tilde{x}} \right|_{x^*} \tilde{x} + \left. \frac{\partial f}{\partial u} \right|_{u^*} u \\ \left. \frac{\partial f}{\partial \tilde{x}} \right|_{x^*} &= A(t) \\ \left. \frac{\partial f}{\partial u} \right|_{u^*} &= B(t) \end{aligned} \quad (4.14)$$

Generally, this linearization is only valid in the vicinity of the desired state and input. This vicinity is often called *the region of attraction*, outside of which the LQR will not perform well. Often times, use of additional controllers is necessary to bring the system to its region of attraction where the LQR can be activated. Research using Lyapunov-based methods has estimated the region of attraction for underactuated robotic systems [132]. Planar mechanisms, however, possess the robust characteristic of having an infinite region of attraction within their workspace [132]. Since the gravity is counterbalanced by the surface, any point can be chosen for linearization and it will be valid. Our rehabilitation robot shares this same feature. The dynamics of the robot was thus symbolically linearized using Maple®. The resulting time-dependent symbolic linear dynamic matrices $A(t)$, $B(t)$, $C(t)$, and $D(t)$ were then exported to MATLAB code. This code was structured such that the desired trajectories could be substituted in the symbolic matrices iteratively to produce the online matrices. The schematics of the integrated planner and LQR controller can be found in Fig. 4.5. Recall that the human dynamics was not considered in the linearized model to improve computation speeds in the controller's online implementation.

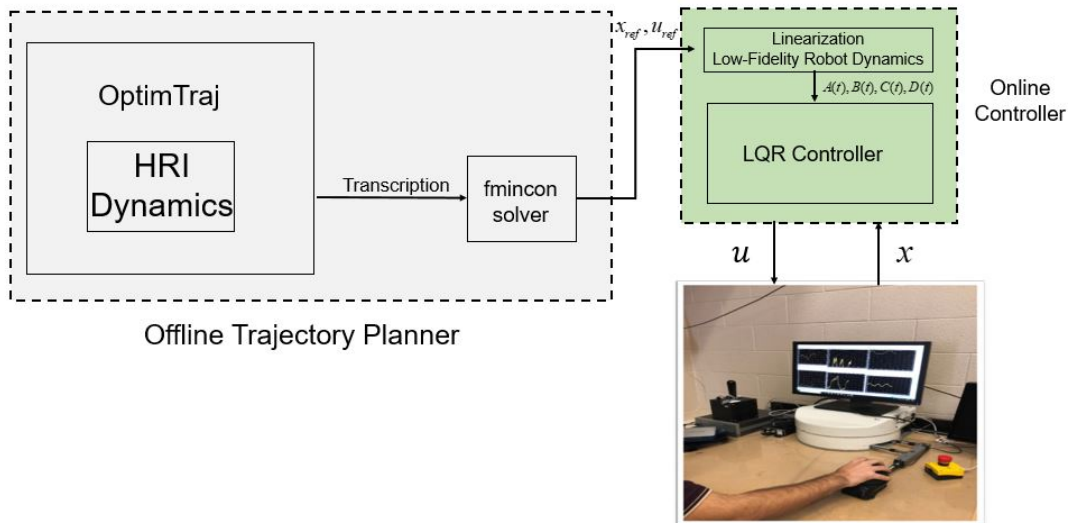


Figure 4.5: Schematics of integrated planner and controller. The planner runs offline and provides the desired values that the controller uses to stabilize the trajectory. The resulting robot torques are then applied to the robot manipulator to advance its state forward in time.

4.3.2 Simulation Results

The LQR trajectory stabilization results are shown in Fig. 4.6. The controller was able to successfully stabilize the trajectory at position and velocity levels. Note that due to excluding human muscle activations within this portion of thesis, the input robot torques required were higher than the open-loop torques.

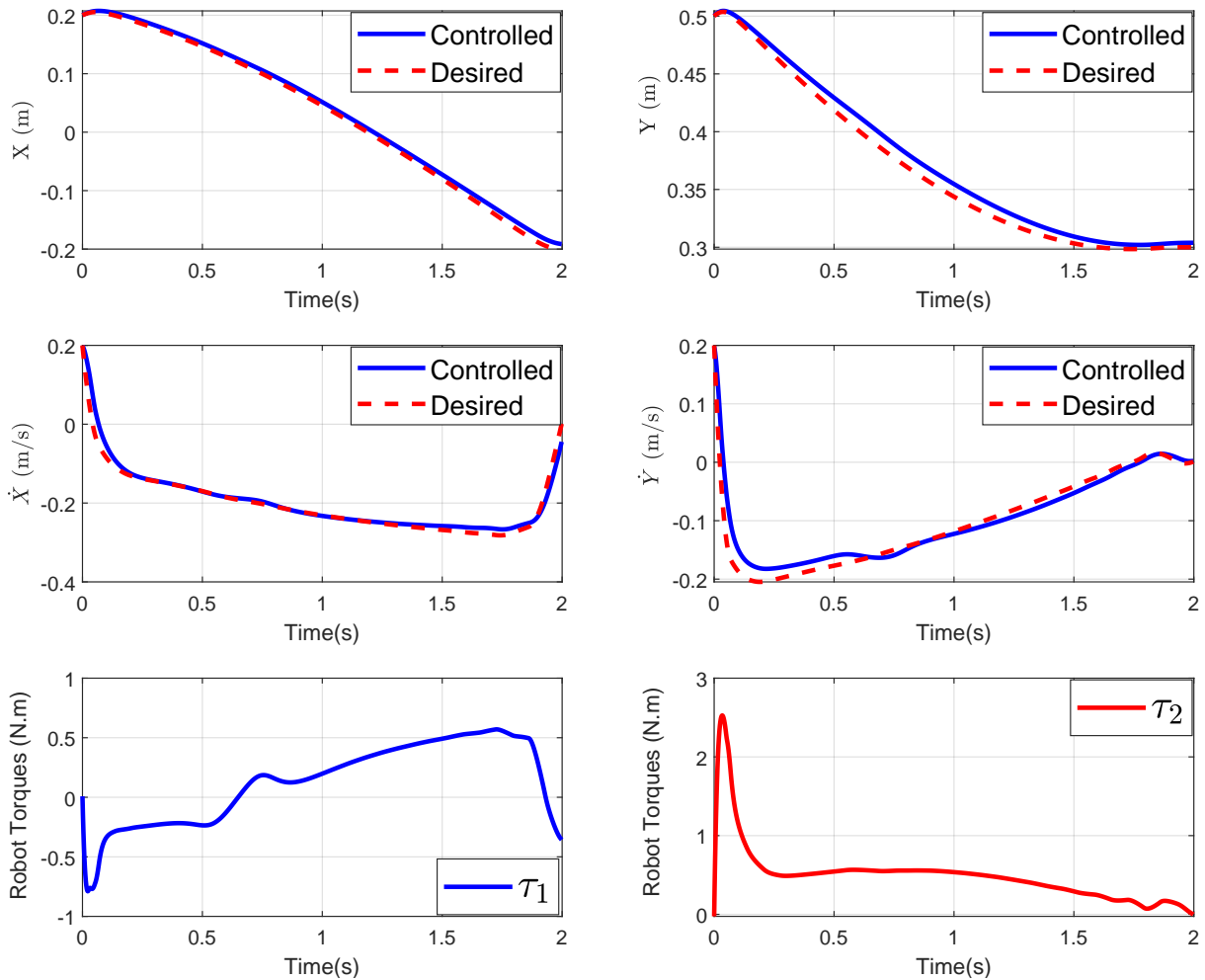


Figure 4.6: LQR control results in simulation. X and Y are end-effector positions. \dot{X} and \dot{Y} are end-effector velocities. Also, τ_1 and τ_2 are robot motor torques.

4.4 Experimental Implementation

For the final portion of this chapter, the controller was physically implemented on the rehabilitation robot hardware. Note that the user was not integrated in the loop for this section. A secondary LQR was first utilized to move the end-effector to the start point of the planned trajectory. The controller was then switched to the primary LQR which handled tracking of the end-effector along the manipulation trajectory. For con-

troller switching, a mechanism was necessary to turn the secondary LQR off as an 1 cm position-error threshold was reached. Although this switching can be readily implemented using MATLAB functions or condition blocks in Simulink, problems were encountered: activation of the primary LQR shifts the end-effector away from the start, which reactivates the secondary LQR. To avoid this process, the system’s behaviour was set such that the secondary LQR deactivates once the threshold is met, staying deactivated from then on. To this end, we took advantage of StateFlow blocks in Simulink, which are typically used in Finite-State machines. The LQR weights are tuned online to acquire the best trajectory tracking and acceptable input values; the weights are reported in Table 4.2. Fig. 4.7 plots the experimental end-effector position and velocity results. Although performance degradation was observed in the velocity level, the overall tracking was successful, which confirmed the potential for implementing the stabilizer with a subject.

Table 4.2: LQR Weights

Weights	Value
Q	Diag(9200,9200, 300, 300)
R	Diag(50,50)

4.5 Conclusion

In this chapter, we discussed the importance of having a systematic design approach in rehabilitation practice. We adopted an optimization-based trajectory planning process to obtain the optimal manipulation trajectory. An LQR controller was then designed to stabilize this trajectory. The results of simulation and experimental tests for LQR were positive showing successful movement of the end-effector along the given path. The subject is integrated in the loop in the last chapter by utilizing the force sensor data and calculating the patient torque contribution.

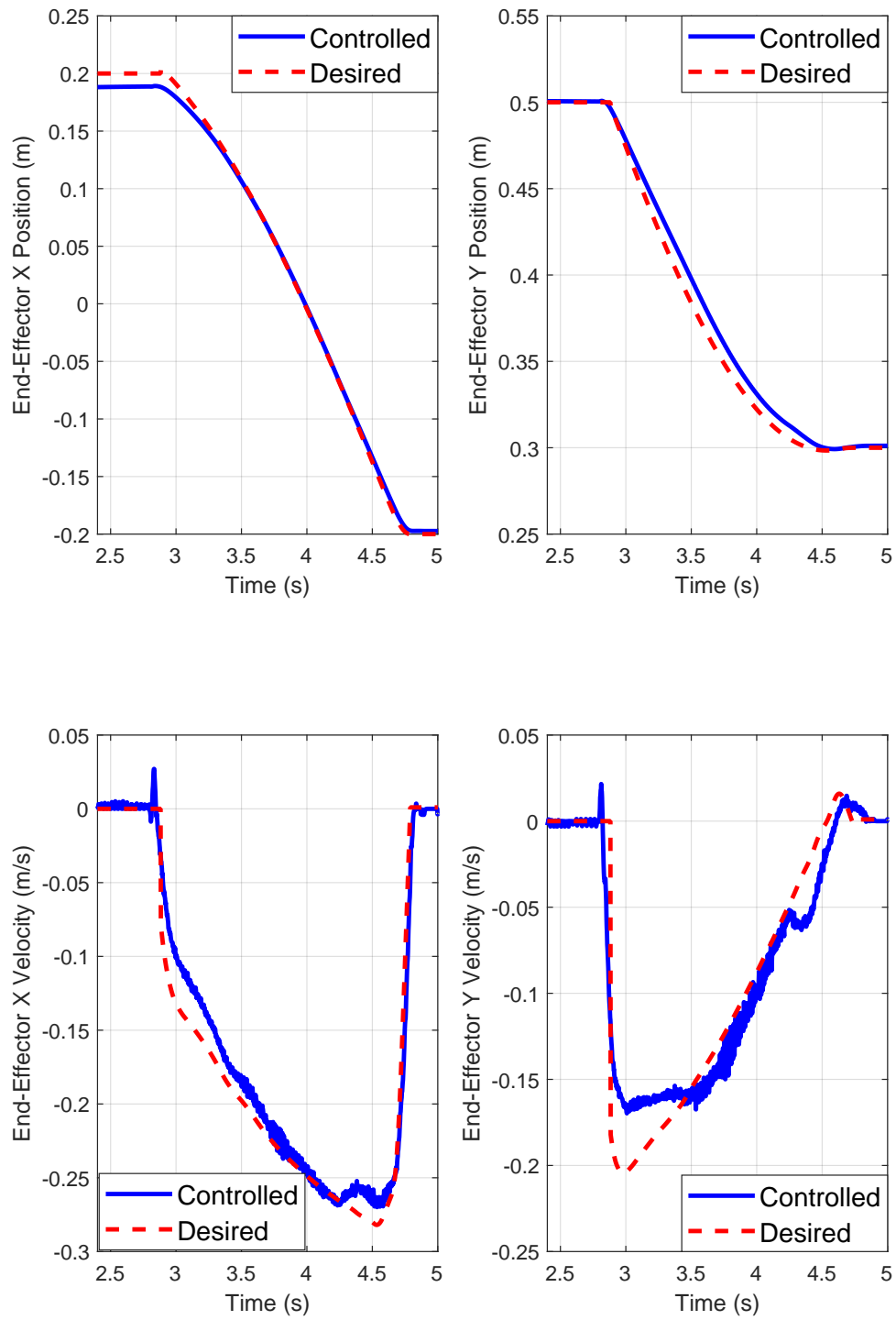


Figure 4.7: Experimental implementation of the trajectory stabilizer on the robot.

Chapter 5

Comparative Study of the Rehabilitation Robot Control Algorithms

5.1 Overview

One of the challenges of rehabilitation robotics is the choice of the control algorithm [89]. To the best of our knowledge, there is no published study on the comparison of multiple control algorithms on a rehabilitation robot. The purpose of this chapter was to evaluate the capabilities of multiple well-known controllers on our robot. These controllers were applied in simulation for a trajectory tracking problem. After designing a comparison criteria, the performance of each controller was assessed. The controllers were then tested on the robot hardware in real-time. Two of the best candidates were selected to be used with a human subject in subsequent chapters. Based on our particular investigation, general guidelines were proposed for selecting a controller for rehabilitation robots and assistive devices with similar DC actuators.

5.2 Controller Design

In this section, we discuss the features and formulation of the each control algorithm. Five controllers were proposed and tested on the robot:

1. Proportional-Integral-Derivative
2. Computed-torque Proportional-Integral-Derivative
3. Linear Quadratic Regulator
4. Sliding Mode Control
5. Nonlinear Model Predictive Control

Since we discussed the formulation of LQR in Chapter 4, we only focus on the other four controllers in this section.

5.2.1 Proportional-Integral-Derivative (PID) Controllers

PIDs are one of the most common algorithms in industry [10]. These model-free controllers are more computationally tractable than model-based approaches and are robust in most applications. The final control input is dependent only on the state-error and the weights, as demonstrated below:

$$e(t) = \theta_{R_d}(t) - \theta_R(t) \quad (5.1)$$

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (5.2)$$

where $e(t)$ is the error between the reference robot joint angle $\theta_{R_d}(t)$ and the measured robot joint angle $\theta_R(t)$. K_p , K_i , and K_d are the controller weights for finding the input $u(t)$. Though simple, there are some drawbacks associated with PID formulation:

- PID is completely reactive and does not include any predictive element; sudden disturbances and/or path change can degrade its performance.
- The weights have direct effect on the inputs; extra care is required to avoid high-input values in the weight tuning process.
- PID is often associated with high-overshoot results, which calls for extensive weight adjustment. Tuning approaches like Ziegler Nichols mitigate the direct weight effect on the input and overshoot problems to some extent [153].

- PID is usually used in an independent-joint approach, i.e. an independent controller is set for each robot joint. This method neglects the dynamic coupling of the system; reducing the state error in one joint may create a negative offset on the other one. The dynamic coupling problem is exacerbated in high-speed trajectory tracking.

The schematic of PID controller is shown in Fig. 5.1. The encoder angle values are used at each iteration within the controller.

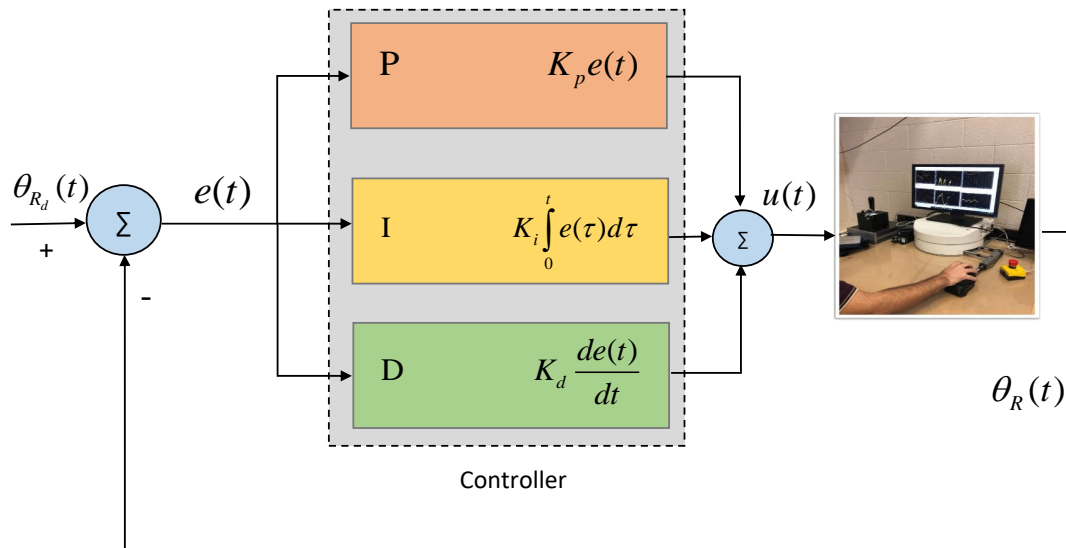


Figure 5.1: PID block diagram

5.2.2 Computed-Torque PID Controllers

To some degree, a PID's performance is improved by utilizing a feedforward term. In computed-torque PID, a model-based hierarchical controller structure is used; the nonlinear dynamics of the robot is cancelled by using the inverse dynamics in the inner-loop; the outer-loop feedback PID term is then utilized to correct for the errors of the resulting linear system. This approach is an anticipatory controller (instead of a totally reactive one) that has better performance characteristics. Instead of the coupled nonlinear system, PID is applied to a decoupled linear system.

The feedforward input term is shown in Eq. 5.3. u_{Model} was obtained by using the inverse dynamic model of the robot. The feedback PID term formulation in Eq. 5.4 is similar to

the previous section. The final control input is demonstrated in Eq. 5.5; applying this input to the nonlinear dynamics of the robot resulted in a linear control problem in the Eq. 5.6. Note that as the mismatch between the model of the system and the real dynamics increases, error terms are added to Eq. 5.6. The performance degrades drastically when the mismatch is high; that is to say, computed-torque PID is only applicable when an accurate model of the dynamical system is available. The schematic of this controller is depicted in Fig. 5.2.

$$u_{Model}(t) = M(\theta_R)\ddot{\theta}_{R_d}(t) + C(\theta_R, \dot{\theta}_R)\dot{\theta}_R(t) \quad (5.3)$$

$$u_{PID}(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (5.4)$$

$$\begin{aligned} u(t) &= M(\theta_R)(\ddot{\theta}_{R_d}(t) + u_{PID}(t)) + C(\theta_R, \dot{\theta}_R)\dot{\theta}_R(t) \\ &= u_{Model}(t) + M(\theta_R)u_{PID}(t) \end{aligned} \quad (5.5)$$

$$\ddot{e}(t) = u_{PID}(t) \quad (5.6)$$

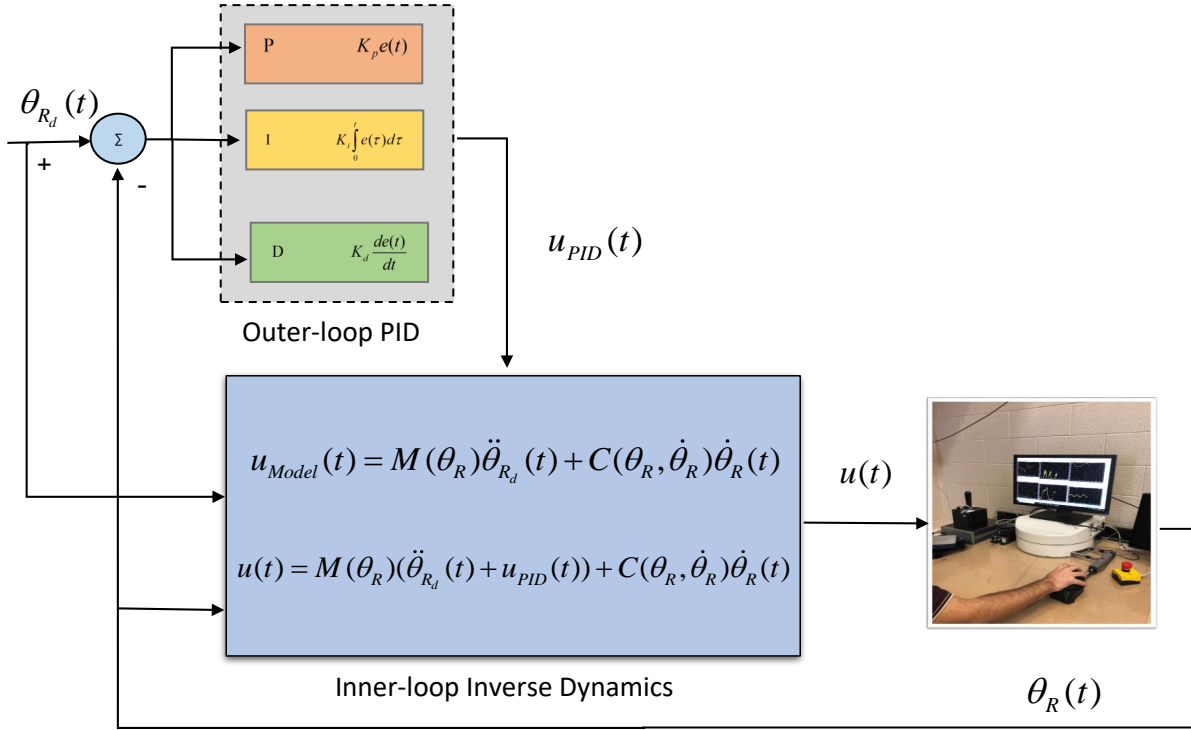


Figure 5.2: Computed-Torque PID block diagram

5.2.3 Sliding Mode Control (SMC)

SMC is a powerful nonlinear control method [123, 36], which permits the direct consideration of the uncertainty in control design, and therefore provides stronger robustness characteristics than other feedback control algorithms. The uncertainty is broadly divided into perturbations and robot plant uncertainty. In the scope of our research, perturbation is imposed by the patient. Moreover, plant uncertainties are caused by inaccurate model parameters which stem from errors in system parameter identification (SPI). These include any errors in the mass, length, and moments of inertia properties. Roughly speaking, patient perturbations are higher than the robot plant uncertainty. In general, SMC can alleviate the negative impact of uncertainties.

Formulation

The general form of the robot model is shown below, where $f(\theta_R, \dot{\theta}_R, t)$ is the dynamics model, and $u_R(t)$ is the input part. The disturbance term $d(t)$ is added to the equation for the sake of completeness.

$$\ddot{\theta}_R = f(\theta_R, \dot{\theta}_R, t) + d(t) + u_R(t) \quad (5.7)$$

A sliding surface S was defined as Eq. 5.9. $\epsilon(t)$ in Eq. 5.9 is the error between the desired joint angles and the measure joint angles, which converges to zero as $S = 0$ (Eq. 5.10). λ is a tunable weight and determines the speed of convergence.

$$\epsilon(t) = \theta_{R_d}(t) - \theta_R(t) \quad (5.8)$$

$$S = \frac{d}{dt}(\epsilon(t) + \lambda) = 0 \quad (5.9)$$

$$\begin{aligned} \dot{\epsilon}(t) &= -\lambda\epsilon(t) \\ \epsilon(t) &= e^{-\lambda(t-t_0)}\epsilon(t_0) \end{aligned} \quad (5.10)$$

In order to find the control input, the first derivative of the sliding surface was calculated. Since the control input appears in the first derivative of the sliding surface, it is said to have a *relative degree of one*. The resulting low-frequency input u_{LF} control input was then found by setting $\dot{S} = 0$:

$$\begin{aligned} \dot{S} &= f(\theta_R, \dot{\theta}_R, t) + d(t) + u(t) - \ddot{\theta}_{R_d}(t) + \lambda\dot{\epsilon}(t) \\ u_{LF}(t) &= -f(\theta_R, \dot{\theta}_R, t) - d(t) + \ddot{\theta}_{R_d}(t) - \lambda\dot{\epsilon}(t) \end{aligned} \quad (5.11)$$

On the other hand, uncertainties exist in the model dynamics and the disturbance model in Eqs. 5.12 and 5.13:

$$f(\theta_R, \dot{\theta}_R, t) = f_m(\theta_R, \dot{\theta}_R, t) + \Delta f(\theta_R, \dot{\theta}_R, t) \quad (5.12)$$

$$d(t) = d_m(t) + \Delta d(t) \quad (5.13)$$

where $f_m(\theta_R, \dot{\theta}_R, t)$ is the modeled part of the system dynamics and $\Delta f(\theta_R, \dot{\theta}_R, t)$ is the uncertainty term associated with system dynamics. Similarly, $d_m(t)$ is the modeled part of the disturbance and $\Delta d(t)$ is the uncertainty term of disturbance. A higher frequency term was added to the control input for robustness. The derivation of this term, which comes from Lyapunov stability theorem, is discussed in Appendix A:

$$u_{HF} = -K \text{sign}(S/\psi) \quad (5.14)$$

where K , and ψ are other tunable weights. It is worthwhile to mention that the discontinuous uncertainty input term u_{HF} may lead to a phenomenon called chattering, in which the input switches between positive and negative S values on the sliding surface. Chattering damages the hardware and should be circumvented. There are multiple methods to address this negative effect. We replaced the discontinuous term with the continuous \tanh function to solve this issue. Eq. 5.15 describes the final control input. The schematic of this controller is displayed in Fig. 5.3.

$$\begin{aligned} u_{HF} &= -K \tanh(S/\psi) \\ u_{SMC} &= u_{LF} + u_{HF} \end{aligned} \quad (5.15)$$

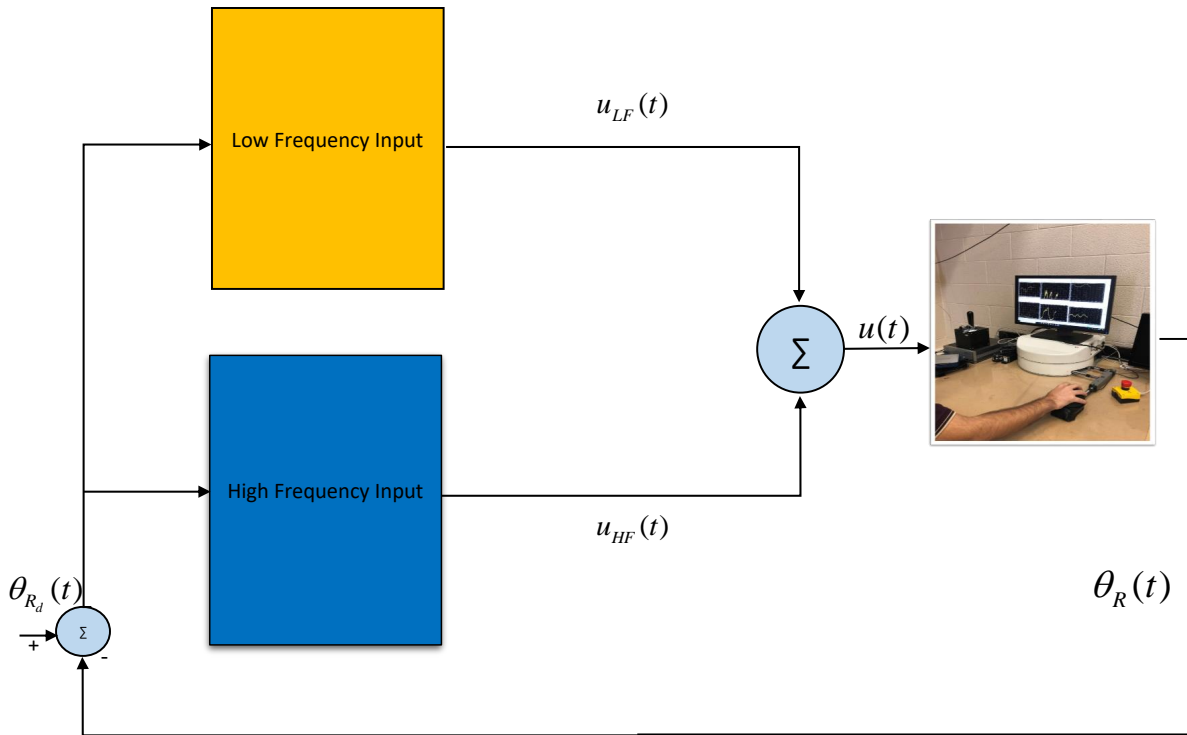


Figure 5.3: SMC block diagram

5.2.4 Nonlinear Model Predictive Control (NMPC)

Generally, all the aforementioned controllers lead to acceptable results in many applications. However, it is not possible to explicitly include constraints in their design. Constraints are imposed in almost all the control fields; motors should not surpass their power limit; a robotic arm's end-effector should stay in its workspace; an autonomous car should avoid obstacles in its path, and so on. To this end, MPC utilizes online constrained optimization. The nonlinear version of MPC (NMPC) allowed for the use of nonlinear dynamics in the optimization. Interestingly, the unconstrained version of MPC with quadratic stage-cost and linear model reduces to the LQR formulation.

Formulation

At each timestep, MPC solves a finite-horizon constrained optimal control problem to find the control inputs on the *Prediction Horizon*. Then, only the first input is applied

on the dynamics; the horizon moves forward and the same procedure repeats for the next timestep. This is known as *Receding Horizon* procedure, which adds the predictive element to MPC. Fig. 5.4 displays the general scheme of MPC.

The optimization problem is shown in Eqs. 5.16-5.20. H is the prediction horizon, $\ell(x(t), u(t), t)$ is the stage cost, and $\ell_T(x(H), H)$ is the terminal cost. The optimization included state-input inequality, dynamics, and initial condition constraints. Since obstacles were not considered in the trajectory, path constraints were not used.

$$\min_{X,U} \int_0^H \ell(x(t), u(t), t) dt + \ell_T(x(H), H) \quad (5.16)$$

$$u_{min} \leq u(t) \leq u_{max} \quad \text{for } t \in [0, H] \quad (5.17)$$

$$x_{min} \leq x(t) \leq x_{max} \quad \text{for } t \in [0, H] \quad (5.18)$$

$$\dot{x}(t) = f(x(t), u(t), t) \quad \text{for } t \in [0, H] \quad (5.19)$$

$$x(0) = x_0 \quad (5.20)$$

There are multiple numerical recipes for discretizing and solving the infinite-dimensional optimization problem of Eq. 5.16. A review on this matter was presented in Chapter 2. In this regard, indirect methods have led to Newton/GMRES recipes for solving the optimization [130]. On the other hand, we made use of direct collocation to obtain the solution. Similar to Chapter 4, both inputs and states were decision variables.

Applying direct-collocation to the continuous optimization problem led to a nonlinear program (NLP), which is much harder to solve than Quadratic Programs (QP) or Linear Programs (LP) that result from linear MPCs [62]. As we see in the subsequent sections, the complexity of the NLP caused problems in real-time deployments of NMPC. The discretized NLP is shown below:

$$\begin{aligned} \min_{X,U} \sum_{k=1}^{H-1} \ell(x(t_k), u(t_k), t_k)(t_k - t_{k-1}) + \ell_T(x(t_H), H) \\ U = [u(0), u(1), \dots, u(H-1)] \\ X = [x(0), x(1), \dots, x(H)] \end{aligned} \quad (5.21)$$

$$u_{min} \leq u(t_k) \leq u_{max} \quad \text{for } k = 1, 2, \dots, H - 1 \quad (5.22)$$

$$x_{min} \leq x(t_k) \leq x_{max} \quad \text{for } k = 1, 2, \dots, H \quad (5.23)$$

$$x(t_{k+1}) = f(x(t_k), u(t_k), t_k) \quad \text{for } k = 1, 2, \dots, H - 1 \quad (5.24)$$

$$x(t_0) = x_0 \quad (5.25)$$

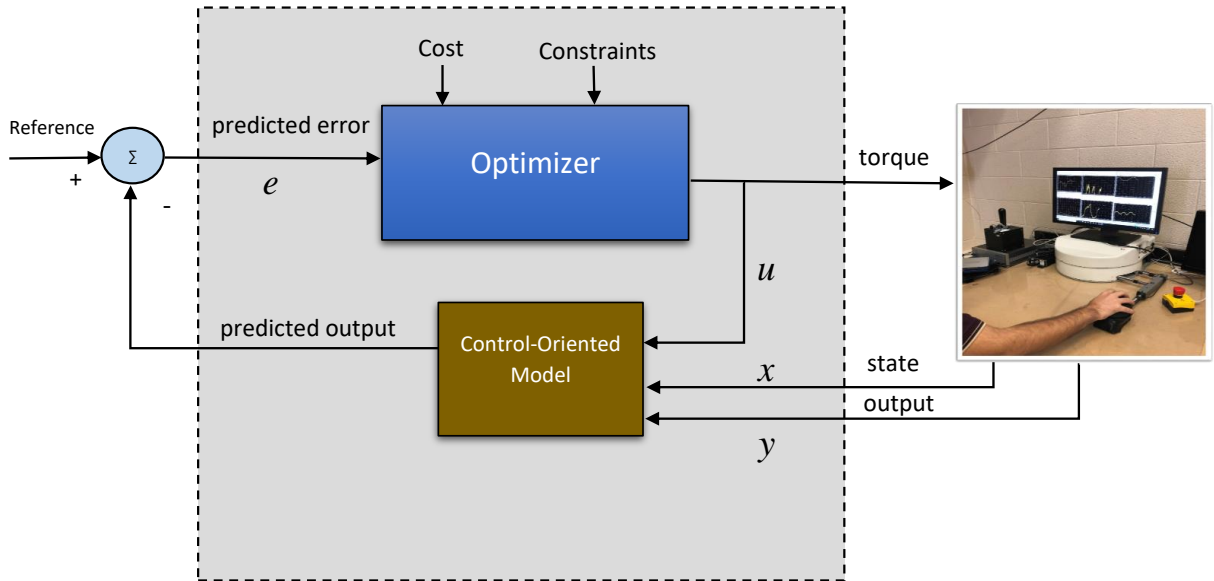


Figure 5.4: MPC block diagram

For optimal control problems with large state-input space, finding the Lagrangian and Hessian of the cost function is computationally demanding and this process hinders the real-time application of NMPC. As a result, solvers like `fmincon` that utilize numerical recipes for finding these values will face computational issues in real-time. To mitigate this problem, research has been conducted on symbolic differentiation for speeding up NMPC [57, 83, 8]. These methods posit that the symbolic gradients and Hessians can be calculated offline; the numerical values then replace the symbols during the online computation. This

idea changes the numerical differentiation problem to a numerical substitution one, and hence expedites the online optimization. In this thesis, we applied two of these approaches on the robot dynamics: CasADI-based and Maple-based methods. In what follows, a comparison of these methods is presented.

CasADI-based NMPC

CasADI is an open-source software framework for nonlinear optimization algorithmic differentiation [9]. This algorithm takes advantage of forward and reverse modes of algorithmic differentiation of expression graphs to produce gradients and Hessians. CasADI can be used in Octave/MATLAB, Python, and C++. Also, stand-alone C code can be generated from all these platforms. CasADI offers multiple NLP solvers; based on our problem, we used the sequential quadratic programming (SQP). The differentiation was done offline.

Maple-based Symbolic NMPC

Developed by Maitland et. al [84], this idea suggests using Maple[®] for producing symbolic gradients and Hessians. Maple takes advantage of extensive code optimization routines and consequently, is a great candidate for symbolic differentiation. Using Maple's code generation, the gradients and Hessians were exported to MATLAB, where they were filled with values at each time-step. Similar to CasADI, the process of symbolic differentiation in Maple was offline and its computation did not affect the online turnaround time.

The stage and terminal costs are defined below. The square of the tracking error, input, and input rate were chosen as the cost, to penalize the state error and high robot motor torques and input jerks.

$$\begin{aligned}
 \ell(x(t_k), u(t_k), t_k) &= (x(t_k) - x_{ref}(t_k))^T Q (x(t_k) - x_{ref}(t_k)) + u(t_k)^T R_1 u(t_k) \\
 &\quad + \dot{u}(t_k)^T R_2 \dot{u}(t_k) \\
 \ell_T(x(t_H), H) &= (x(t_H) - x_{ref}(t_H))^T Q (x(t_H) - x_{ref}(t_H))
 \end{aligned}
 \tag{5.26}$$

where Q , R_1 , and R_2 are the weights associated with state errors, inputs, and input rates, respectively. Equality and inequality constraints were added to the stage cost using Lagrange multipliers and slack variables, respectively. Moreover, log barrier functions were

utilized to penalize solutions with zero slack variables [146]. The resulting unconstrained optimization problem, displayed in Eq. 5.27, ensured that the stage cost was minimized while meeting both the equality and inequality constraints. States, inputs, Lagrange multipliers, and slack variables were the final decision variables $z = [x(t_k), u(t_k), lm(t_k), s(t_k)]$. The nonlinear equations were obtained by applying Karush-Kuhn-Tucker (KKT) optimality conditions to Eq. 5.27; these equations were then solved to acquire z^* .

$$\min_{X,U} \sum_{k=1}^{H-1} (\ell + \ell_T + \ell_{eq} + \ell_{ineq} - \mu \ell_{barrier}) \quad (5.27)$$

where:

$$\ell_{eq} = lm(t_k)[\dot{x}(t_k) - f(x(t_{k-1}), u(t_{k-1}, t_k))] \quad (5.28)$$

$$\begin{aligned} \ell_{ineq} = & [u_{min}(t_k) - u(t_k) + s_{umin}(t_k)] + [u(t_k) - u_{max}(t_k) + s_{umax}(t_k)] \\ & + [x_{min}(t_k) - x(t_k) + s_{xmin}(t_k)] + [x(t_k) - x_{max}(t_k) + s_{xmax}(t_k)] \end{aligned} \quad (5.29)$$

$$\ell_{barrier} = \log(-s_{umin}(t_k)) + \log(-s_{umax}(t_k)) + \log(-s_{xmin}(t_k)) + \log(-s_{xmax}(t_k)) \quad (5.30)$$

Newton's root-finding method was utilized to solve the final nonlinear equation. This method is an iterative approach that starts from an initial guess z^k and applies updates until a convergence criteria is met. Loosely speaking, this method is the second-order version of gradient-descent. The update rule uses the Lagrangian and Hessian of the cost function (Eq. 5.31). Scalar α_k is the update step and was found using a line-search method. Since Newton's root-finding method is sensitive to the initial guess, MATLAB's `fmincon` was used only for the first iteration to give an accurate solution, which was then used as the initial guess for the Newton's solver. This process was implemented offline to avoid the computational costs of `fmincon` in real-time. The scheme of the Maple-based NMPC is shown in Fig. 5.5.

$$\begin{aligned} H_L(z^k)\Delta z^k &= -\Delta L(z^k) \\ z^{k+1} &= z^k + \alpha^k \Delta z^k \end{aligned} \quad (5.31)$$

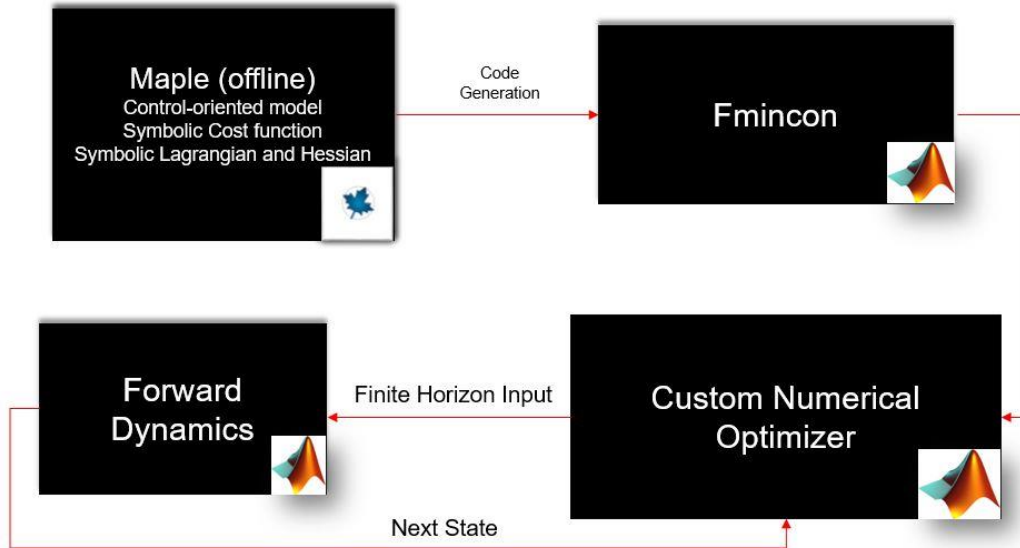


Figure 5.5: Maple-based NMPC scheme

Comparison Between Maple-based and CasADI-based NMPC

After implementing both controllers in MATLAB, they were transferred to Simulink for performance comparison. To this end, we used the MATLAB Systems block for implementation. Using this option, the loading of symbolic values ran only once, while the main loop ran at each iteration; this option was useful for efficient implementation of symbolic NMPCs. In both controllers, some of the functions were not compatible with QUARC’s code-generation. As a result, QUARC’s communication API was utilized. The controllers were placed in the first Simulink file which ran in normal simulation mode; the second Simulink file, which included the robot dynamics, ran in the external mode. The states and inputs were transferred between these two files via a TCP/IP communication protocol. Before we implemented the NMPC controllers on the experimental setup, we evaluated them in simulation. The model sampling time was set to 2 ms (500 Hz frequency), similar to the hardware sampling rate. This way, we could test the real-time practicality of the controllers in simulation. It is worthwhile to mention that due to the rapid update frequency of the robot, online computational time was our primary challenge in NMPC implementation. Timer functions were used to check the turnaround time for each control iteration. The results are shown in Fig. 5.6. The CasADI turnaround time was higher than Maple-based NMPC. In fact, CasADI’s computation time was even higher than the

model sampling time and hence, it could not be applied in real-time. Overall, the Maple-based NMPC provided a lower turnaround time, which made it more suitable for physical implementation. Having said that, we can see from Fig. 5.6 that its computation time was just slightly lower than model sampling-time. The rule of thumb in control is that the frequency of the controller should be 5-10 times higher than the frequency of the model. As a result, future research should focus on speeding up the optimization or the solver. For instance, [85] has examined model reduction methods within MPC. Notably, the iteration time only included the solver, not the other sections of the code. Specifically, the QUARC communication API added a considerable delay to each iteration.

To sum up, despite the improvements in the turnaround time by taking advantage of algorithmic and symbolic differentiation, due to implementation limitations and the delay in communication API, NMPCs were not able to provide fast enough solutions. As seen in subsequent sections, this problem hurt their performance compared to other controllers.

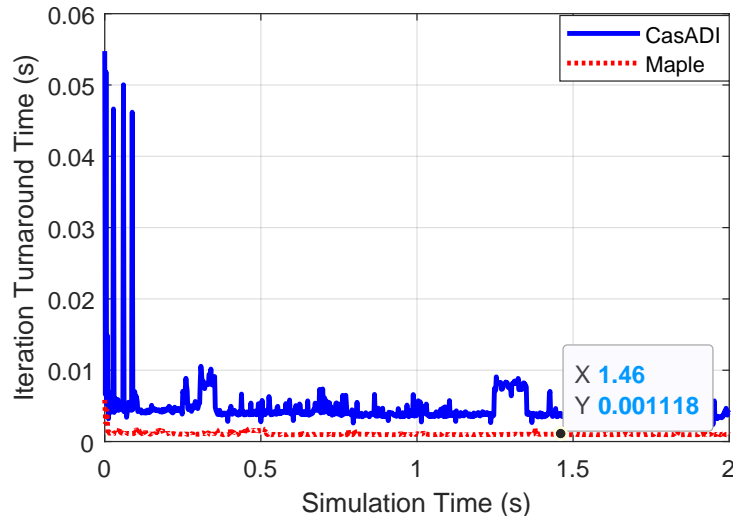


Figure 5.6: Comparison between the iteration turnaround time of CASADI and Maple NMPC

5.3 Simulation Results

All the aforementioned controllers were applied to the robot in simulation (MATLAB 2019b). A circular desired trajectory was set for assessment. The 2 ms sampling time, same as the hardware sampling time, was set. The control-oriented robot model was

selected for all the model-based controllers and the robot motor torque was applied to the high-fidelity model to evaluate the performance. A prediction horizon of $H = 10$ was set for both NMPCs. The root mean square error (RMSE) was calculated to evaluate the position/velocity performance of the controllers (Eq. 5.32). \hat{z}_i and z_i are the desired and observed variables (position/velocity), respectively; n is the number of data points. The results are shown in Table 5.1. The tracking errors were close in simulation. Velocity performance degradation was observed in NMPCs due to the high sampling time.

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (\hat{z}_i - z_i)^2}{n}} \quad (5.32)$$

Table 5.1: RMSE Simulation

Controller	$e_x(m)$	$e_y(m)$	$e_{\dot{x}}(m/s)$	$e_{\dot{y}}(m/s)$
PID	0.027	0.054	0.085	0.102
Comp PID	0.030	0.061	0.074	0.092
LQR	0.030	0.060	0.096	0.106
SMC	0.042	0.081	0.053	0.081
CasADI-based NMPC	0.049	0.086	0.049	0.065
Maple-based NMPC	0.026	0.046	0.102	0.136

The end-effector position and velocities are shown in Fig. 5.7. Also, the inputs are depicted in Fig. 5.8. LQR displayed the fastest tracking response. In addition, PID had the highest input. In fact, PID exceeded the torque limit in the beginning of simulation. As discussed in Chapter 4, the spike in the inputs is for overcoming the robot inertia and is an important factor for evaluating the controller performance; it shows how much acceleration the controller induces in robot joints. High acceleration results in more jerk on the end-effector and less patient comfort at the beginning of rehabilitation. It may even lead to damage to the upper-arm in extreme cases.

It was again validated that the Maple-based NMPC presents a faster response than the CasADI-based NMPC. Having said that, it was observed that despite abiding by imposed input constraints, Maple-based NMPC required a high torque for maintaining the robot on the trajectory. Upon further investigation, it was found that increasing the sampling time to 10 ms greatly improved the performance of both NMPCs. The results for the new NMPCs are shown in Fig 5.9.

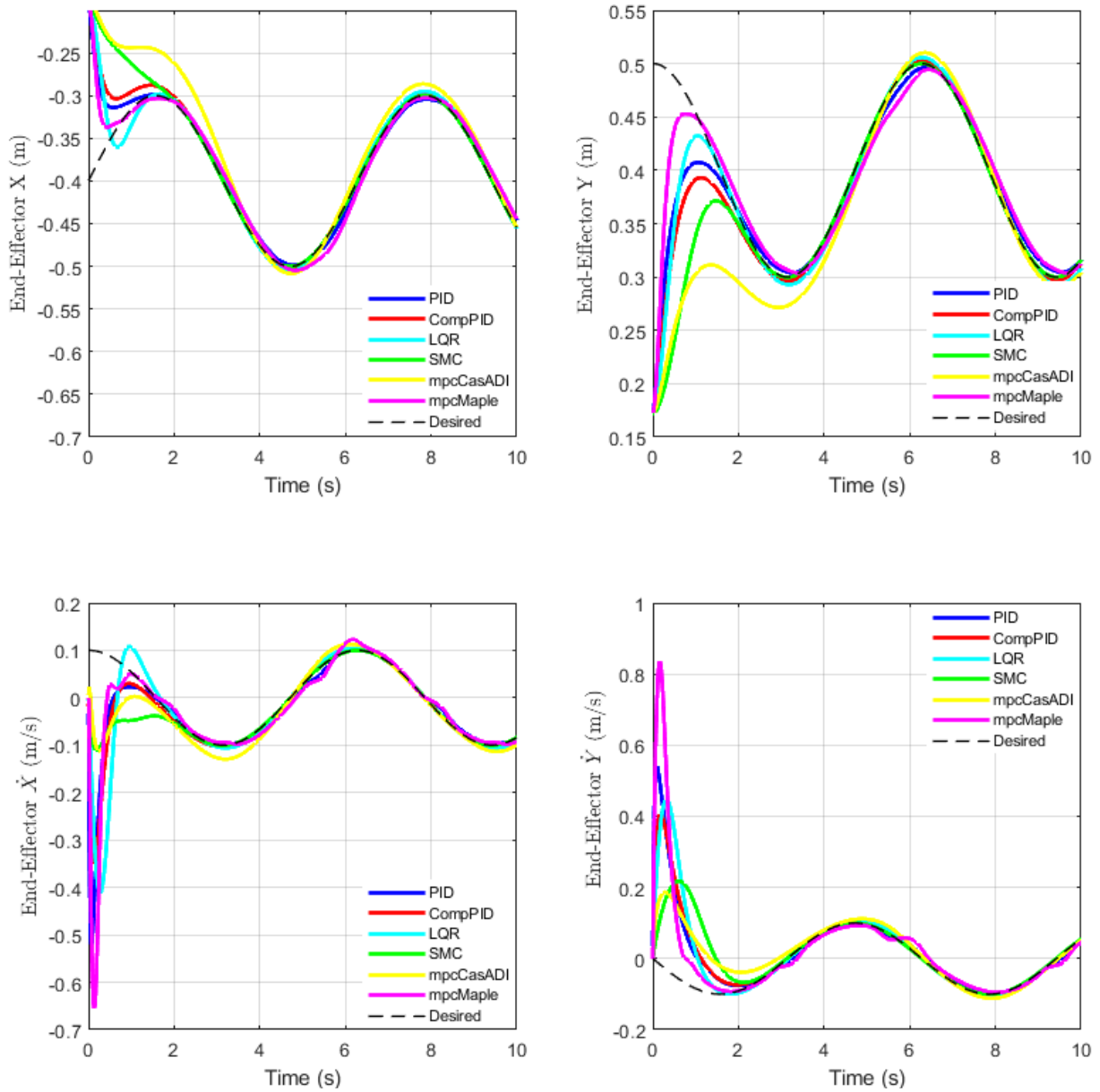


Figure 5.7: End-Effector position/velocity simulation comparison (2 ms sampling time)

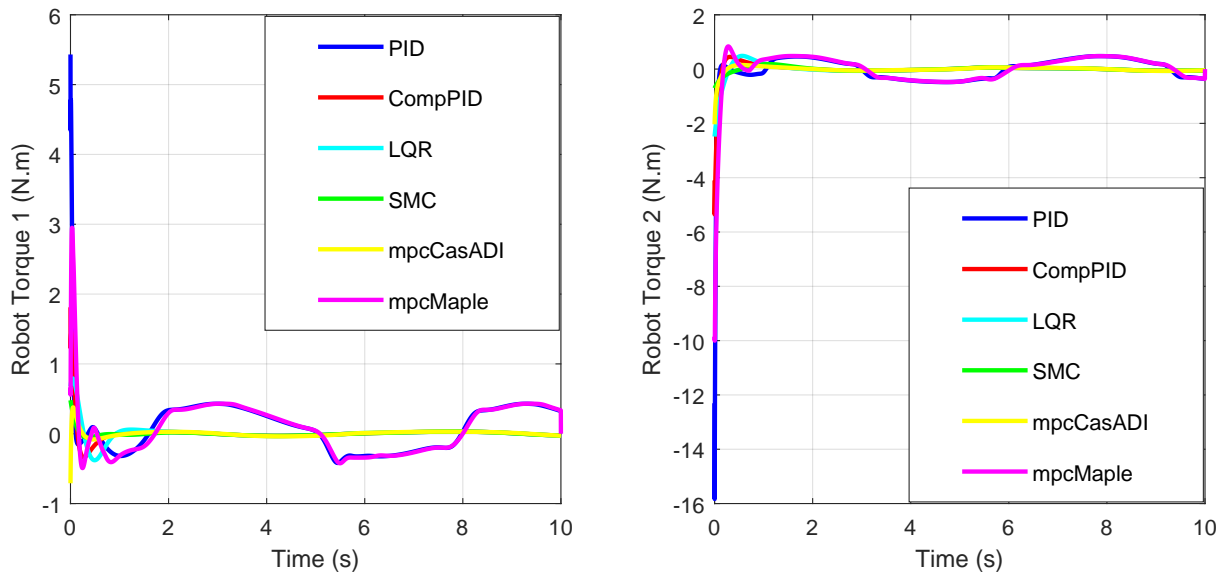


Figure 5.8: Robot torques simulation comparison (2 ms sampling time)

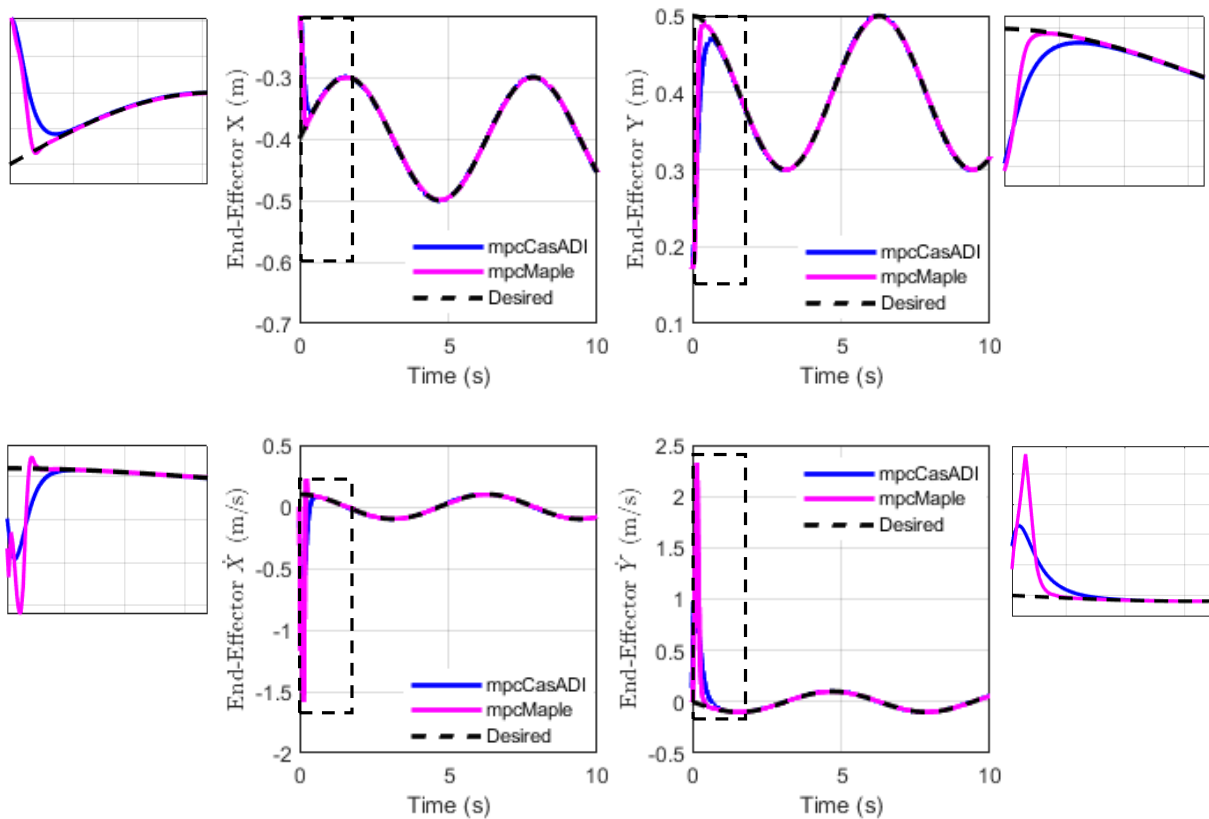


Figure 5.9: NMPC implementations with increased sampling time (10 ms sampling time)

5.4 Experimental Results

To further investigate the performance of the controllers on real hardware, all of them, except for CasADI-based NMPC, were implemented on the experimental setup. The goal of this section was to do a comparative study on the advantages and disadvantages of each controller and to make generalizations on the best controllers to use on rehabilitation robots. This study can potentially help the rehabilitation robotics community to choose the suitable controller for their application. To review, we were limited to use Simulink to connect to QUARC interface. In contrast to utilizing ROS or stand-alone C code, this approach is not common in real-time control, and is slower than other methods. Also, note that the subject was not considered in the comparative study and the pure robot control was examined. Having said that, the comparison criteria included both the control and

rehabilitation aspects.

5.4.1 Comparison Criteria and Implementation

Two control scenarios were considered for comparison, namely **Point Stabilization** and **Tracking**. In the former, the robot should reach a certain point in the workspace and the controller stabilized the end-effector about the point. This scenario is very common in point-to-point reaching movements in rehabilitation, when the focus is only on the final point. For point stabilization, we considered the end-effector velocity, the norm of robot input vector, the norm of robot input rate vector, and the overall controller speed as the comparison criteria. The end-effector velocity and input rate norm are correlated to the end-effector jerk and patient comfort. Due to the high use frequency of rehabilitation robots, the norm of the input vector was chosen to improve DC motor durability. In other words, the same performance with lower input was preferred.

5.4.2 Tuning Process

In order to have a fair comparison, it is imperative to apply a generalized tuning process for all methods. However, this process is highly time-consuming and requires extensive research. In fact, to the best of the author's knowledge, none of the comparative control studies in the literature have used this strategy [69, 79, 120, 61, 32, 33]. Therefore, we adopted another strategy to make the comparison valid. Knowing that all of the controllers were capable of producing near-perfect tracking, the position weights were adjusted to reach just below 1% position error. All the other weights were tuned manually as long as the former error bound was valid. By this approach, position error was set highly similar for all the controllers and the primary focus was set on other criteria for comparison. Notably, due to this tuning strategy, the resulting controllers were highly aggressive and were not suitable for rehabilitation purposes. These set of weights were only useful for the comparison study and another set should be selected when working with patients. The tuning was conducted online on the point stabilization; the same weights were then utilized for tracking. The weights for all the controllers are presented in Appendix B.

5.4.3 Point Stabilization Results

The final point of the robot is displayed in Fig. 5.10. A 3 Ampere saturation block was put on the final current to circumvent the motors overload. Distance from the final point

$|d|$, input norms $|u|_2$ (Eq. 5.33), and input rate norms $|\dot{u}|_2$ (Eq. 5.34) were selected as the quantitative metrics for comparison.

$$|u|_2 = \sqrt{\sum_{j=1}^2 \sum_{i=1}^n u_{i,j}^2} \quad (5.33)$$

$$|\dot{u}|_2 = \sqrt{\sum_{j=1}^2 \sum_{i=1}^{n-1} \left(\frac{u_{i+1,j} - u_{i,j}}{\Delta t} \right)^2} \quad (5.34)$$

The results are shown in Table 5.2 and Figs. 5.11, 5.12. The tuning process made the final distance from the target relatively low for all the controllers. Maple-based NMPC had the least input norm and LQR had the least input rate norm. NMPC and SMC were associated with the least end-effector velocities in X and Y directions, respectively. The inputs for PID were much higher than the input limit and are hence shown in a separate figure (Fig. 5.13); the damage to the DC motors was avoided by using the current saturation block. We tried to constrain the PID torques to the input limits $[-10, 10]$ N.m by dividing all the weights by a constant value (Fig. 5.14). The results were not satisfactory and the controller could not reach the desired point in the assigned 5 second time limit (Fig. 5.15).

PID and computed-torque PID provided the fastest responses. Upon further investigation, it was found that these controllers reached the maximum current; they presented a pseudo bang-bang control behavior which is associated with minimum time solutions. Their fast response is explained with this effect. Considering the fact that response speed was our last metric and considering the high input requirements of these controllers, they were not considered as the best candidates.

Maple-based NMPC presented the same computational deficiency in real-time implementation. The NMPC delay in reaching the desired point is evident in Fig. 5.11.

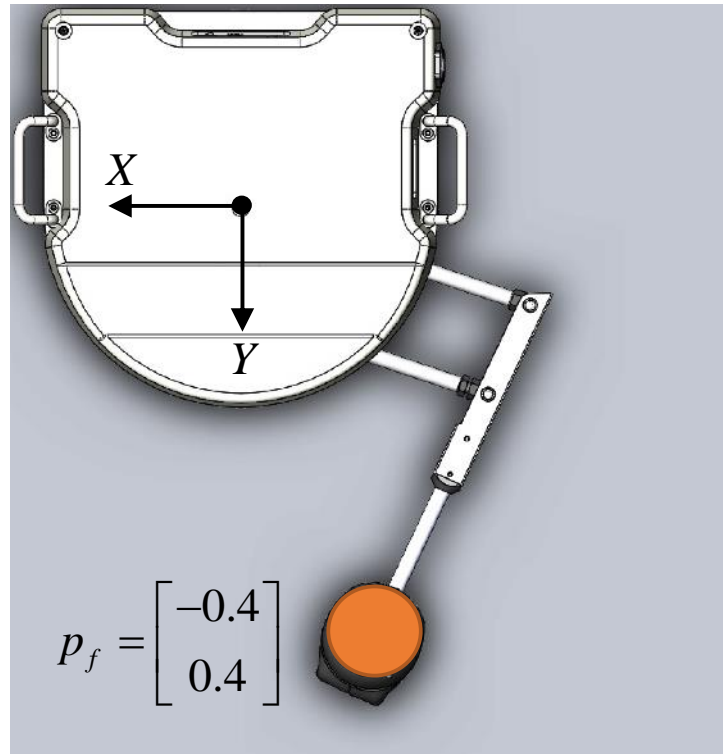


Figure 5.10: Robot schematics for point stabilization

Table 5.2: Control Point Results

Controller	$ d (cm)$	$ u _2 (N.m)$	$ \dot{u} _2 (\frac{N.m}{s})$
PID	0.650	360.873	0.120
Comp PID	0.590	76.834	0.007
LQR	0.640	50.756	0.004
SMC	0.370	42.494	0.012
NMPC	1.430	35.841	0.015

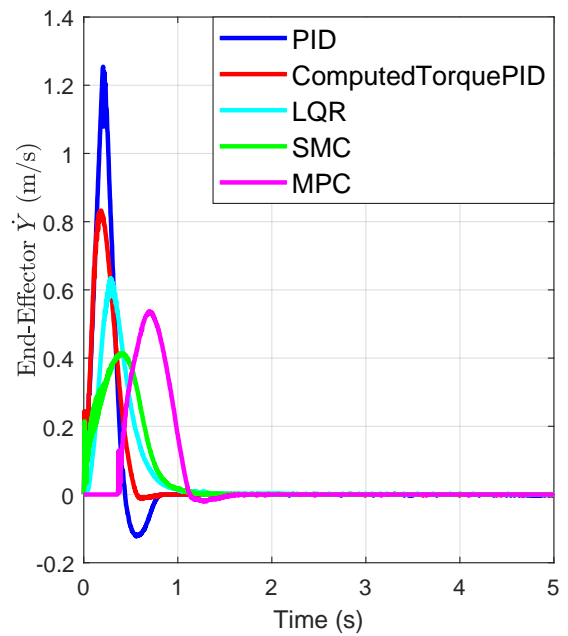
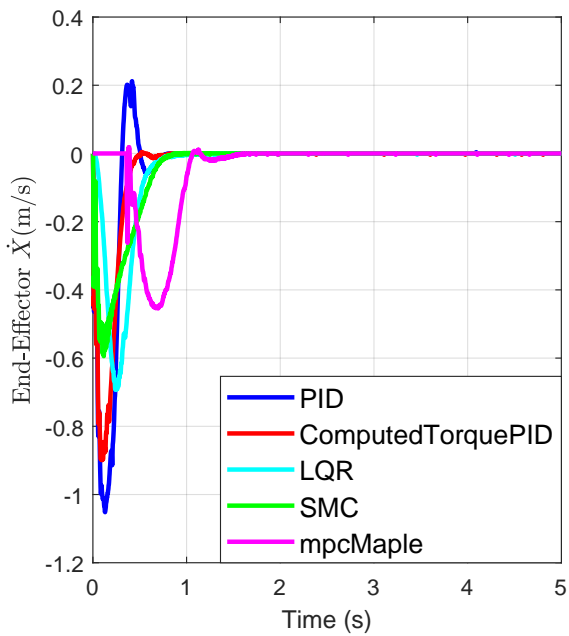
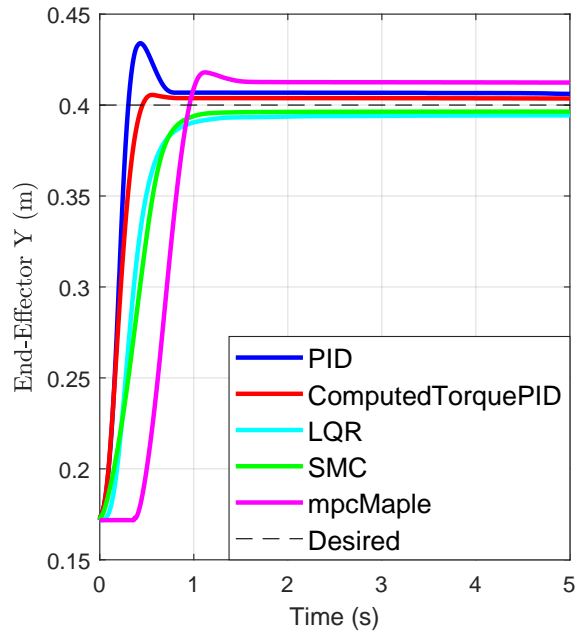
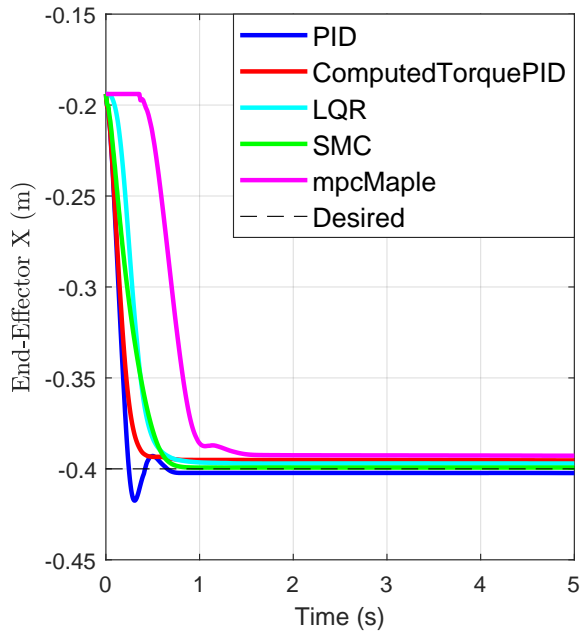


Figure 5.11: Point stabilization results

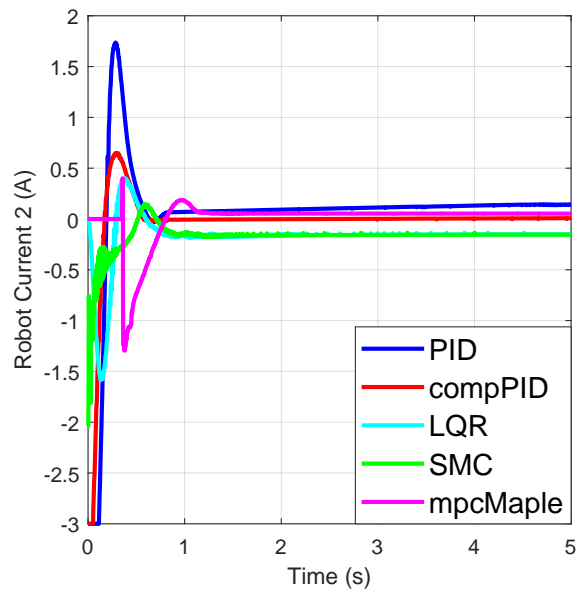
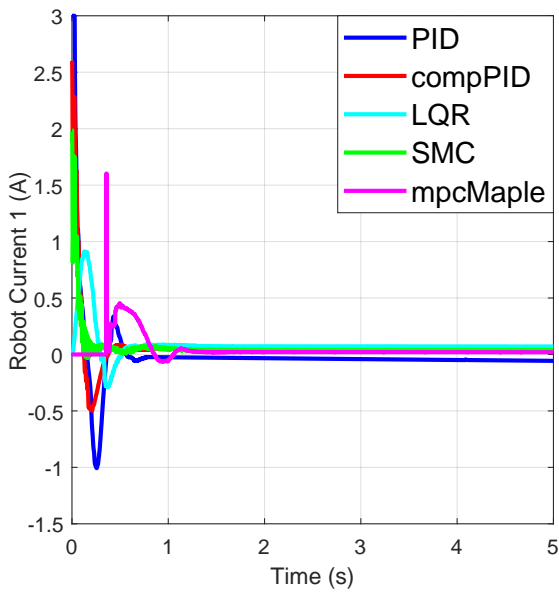
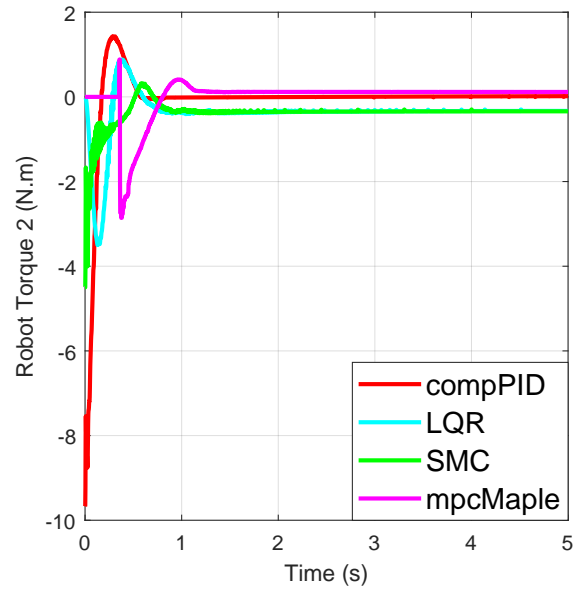
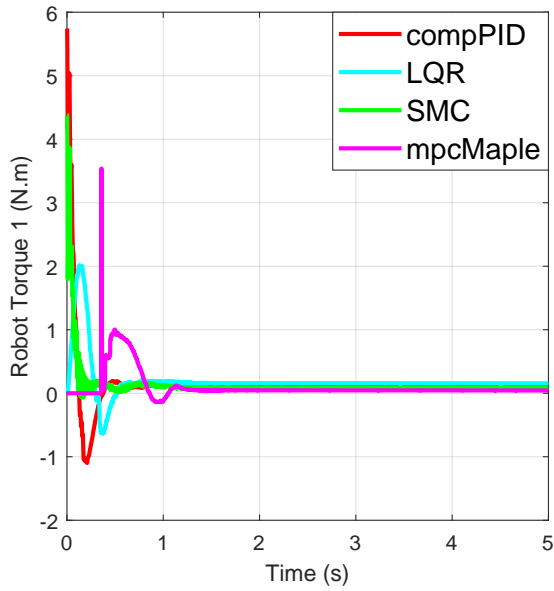


Figure 5.12: Point stabilization torque/current results

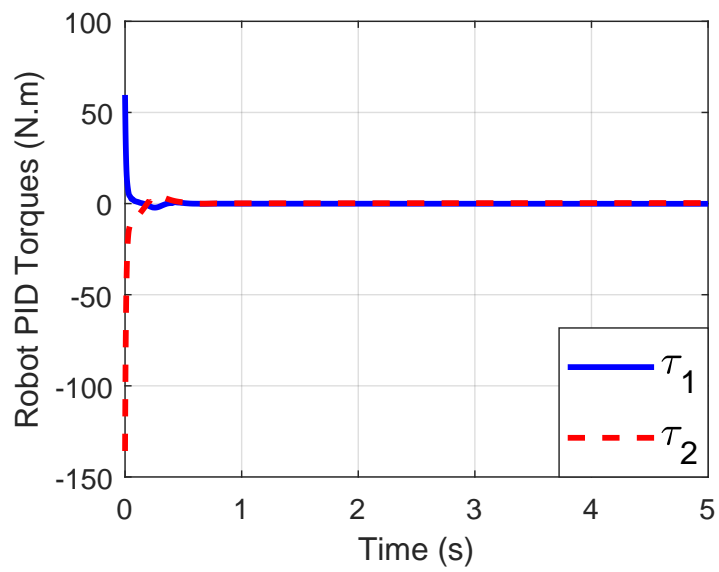


Figure 5.13: PID point inputs

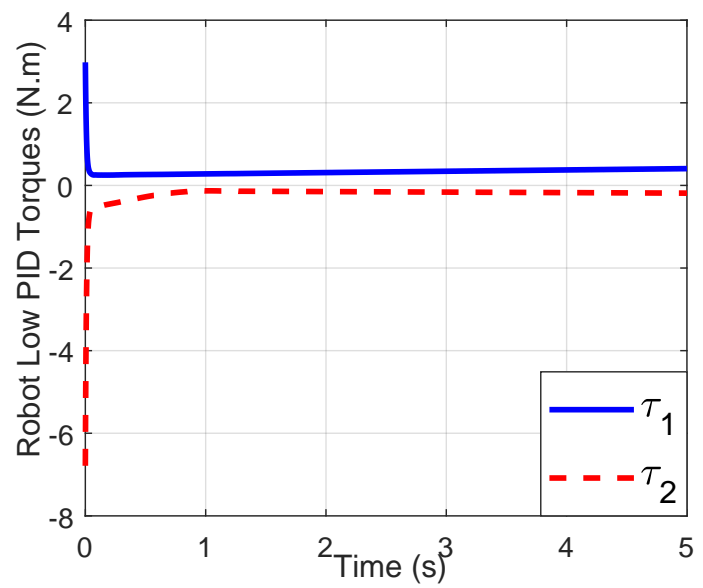


Figure 5.14: PID low inputs

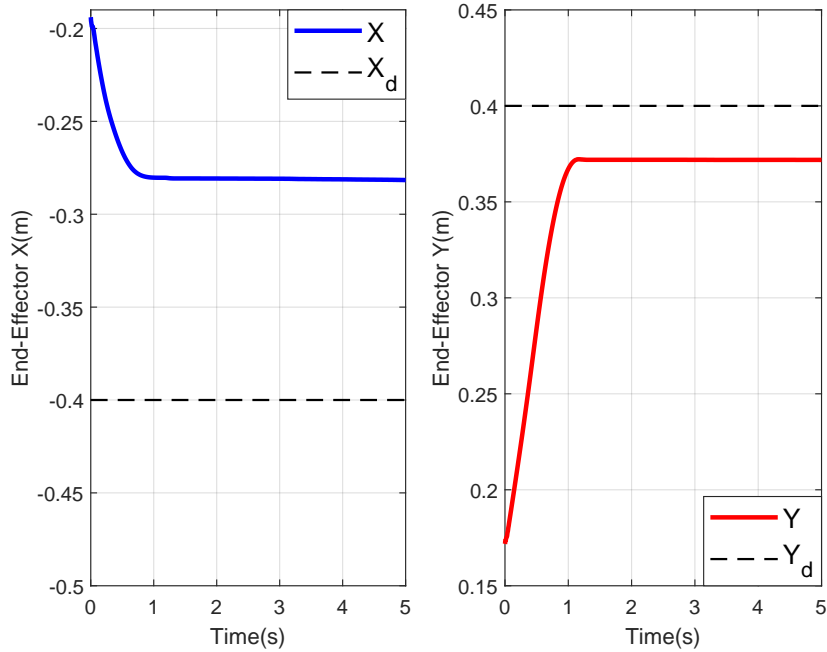


Figure 5.15: PID end-effector positions with lower inputs

5.4.4 Tracking Results

The same circular trajectory as the simulation was defined for physical testing (Fig. 5.16). The norm of position ($|e_d|_2$) and velocity ($|e_v|_2$) errors along with input and input rate norms are presented in Table 5.3. SMC and LQR had the least input and input rate norms, respectively. The tracking results are shown in Figs. 5.17-5.20. PID, LQR, and SMC had the best position tracking performance; Note that the fact that computed-torque PID had a lower position error than the advanced controllers is because of its faster response at the beginning, when the error values are very high. It is evident from Figs. 5.17, 5.18 that this controller does not outperform other algorithms in tracking. Overall, SMC and LQR represented the best position/velocity tracking performance.

The high turnaround time of NMPC stymied the successful tracking in position and velocity levels, hence stressing the need for expediting the optimizer to be used in rapidly-updated systems. The torque/current results are depicted in Fig. 5.21. Again, the PID torques were much higher than the limit and therefore, was not shown. In addition, it was observed that computed-torque PID exceeded the torque limits, as well.

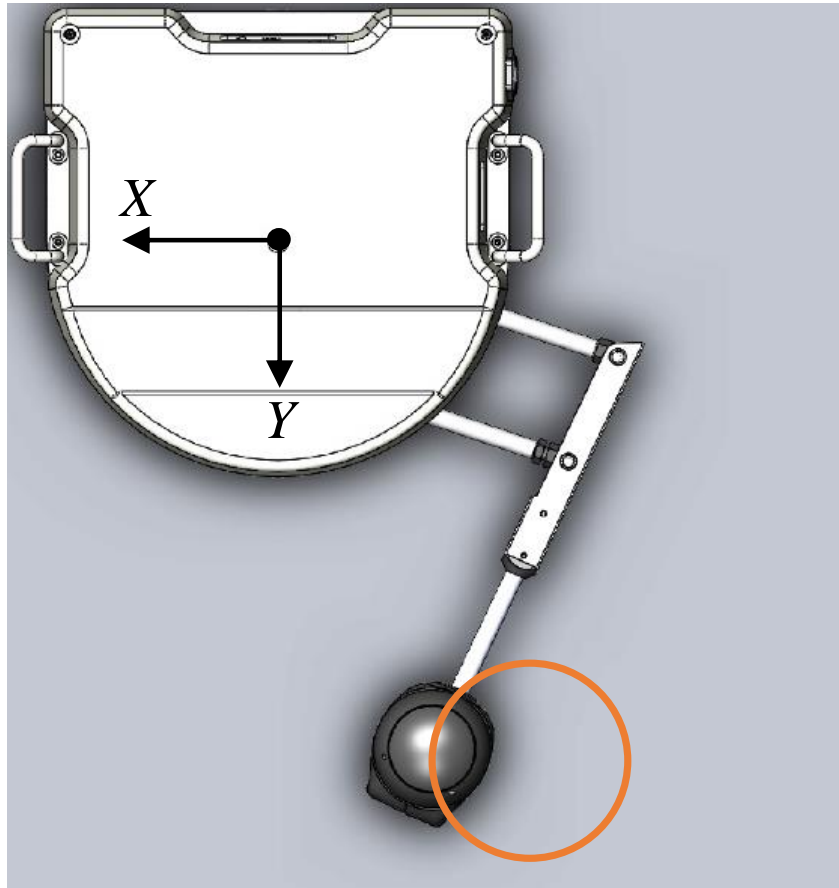


Figure 5.16: Robot schematics for tracking

Table 5.3: Control Tracking Results

Controller	$e_d(m)$	$e_v(m/s)$	$ u _2 (N.m)$	$ \dot{u} _2 (\frac{N.m}{s})$
PID	3.510	19.104	514.062	0.169
Comp PID	3.688	16.019	124.945	0.008
LQR	4.295	12.855	75.424	0.006
SMC	4.825	10.048	56.484	0.013
NMPC	5.395	13.606	65.269	0.009

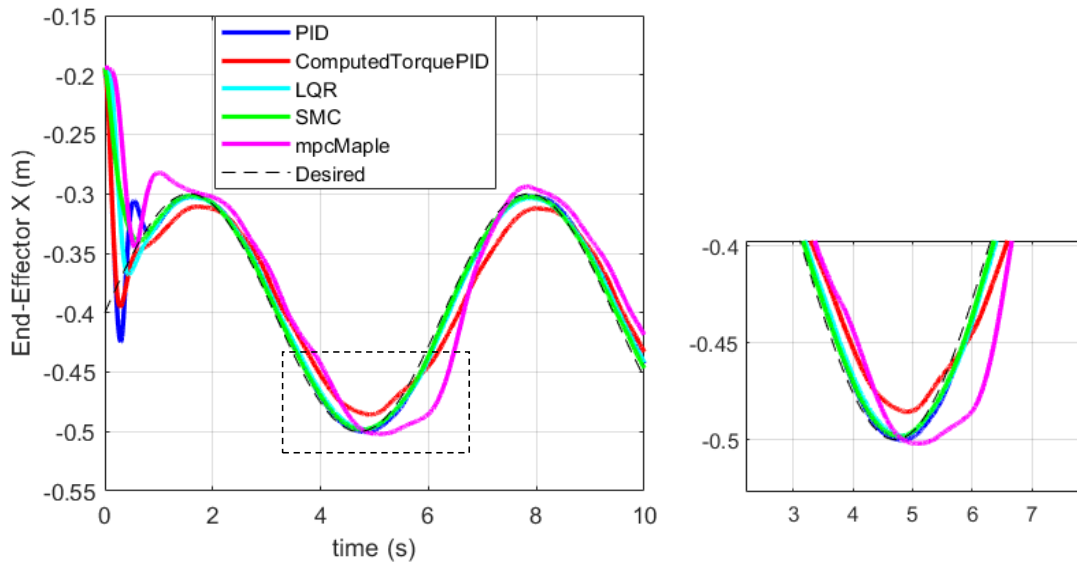


Figure 5.17: End-effector X position result for tracking

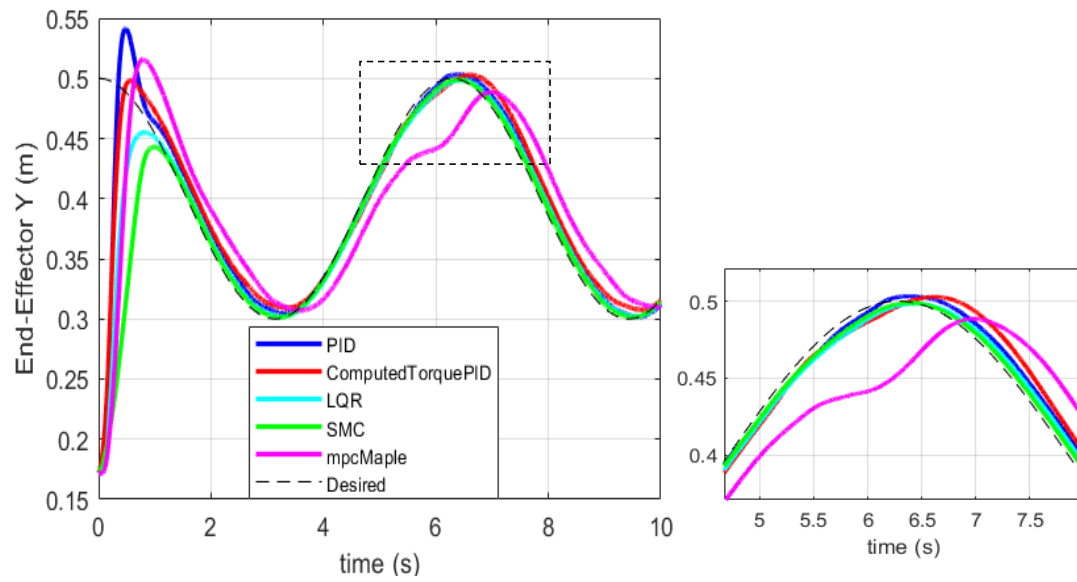


Figure 5.18: End-effector Y position result for tracking

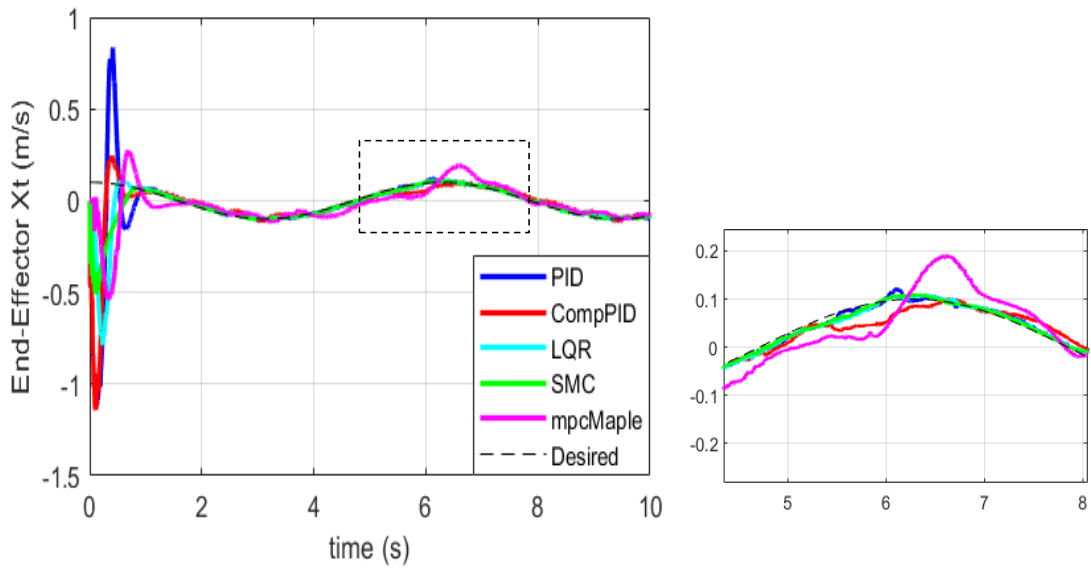


Figure 5.19: End-effector X velocity result for tracking

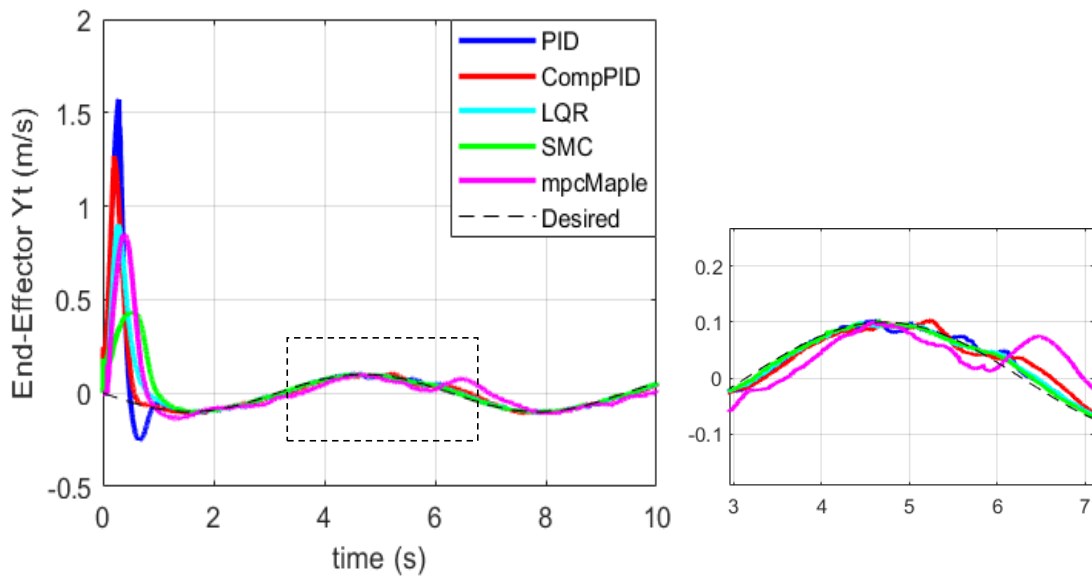


Figure 5.20: End-effector Y velocity result for tracking

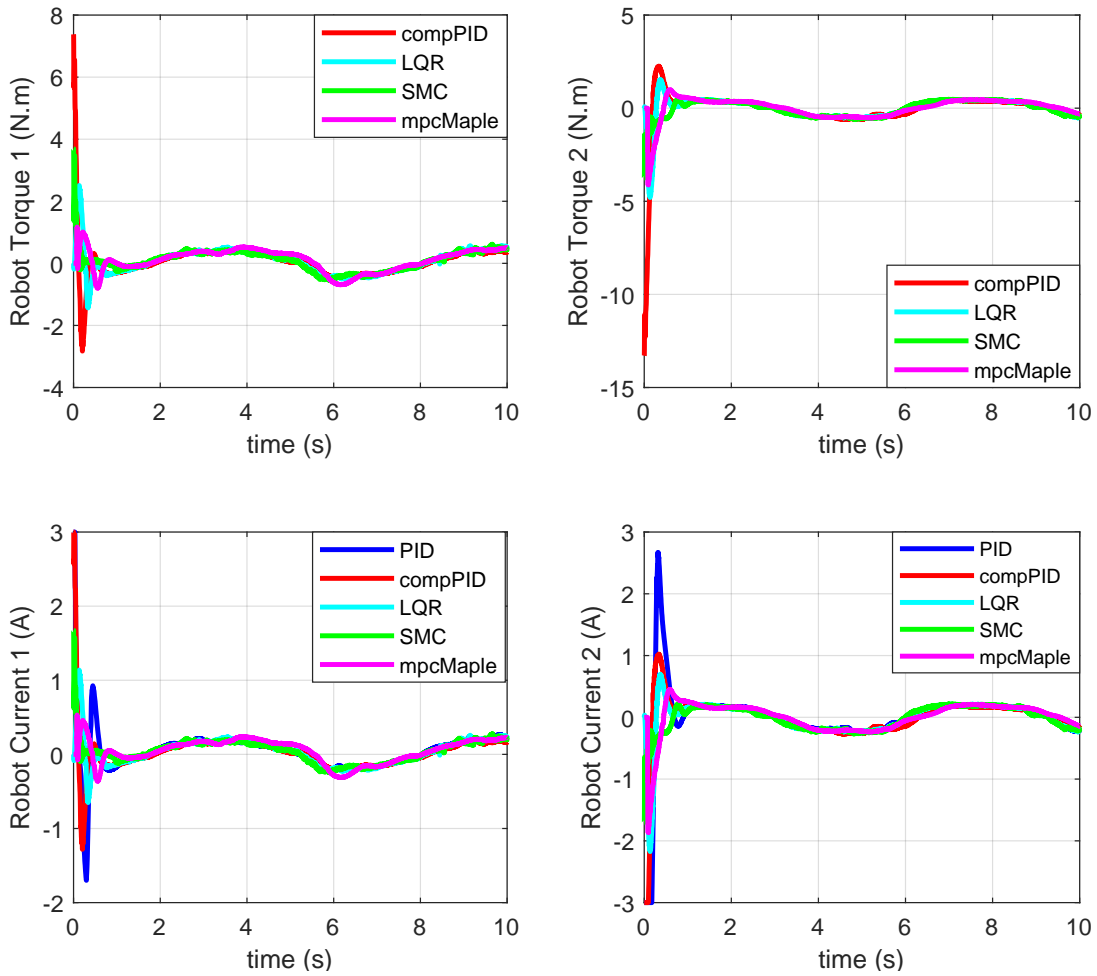


Figure 5.21: Robot torques and current results for tracking

5.5 Conclusion

In this chapter, we tested five controllers on the robot. The simulation results showed that the CasADI-based NMPC solver was not fast enough for real-time application. Hence, we focused on comparing the other controllers in real-time. The resulting high inputs and high input rates of PID, and computed-torque PID justified the use of more advanced controllers. Despite the improvement in the turnaround of Maple-based NMPC by utilizing symbolic gradients, the need for further expediting the optimization was observed. All in

all, it was concluded that SMC and LQR were the best candidates for our rehabilitation robot. It is hypothesised that the success of the SMC is due to its capability to handle nonlinear systems and the success of LQR is because of its good performance on planar mechanisms.

Our comparative study showed that for desirable rehabilitation performance, advanced controllers are necessary. SMC, as a nonlinear controller, provided promising results. We suggest this controller for highly nonlinear assistive robots but with further study on the chattering when experiencing high disturbances, as in patient forces. The successful performance of LQR made it a potential candidate for rehabilitation robots as well; having said that, it should be noted that the planar mechanism of our robot contributed to the performance of LQR. It is hypothesized that performance degradation is observed if LQR is applied on non-planar robots like exoskeletons. For these systems, we suggest either SMC or NMPC but with extra care on the computational time. It is worthwhile to review that part of the problem with the high turnaround time of NMPC was the limitation with implementation. The QUARC communication API introduced an extra delay to each iteration which further made the controller slower. This problem is not observed on other robots which enable ROS implementation or stand-alone C code generation. It should be noted that findings in this chapter cannot be generalized to all rehabilitation robots and assistive devices. For instance, robots with pneumatic and hydraulic actuators present very different characteristics compared to robots with electrical actuators. Having said that, this comparative study can be useful for DC-actuated robots with similar complexity of dynamics as our setup.

Chapter 6

Deep Reinforcement Learning Tuning of the Model-based Controllers

6.1 Overview

In the previous chapter, we tested multiple controllers on the robot; all of these controllers have a firm theoretical foundation. For instance, much progress has been made on their convergence [19, 45]. On the other hand, not much theoretical insight have been gathered about deep learning or DRL controllers. Hence, except for special applications like end-to-end control from pixels to torques [98], the well-known controllers are probably the first choice. Nevertheless, extensive manual tuning is required for each controller. DRL help promote a generalized strategy to tune the weights of a controller; this method is not limited to controllers. Any problem that can be formulated as a sequential decision making is a valid application for this framework. For instance, DRL can be used to tune the weights of a parameter identification optimization. RL and DRL have been utilized before for tuning the weights of a controller. However, these methods have either used a discrete set of weights [60, 137], or have focused on a specific high-level controller [144]. In contrast, the tuner presented in this chapter is able to choose from the infinite continuous space. Moreover, this method is applicable to all low-level and high-level controllers. In what follows, the DRL algorithms that have been utilized are explained. A review on reinforcement learning and actor-critic method was presented in Section 2.5.

6.2 Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor-critic DRL method [75]. The actor uses its policy to map the states to the actions, and the critic updates the actor's policies to produce actions that maximize the cumulative reward. The schematics of actor-critic are shown in Fig. 6.1, where S is the state, A is the action, and $Q(S, A)$ is the state-action value function. Unlike Asynchronous Advantage Actor Critic (A3C) methods [97], DDPG produces the action instead of its probability. The algorithm makes use of the deterministic policy gradient (DPG) algorithm [122] to update the actor.

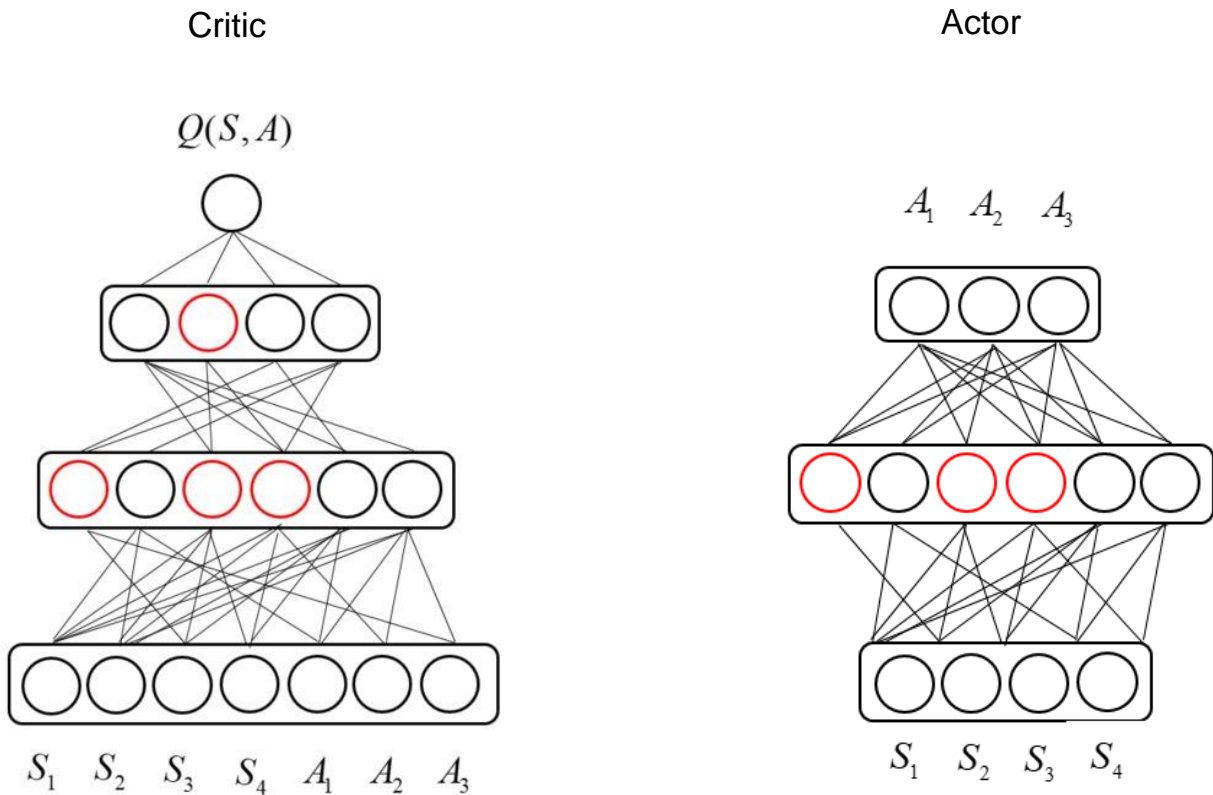


Figure 6.1: Actor-Critic schematics

The following explains the characteristics of DDPG algorithm:

1. **Off-policy:** instead of using recent data to update the policy, DDPG records the

data in a buffer, to be used later for the policy update. The latest update to the policy always comes from the previously recorded data.

2. **Model-free:** the algorithm does not require any transition model of the environment to find the optimal policy. It only needs the (state, action, next-state, reward) tuples. A model can be utilized if it is available to produce the tuples. For instance, the openAI API has coded the dynamics of most of its environments [23]. However, it is not necessary to have the model. Often times, when the model is not available or it is rather difficult and time-consuming to derive it, an approximate representation of the model is used to generate the tuples. This includes, but is not limited to, Fourier-Wavelet Bases, Decision Trees, Least Square Approximations, and Neural Networks. Unlike model-based controllers like LQR or MPC, we are not bound to represent the physical transition model of the system. Any approximation of it, as long as it produces accurate enough tuples, can be utilized.
3. **Continuous-Space:** one of the main advantages of DDPG is that it can handle continuous state-action space problems. As a result, it is a valid candidate for dealing with physical system problems, namely robotic applications. In our application, there is no need to discretize the weight parameter space. Having access to infinite-dimensional parameter spaces enables better tuning strategies.

Contrary to Deep Q Networks (DQN), DDPG can handle continuous-state spaces. Nevertheless, multiple ideas were inspired from DQN in the derivation of DDPG:

- **Replay Buffer:** the acquired samples from the environment are highly correlated physical states and are not independent and identically distributed (non iid). Training the agent with these samples leads to divergence [98]. To alleviate this issue, a replay buffer is defined, in which the samples are recorded. When the buffer reaches a certain size, a random batch of samples are collected to train the agent. This process continues until the end of the training epochs. The random sampling from the buffer removes the non iid sample problem.
- **Target Network:** as mentioned in Chapter 2, RL is a semi-supervised algorithm; i.e. the targets are generated during training. In DDPG, a one-step Temporal Difference (TD) approach is used to calculate the target value for the critic, which requires next state-action value function $Q(s', a')$. However, using the critic to produce $Q(s', a')$ results in a self-updating critic. Needless to say, this also leads to divergent behaviors [98]. The solution is to use another network, called **Target Critic**, for producing

the $Q(s', a')$. It is shown in [75] that by using a **Target Action** network to generate the a' in $Q(s', a')$, more accurate results are achieved. Overall, the DDPG algorithm includes four neural networks: Critic, Actor, Target Critic, and Target Actor. A soft policy update is used for changing the target network’s weights. By defining a soft-update parameter $\tau \ll 1$, a linear combination of the original networks and target network weights will substitute the target network weights at each iteration (Eq. 6.1). $\theta^Q, \theta^\mu, \theta^{Q'}, \theta^{\mu'}$ are critic, actor, target-critic, and target-actor network weights, respectively.

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}\end{aligned}\tag{6.1}$$

- Exploration: approximations added to the Bellman equation remove the guarantee for finding the global optimum. In fact, It is highly probable that the optimizer is stuck in a local optimum. When training the agent, DQN adds a Gaussian noise to the output to explore new areas of the search space other than the ones that the optimizer suggests. This is called **Exploration vs Exploitation Dilemma** in RL literature and is a hot topic in its community [113, 43]. DDPG utilizes Ornstein–Uhlenbeck (OU), a correlated noise originally developed for explaining the Brownian motion of particles in Physics [18].

DDPG is summarized in Algorithm 1.

Algorithm 1 DDPG

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode=1, M **do**
 Initialize a random process N for action exploration
 Receive initial state s_i
 while Termination Condition not Met **do**
 Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and OU noise
 Execute action a_t and observe reward r_t , new state s_{t+1} , and done flag *done*
 Store transition $(s_t, a_t, r_t, s_{t+1}, done)$ in replay buffer R
 Sample a random mini-batch of B transitions $(s_t, a_t, r_t, s_{t+1}, done)$ from R
 Set $y_i = r_i + \gamma Q(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s_i, \mu(s_i|\theta^Q)|\theta^Q) \nabla_{\theta^\mu} \mu(s_i|\theta^\mu)$$

 Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

 end while
end for

6.2.1 Problems of DDPG

Function approximations in RL lead to errors and sub-optimal policies. These errors exist in both value-based methods (like Q-learning) and actor-critic methods. In [44], the problems of DDPG are mentioned as follows:

- **Overestimation Bias:** it is shown that the value function estimates Q in DDPG are higher than the actual values. This fact introduces error in the critic's output and thus leads to overall error in the actor's values.
- **Variance:** DDPG is reportedly known to lead to high-variance estimates [44, 29], which is one of the causes of the aforementioned overestimation bias. In addition, high variance produces noisy gradients for the policy update and reduces learning speed [127].

6.2.2 Twin-Delayed Deep Deterministic Policy Gradient (TD3)

A modified version of DDPG called “Twin-Delayed Deep Deterministic Policy Gradients” (TD3) was introduced in [44]. This algorithm alleviates the above issues by applying the following changes:

1. Double Critic: to reduce the overestimation, two critic and target critic networks are used. The target is then calculated by taking the minimum between the Q values: $y = r + \gamma \min_{i=1,2} Q_{\theta'_i}$, where y is the target for the critic network, r is the immediate reward, and $0 < \gamma < 1$ is the discount factor hyperparameter. All the aforementioned networks are updated separately similar to the formulation presented in Algorithm 1.
2. Delayed Actor: it is shown that high-error in the Q estimates from the critic will exacerbate the errors in the actor, which then affect the efficiency of the critic. The accumulation of error from both networks lead to high final errors. By updating the critic with higher frequency than the actor, the latter is updated with more accurate feedback from the former, hence reducing the overall error. To this end, we delayed the actor by updating it every two iterations.
3. Target Policy Smoothing Regularization: target Q values are susceptible to function approximation errors. This problem causes a part of the variance observed in DDPG. The smoothing regularization adds an extra noise to the target action $\pi_{\phi'}(s')$. The target formulation then changes as follows, where ϵ is the newly introduced noise:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi'}(s') + \epsilon) \quad (6.2)$$

TD3 has outperformed many of the state of the art RL algorithms on multiple OpenAI environments; the results in the original paper [44] has shown that this algorithm provides a higher cumulative reward on all of the environments except for InvertedPendulum-v1 and InvertedDoublePendulum-v1. Even in these environments, TD3’s reward is very close to the best candidate.

6.3 Implementation

Generally, DRL algorithms are applied in Python due to its extensive deep learning libraries like Tensorflow or Pytorch. However, since our controllers were written in MATLAB, writing the tuner in Python would be problematic. There are not any reliable

approaches to connect Simulink and Python for online data transfer. On top of that, we predicted that the delay that would result from the Simulink/Python connection would stymie the online implementation of the tuner on the hardware. Consequently, the tuner is coded in MATLAB 2019b. Due to its current limitations to only DDPG, we did not use the MATLAB’s reinforcement learning toolbox. In addition, MATLAB’s deep learning toolboxes could not handle the policy gradient updates. Hence, the DRL program was written from scratch as MATLAB script. Mini-batch learning was used to train the agent; that is, a mini-batch of data was randomly extracted from the replay buffer and used for training at each iteration. Instead of using loops in the mini-batch training implementation, we took advantage of vectorization which greatly expedited the training. The details of networks and hyperparameters are presented in Appendix C.

6.3.1 Algorithm Validation

To validate the TD3 implementation, the algorithm was used as a controller on a pendulum swing-up problem. The same reward function as the OpenAI Pendulum-v0 environment was set. The angle is shown in Fig. 6.2. The TD3 controller was able to swing-up the pendulum in the assigned time. The switching between -180 and 180 degrees is due to the fact that the absolute value of the target is used in the OpenAI reward function.

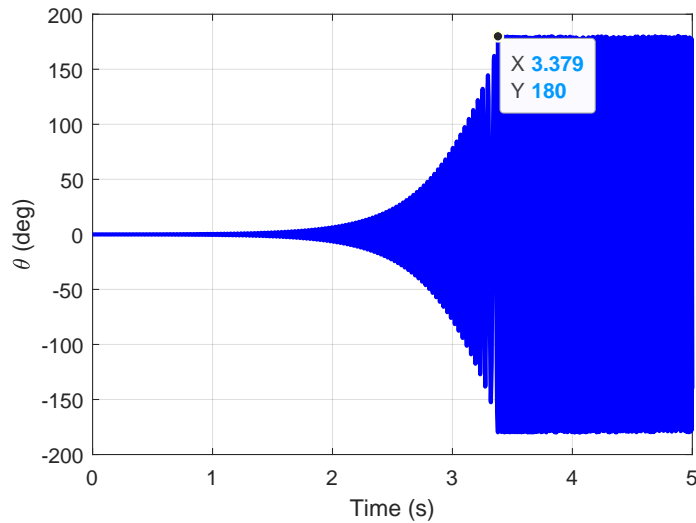


Figure 6.2: The TD3-controlled pendulum

6.3.2 Tuner Structure

The tuner was implemented on the SMC to adjust the K , λ , and ψ weights. For the reward function, the following formulation was considered:

$$\begin{aligned}
 r &= -|d(t)| - 0.1|v(t)| - |\dot{u}(t)| \\
 |d(t)| &= |p(t) - p_d(t)| \\
 |v(t)| &= |\dot{p}(t) - \dot{p}_d(t)| \\
 |\dot{u}(t)| &= \frac{|u(t) - u(t-1)|}{\Delta t}
 \end{aligned} \tag{6.3}$$

where $|d(t)|$ is the iteration position error, $|v(t)|$ is the iteration velocity error, and $\dot{u}(t)$ is the iteration input rate. For the tuning strategy, we considered two general options:

1. Whole-episode strategy: in this method, the tuner sets the weights in the beginning of the iteration (Fig. 6.3). The weights are fixed through the whole episode. The tuner then adjusts them based on the cumulative reward of the whole-episode. This results in an offline tuner. The final weights are set only once, at the beginning. Although this approach is a valid tuning strategy and is used in the literature [60, 55], the resulting controller will not be subject-specific. Hence, we adopted the next strategy.

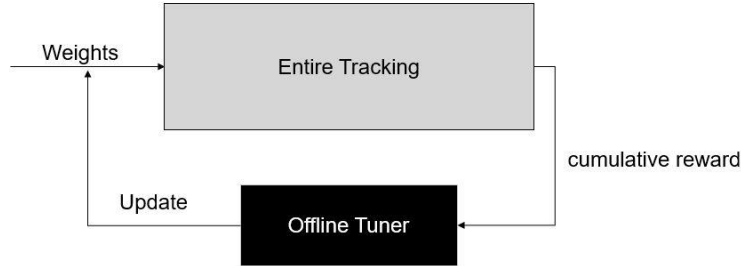


Figure 6.3: The whole-episode tuning strategy

2. Iteration-based strategy: in contrast to the previous approach, this approach adjusts the weights at each iteration (Fig. 6.4). This results in a subject-specific controller that changes the weights based on patient conditions. The tuner reacts to the increase in position/velocity error by increasing the corresponding weight. It is worthwhile to mention that the same strategy can be applied in the force level. Instead of calculating the end-effector position from encoder values, it is possible to record the

patient force and compare it to the healthy subject force. This method requires data collection with a post-stroke and a healthy subject before training. Hence, the former approach was adopted due to its ease of implementation.

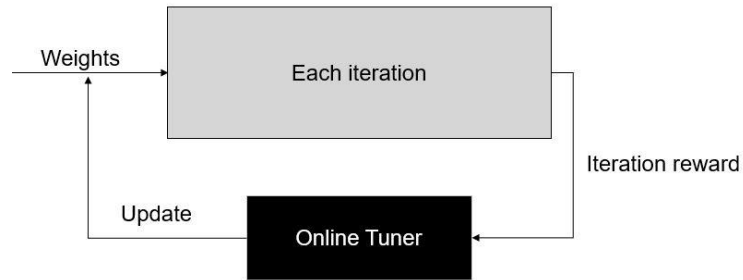


Figure 6.4: The iteration-based tuning strategy

The schematic of the final controller-tuner is shown below:

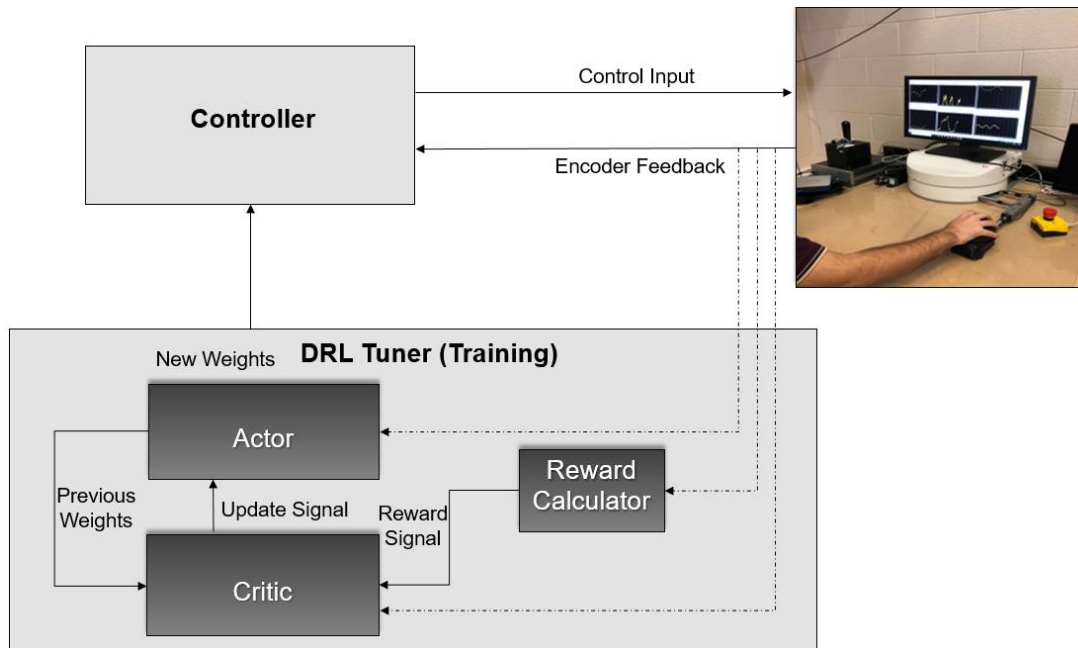


Figure 6.5: The controller-tuner scheme

6.4 Simulation Results

The iteration-based strategy was applied on the SMC method for following a circular trajectory. To simulate the patient in the testing stage, a random disturbance was added to the input at each iteration. The manually-tuned SMC weights in Section 5.4 were set as the initial values in training. The adaptive weights in the testing stage are shown in Fig. 6.6. The tuner produced approximately constant weight values. It is hypothesized that since SMC considers disturbance rejection in its formulation, constant weights were enough for handling the added disturbance at each iteration.

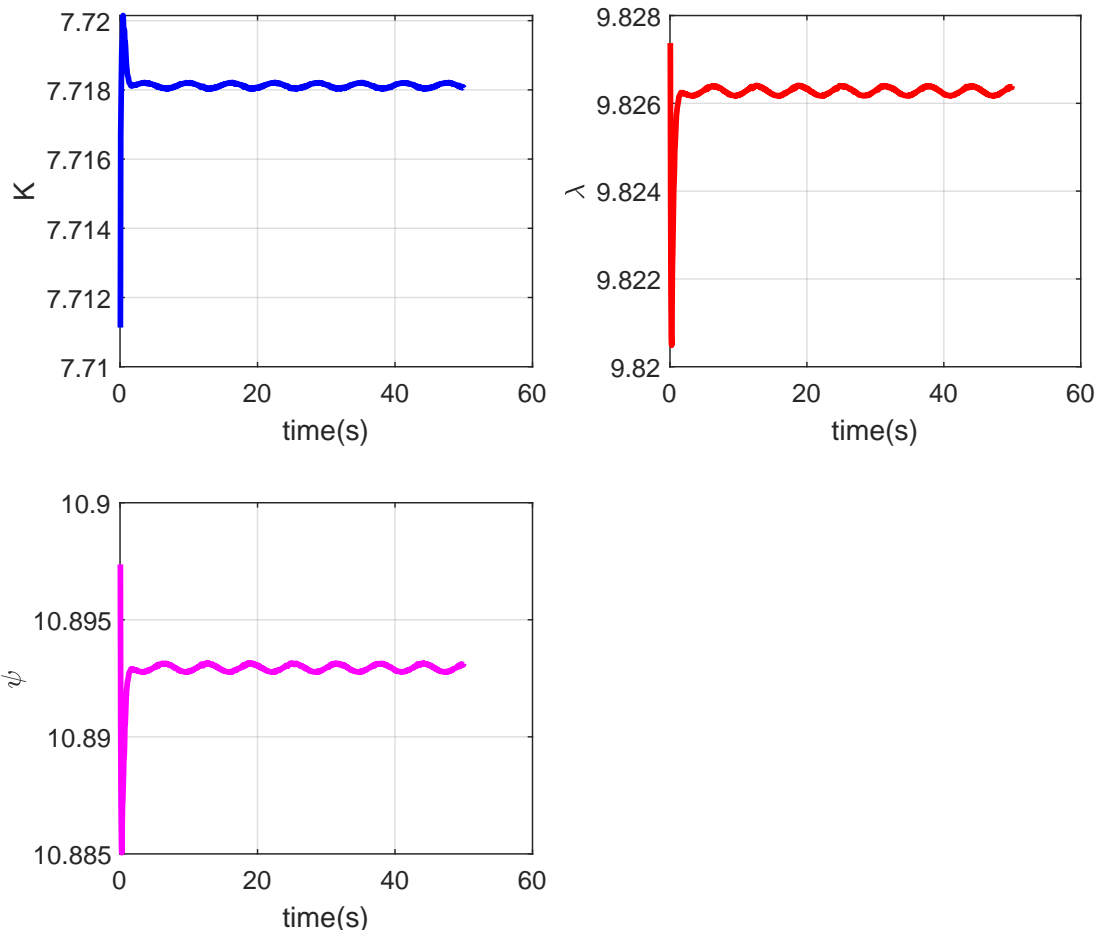


Figure 6.6: The SMC adaptive weights

To evaluate the adaptive weight adjustment, the results were compared with a fixed-

weight SMC, where the training initial values were assigned as the fixed parameters. The end-effector position with fixed and adaptive weights are shown in Fig. 6.7. The simulation results demonstrated that fixed weights led to high position errors when high disturbance was imposed upon the robot; this can be similar to the effect of the subject in hardware testing. We tested the tuner with the same reward function on LQR; the results were unsatisfactory and hence, are not presented.

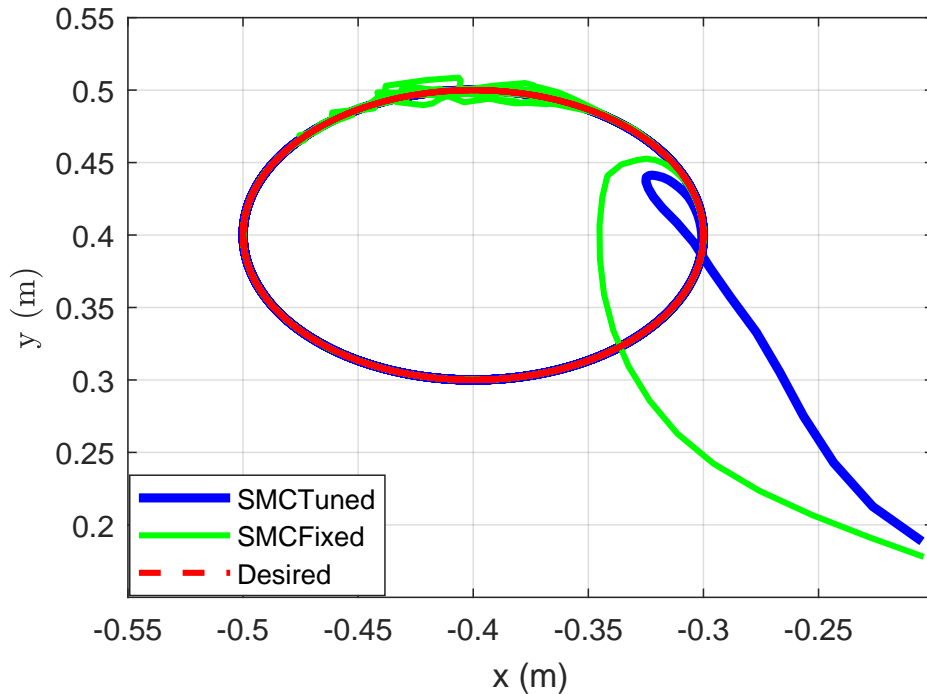


Figure 6.7: The SMC position comparison between fixed vs adaptive weights

6.5 Conclusion

In this chapter, we implemented the TD3 DRL algorithm to adaptively tune the SMC controller in simulation. An iteration-based strategy was adopted where the weights were adjusted at each iteration. There was no limitations in weight selection as TD3 can handle the infinite-dimensional continuous space. The simulation results showed considerably lower position errors compared to manually-tuned fixed weights. The tuner did not present successful weight adjustment when applied to LQR.

Chapter 7

Subject in the Loop Experimental Implementation

7.1 Overview

Subjects play a cardinal role in rehabilitation robot control. Unless considered in the control scheme, subjects might experience aggressive controller behavior as they resist the robot torque. Moreover, subject consideration enables the online monitoring of subject data and adaptively modifying the control strategy, based on patient progress. One may approach this issue by integrating the upper-extremity model with the robot model and considering the new integrated HRI model in the control design [50]. When coupled with muscle-models (as discussed in Chapter 2), this strategy can benefit the clinicians by monitoring the progress of the patient in the muscle-level and target specific muscle groups; this data can be utilized to design better controllers. Nevertheless, this strategy puts more burden on the model-based controllers, especially the ones with online optimization, because the final control-oriented HRI model is much more complicated than its robot model counterpart. Since our robot is equipped with a force/torque sensor, the patient force was directly measured and incorporated in controller design. The muscle-level information can later be obtained offline, from the force data, with forward static optimization that solves for muscle redundancy. Due to COVID 19 situation, we were not able to incorporate post-stroke patients in our experiments. Hence, we tested our controllers with healthy subjects. Generally speaking, not much can be done for directly controlling the subject behavior, i.e. the subject is not solving the inverse kinematics problem; this means that despite the advanced control strategies, the subject may show unpredicted and undesired

behavior. One line of research in this regard has focused on deep learning methods for patient pose detection [92]. A non-contacting coaching strategy can then be added to the main control strategy to alert the patient if they are not maintaining the right pose while doing the rehabilitation practices.

7.2 Experimental Considerations

The tests were performed on a healthy male subject (age: 25 years old, weight: 85 kg, height: 171 cm). The following procedures were applied to best simulate the condition of a post-stroke patient. The healthy subject was asked to use their non-dominant arm and relax their hand to avoid full muscle contraction. A strap was utilized to counterbalance the arm’s gravity force. The subject was asked to keep their shoulder as still as possible to circumvent any undesired movements when doing the practice.

7.3 Implicit Force Control

We applied an intuitive force control approach on the robot. The controller calculated the required torque $u_{controller}$ for reaching the desired trajectory, at each sample-time. The patient force $F_{subject}$ was then mapped to robot torque $u_{subject}$ via the geometric Jacobian. The difference of these values u_{robot} was then applied to the robot. The mathematical formulation of implicit force control is shown in Eqs. 7.1 and 7.2. The schematic of this controller is shown in Fig. 7.1.

$$u_{subject} = J^T(\theta_R)F_{subject} \quad (7.1)$$

$$u_{robot} = u_{controller} - u_{subject} \quad (7.2)$$

Note that the force sensor outputs the data in its local $X'Y'Z'$ frame. As a result, vector transformation was applied to acquire the global XYZ components (Fig. 7.2):

$$\begin{aligned} \{F\} &= [R(\theta_{R_1})]\{F\}' \\ R(\theta_{R_1}) &= \begin{bmatrix} \cos(\theta_{R_1}) & -\sin(\theta_{R_1}) \\ \sin(\theta_{R_1}) & \cos(\theta_{R_1}) \end{bmatrix} \end{aligned} \quad (7.3)$$

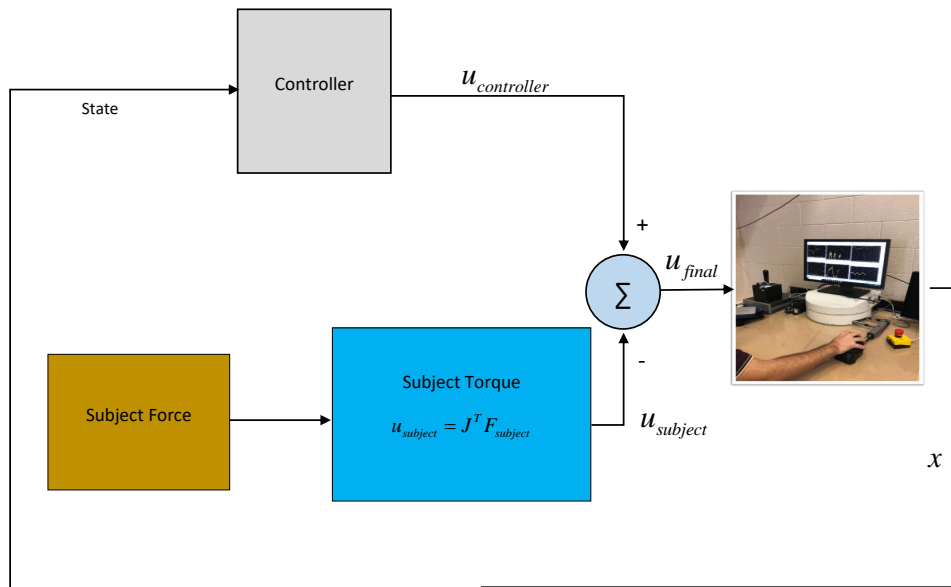


Figure 7.1: The hybrid position/force control scheme

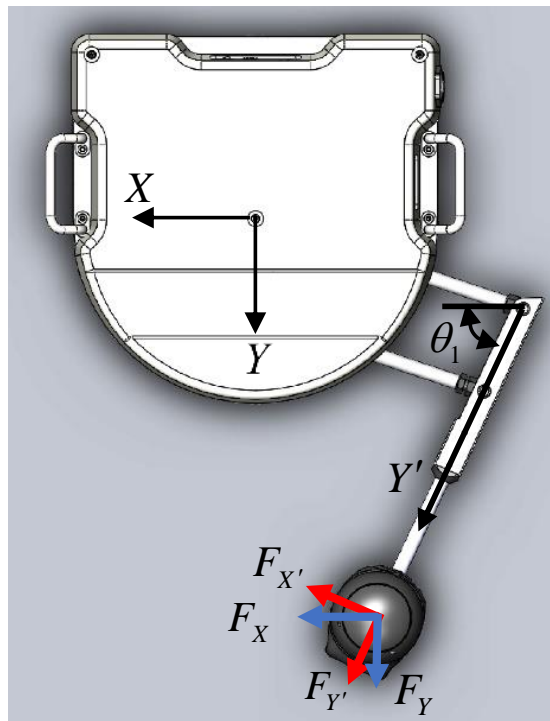


Figure 7.2: The local-global force relation.

7.4 Experimental Results

We implemented the final hybrid position/force controller on the experimental setup for tracking a circular trajectory. The data was recorded for a 40 second test. SMC and LQR position controllers were selected and implemented with the healthy user in the loop. The results are depicted in Figs. 7.3 and 7.4. SMC (with 5,813 m position error two norm) showed a better tracking performance than LQR (with 6.549 m position error two norm). As shown, the horizontal subject force $F_{subject}$ was transformed to their torque contribution $u_{subject}$ (Subject torque). As the subject exerts force in the direction that reduces the tracking error, the robot exerts less torque u_{robot} (Final torque) than the calculated torque by the controller $u_{controller}$ (Controller torque). In contrast, the robot exerts more torque as the subject increases the error. This simple strategy results in an assist-as-needed rehabilitation. Moreover, a gain matrix can be applied to the subject torque to emulate assistive/resistive control strategies.

The hybrid position/force controller was applied with LQR for tracking the systematically designed trajectory by the planner in Chapter 4. In contrast to Section 4.3, this implementation is not just the robot control; the subject is also considered in the loop. Fig. 7.5 plots the position, force, and torque results. Again, an assist-as-needed strategy is presented. Note that the robot torques were not completely the same as the open-loop torques in Section 4.2.2, but both the closed and open loop robot torques had the same order of values. Also, there was no guarantee that the subject muscle activations during the test were the same as the open-loop values in Section 4.2.2. Nevertheless, it is expected that they show similar muscle contraction patterns since the human arm terms in the cost function of the planner optimizer (Eq. 4.6) were chosen based on a study on human natural reaching movements [13]. Due to these facts, the designed trajectory is not fully optimal but is close to the optimal trajectory.

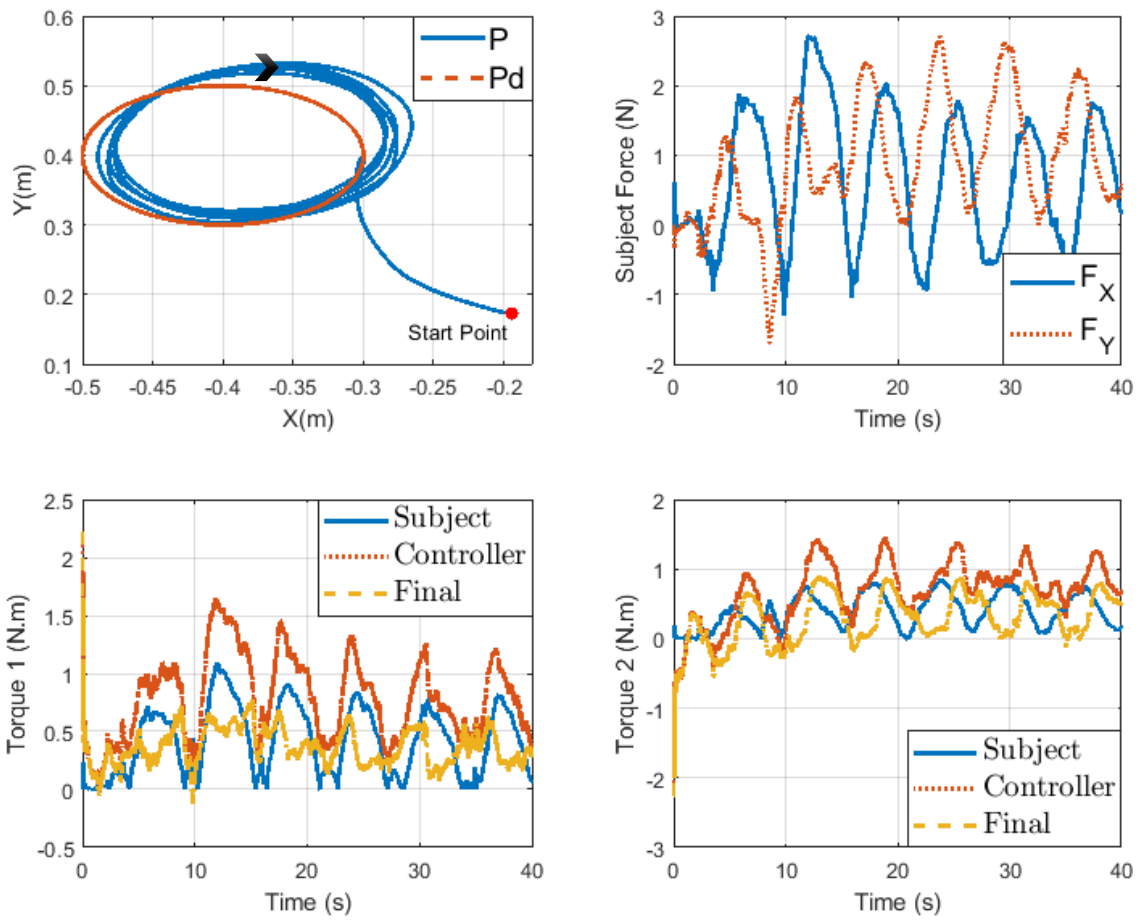


Figure 7.3: SMC subject in the loop results for the circular trajectory. The top left plot shows the end-effector position P and the desired value P_d . The top right plot depicts the subject force in the global XY coordinates. The bottom plots show the subject $u_{subject}$, controller $u_{controller}$, and final torques u_{robot} .

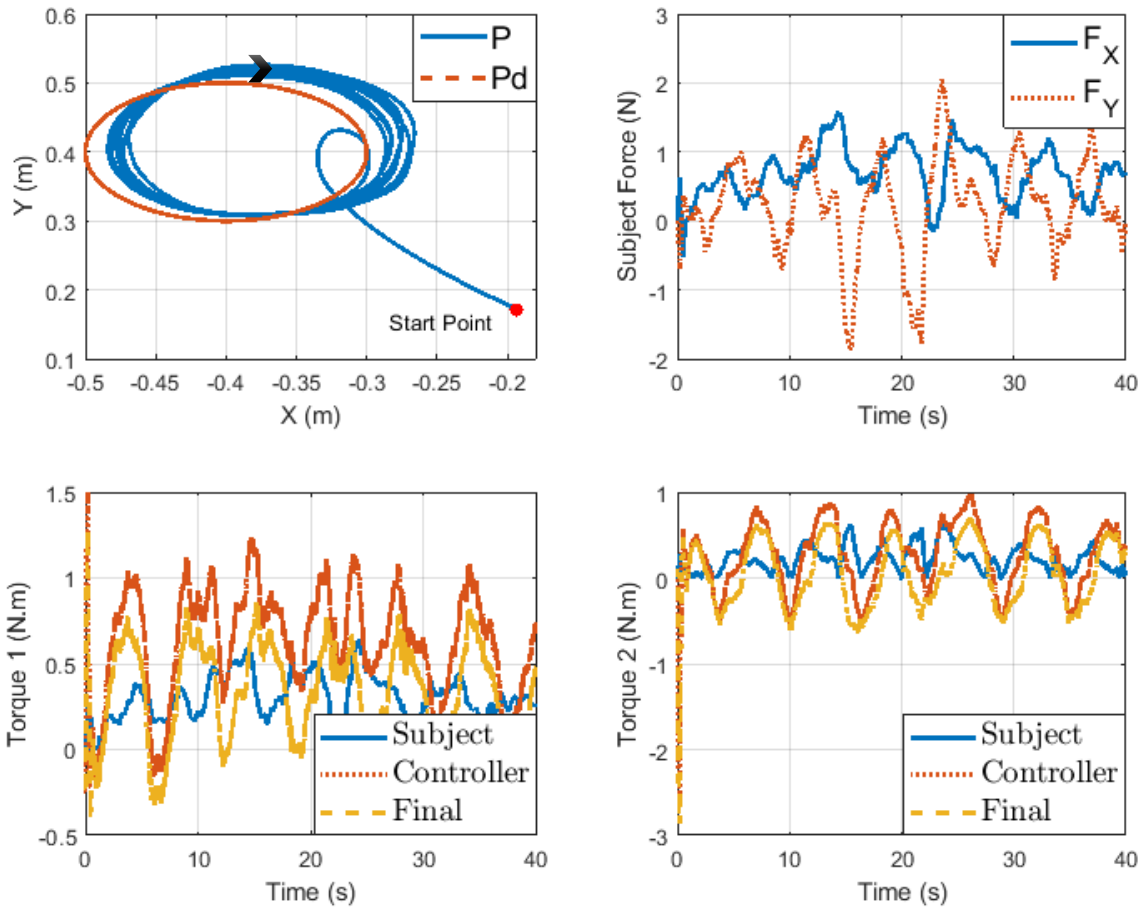


Figure 7.4: LQR subject in the loop results for the circular trajectory. The top left plot shows the end-effector position P and the desired value P_d . The top right plot depicts the subject force in the global XY coordinates. The bottom plots show the subject $u_{subject}$, controller $u_{controller}$, and final torques u_{robot} .

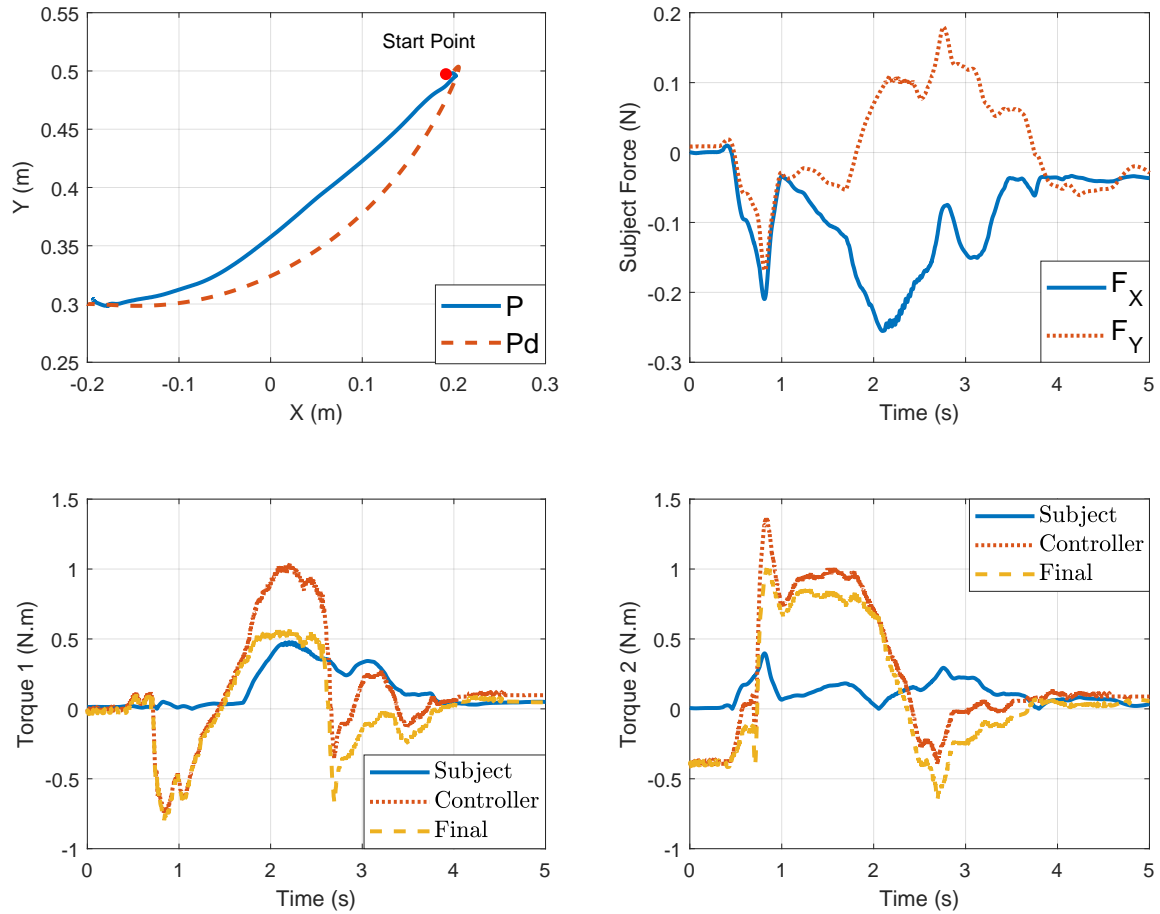


Figure 7.5: LQR subject in the loop testing for tracking the planner trajectory. The top left plot shows the end-effector position P and the desired value P_d . The top right plot depicts the subject force in the global XY coordinates. The bottom plots show the subject $u_{subject}$, controller $u_{controller}$, and final torques u_{robot} .

7.5 Conclusion

In this chapter, we presented the experimental results for the hybrid position/force controller. The procedure was discussed for emulating a post-stroke patient in the experiments, as we were confined to testing healthy subjects. The hybrid position/force control scheme allowed for an intuitive high-level assist-as-needed framework.

Chapter 8

Conclusion and Future Work

8.1 Thesis Summary

This thesis investigated the modeling, planning, model-based control, and subject-specific control tuning of a post-stroke upper extremity robot. The following presents the summary of this project:

- The high-fidelity robot model was utilized in controller testing in simulation; this model was extracted from the MapleSim model of the robot and exported to MATLAB. The low-fidelity version was developed by removing friction and joint stiffness terms and was used as the control-oriented model in our model-based controllers. A PD controller was applied to both models for control-oriented model validation. The results showed good agreement between the high-fidelity and control-oriented models.
- A novel optimization-based trajectory planner was designed for defining the optimal rehabilitation practice. The previously developed HRI model in MapleSim was used within this framework; the muscle dynamics in this model helped promote clinically plausible trajectories. Direct-collocation optimization was selected due to its capability for predicting the optimal states. The cost function included both the robot terms and human arm terms. The weights for each term can be adjusted as the rehabilitation period progresses; less weight might be needed for the human arm terms as improvement is observed in patient motor function. An LQR controller was integrated with the planner to stabilize the trajectory. Simulation and real-time

hardware implementation of the proposed approach showed successful tracking in position and velocity.

- A comparative robot control study was conducted where five controllers were evaluated in simulation and experiments. The tuning procedure was designed to remove the bias that could potentially be caused by different position tracking performances. Two scenarios of point-stabilization and circular trajectory tracking were chosen for comparison. The results demonstrated that the use of more advanced controllers than PID and computed-torque PID was justified as these two controllers showed higher inputs, input rates, and end-effector velocities than the other three controllers; potentially, these effects can reduce the hardware reliability, and patient comfort. Compared to their numerical counterparts, symbolic NMPCs represented faster responses. However, despite their excellent performance in simulation with increased sample-time (10 ms), they were still not fast enough for hardware testing with 2 ms sample-time; this was caused by the delay in communication protocols and the online optimization high turnaround time. Based on the proposed criteria, it was found that SMC and LQR were the best two candidates, the former due to its application to nonlinear systems and the latter due to its suitability for planar mechanisms.
- A DRL-based tuner framework was designed for automatic controller weight tuning. The algorithm implementation was validated by using the DRL as a controller on a pendulum swing-up problem. The tuner was implemented with the SMC controller for subject-specific controller tuning. The weights were adjusted at each iteration. To provide the patient's force in simulation, a random input was added to the controller. The application of the tuner on LQR was not satisfactory. In contrast to previous research on discrete DRL tuners, our framework was able to search the infinite-dimensional continuous space and was not limited to the discrete space. This advantage increased the accuracy of the tuning process.
- To integrate the subject in the control loop, an implicit high level force control algorithm was employed. The resulting hybrid position/force controller was tested with SMC and LQR. Due to COVID 19, testing post-stroke patients was not possible. Several considerations were adopted to simulate post-stroke conditions by the healthy subject; these included using the non-dominant hand and avoiding full muscle contraction. The hybrid controller was used for tracking a circular trajectory and the designed trajectory by the planner.

8.2 Recommendations and Future Work

The following topics are recommended for future research:

- The trajectory planner in Chapter 4 was offline, and the initial and final points were chosen before the online testing. Future research can focus on expediting the optimization. The study would enable the application of the planner in real-time and setting multiple initial and final points during patient testing. To this end, ideas from [86] can be adopted and applied to direct-collocation. Clinicians and specialists would benefit from this research by being able to set multiple initial-final points online while working with patients in experiments.
- The NMPC implementations on our setup with rapidly-updating dynamics suffered from the high iteration turnaround time. A major part of the computational inefficiency was due to the delay in the communication protocol; code generation techniques can be investigated for obtaining direct stand-alone C code to remove the need for the communication protocol. Also, the online optimizer needs to be faster. In future research, POD model reduction ideas from [83] can be examined to reduce the size of the NMPC, and make the optimization fast enough for real-time implementation.
- The final hybrid position/force controller can be applied to any assistive device equipped with force sensors. This work can be extended to exoskeletons and other rehabilitation robots in the future.
- A drawback of the high-level implicit force control is the desired trajectory enforcement; that is, as the subject deviates from the desired trajectory, a corrective torque is applied to reduce the error. The resulting behavior can lead to excessive torques in high position-error scenarios. On the other hand, the relation between position accuracy and the rehabilitation effect is not clear enough. Future work can investigate this relation and design safer high-level controllers. For instance, it might be beneficial to reset the trajectory itself based on patient performance. Using high-level scenarios like admittance control enables the adaptive adjustment of the rehabilitation practice.
- Due the time frame of the thesis, the comparative study in Chapter 5 was done by manual online tuning of the controllers. In the future, the DRL tuner can be debugged and applied to all of the controllers. Having a generalized tuning approach with the same reward function could remove any bias in tuning and facilitate the

best controller comparison structure. This structure is applicable to any autonomous dynamical system. The results can be utilized to test our current findings in Chapter 5.

- Tuner structure is not also limited to controllers. Future work could research the feasibility of this method on optimization problems. For instance, the weights of all optimizations in OptimTraj or GPOPS II can be potentially adjusted by DRL.
- Due to COVID 19 conditions, we were confined to test our controllers on healthy subjects. Testing procedures were utilized to simulate a post-stroke patient but having an actual one would be beneficial in terms of evaluating the practicality of our approaches. Hopefully, when this period ends, we can test our controllers with post-stroke subjects.

References

- [1] Syed Faiz Ahmed, Athar Ali, Syed Yarooq Raza, Kushsairy A. Kadir, M. Kamran Joyo, and Kanendra Naidu. Model predictive control for upper limb rehabilitation robotic system under disturbed condition. *AIP Conference Proceedings*, 2129(July), 2019.
- [2] Qingsong Ai, Chengxiang Zhu, Jie Zuo, Wei Meng, Quan Liu, Sheng Q. Xie, and Ming Yang. Disturbance-estimated adaptive backstepping sliding mode control of a pneumatic muscles-driven ankle rehabilitation robot. *Sensors (Switzerland)*, 18(1), 2018.
- [3] Noppadol Ajjanaromvat and Manukid Parnichkun. Trajectory tracking using online learning LQR with adaptive learning control of a leg-exoskeleton for disorder gait rehabilitation. *Mechatronics*, 51(March):85–96, 2018.
- [4] Alessandro Alessio and Alberto Bemporad. A survey on explicit model predictive control. *Lecture Notes in Control and Information Sciences*, 384:345–369, 2009.
- [5] Athar Ali, Syed Faiz Ahmed, M. Kamran Joyo, and K. Kushsairy. MPC-PID comparison for controlling therapeutic upper limb rehabilitation robot under perturbed conditions. *2017 IEEE 3rd International Conference on Engineering Technologies and Social Sciences, ICETSS 2017*, 2018-Janua:1–5, 2018.
- [6] Athar Ali, Syed Faiz Ahmed, Kushsairy A. Kadir, M. Kamran Joyo, and R. N.S. Yarooq. Fuzzy PID controller for upper limb rehabilitation robotic system. *2018 IEEE International Conference on Innovative Research and Development, ICIRD 2018*, (May):1–5, 2018.
- [7] F Allg’ower, TA Badgwell, JS Qin, JB Rawlings, and SJ Wright. Advances in control/highlights of ecc’99//chapt. 12. nonlinear predictive controls and moving horizon estimation, 1999.

- [8] Joel Andersson, Johan Åkesson, and Moritz Diehl. Casadi: A symbolic package for automatic differentiation and optimal control. In *Recent advances in algorithmic differentiation*, pages 297–307. Springer, 2012.
- [9] Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.
- [10] Karl Johan Åström and Tore Hägglund. *PID controllers: theory, design, and tuning*, volume 2. Instrument society of America Research Triangle Park, NC, 1995.
- [11] L. Beiner and S. W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.
- [12] Richard Bellman. *The Theory of Dynamic Programming*, 1954.
- [13] Bastien Berret, Enrico Chiovetto, Francesco Nori, and Thierry Pozzo. Evidence for composite cost functions in arm movement planning: An inverse optimal control approach. *PLoS Computational Biology*, 7(10), 2011.
- [14] Bastien Berret, Christian Darlot, Frédéric Jean, Thierry Pozzo, Charalambos Papaxanthis, and Jean Paul Gauthier. The inactivation principle: Mathematical solutions minimizing the absolute work and biological implications for the planning of arm movements. *PLoS Computational Biology*, 4(10), 2008.
- [15] Rachele Bertani, Corrado Melegari, C Maria, Alessia Bramanti, Placido Bramanti, and Rocco Salvatore Calabrò. Effects of robot-assisted upper limb rehabilitation in stroke patients: a systematic review with meta-analysis. *Neurological Sciences*, 38(9):1561–1569, 2017.
- [16] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-Dynamic Programming : An Overview*. (December), 1995.
- [17] Shalabh Bhatnagar, Mohammad Ghavamzadeh, Mark Lee, and Richard S Sutton. Incremental natural actor-critic algorithms. In *Advances in neural information processing systems*, pages 105–112, 2008.
- [18] Enrico Bibbona, Gianna Panfilo, and Patrizia Tavella. The ornstein–uhlenbeck process as a model of a low pass filtered white noise. *Metrologia*, 45(6):S117, 2008.

- [19] Michelangelo Bin, Daniele Astolfi, Lorenzo Marconi, and Laurent Praly. About robustness of internal model-based control for linear and nonlinear systems. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 5397–5402. IEEE, 2018.
- [20] Amy A. Blank, James A. French, Ali Utku Pehlivan, and Marcia K. O’Malley. Current Trends in Robot-Assisted Upper-Limb Stroke Rehabilitation: Promoting Patient Engagement in Therapy. *Current Physical Medicine and Rehabilitation Reports*, 2(3):184–195, 2014.
- [21] Ruth Bonita, Alistair Stewart, and Robert Beaglehole. International trends in stroke mortality: 1970–1985. *Stroke*, 21(7):989–992, 1990.
- [22] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *2. Some Standard Problems Involving LMIs*. 1994.
- [23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [24] Colin Brown. Predictive Forward Dynamic Simulation of Manual Wheelchair Propulsion by. (September), 2018.
- [25] Peter Brown. Contact Modelling for Forward Dynamics of Human Motion. 2017.
- [26] Peter Brown and John McPhee. A Continuous Velocity-Based Friction Model for Dynamics and Control with Physically Meaningful Parameters. *Journal of Computational and Nonlinear Dynamics*, 11(5):1–6, 2016.
- [27] Ming Kun Chang. An adaptive self-organizing fuzzy sliding mode controller for a 2-DOF rehabilitation robot actuated by pneumatic muscle actuators. *Control Engineering Practice*, 18(1):13–22, 2010.
- [28] Won Hyuk Chang and Yun-Hee Kim. Robot-assisted therapy in stroke rehabilitation. *Journal of stroke*, 15(3):174, 2013.
- [29] Richard Cheng, Abhinav Verma, Gabor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel W Burdick. Control regularization for reduced variance reinforcement learning. *arXiv preprint arXiv:1905.05380*, 2019.
- [30] Luigi del Re, Ilya Kolmanovsky, Maarten Steinbuch, and Harald Waschl. Introduction. *Lecture Notes in Control and Information Sciences*, 455 LNCIS, 2014.

- [31] Christopher L Dembia, Amy Silder, Thomas K Uchida, Jennifer L Hicks, and Scott L Delp. Simulating ideal assistive devices to reduce the metabolic cost of walking with heavy loads. *PloS one*, 12(7):e0180320, 2017.
- [32] Mouna Doghri, Sophie Duchesne, Annie Poulin, and Maxim Ouellet. Comparative study of pressure reduction valve controllers in water distribution systems. In *Euro-Mediterranean Conference for Environmental Integration*, pages 1001–1003. Springer, 2017.
- [33] Minh Quan Duong, Sonia Leva, Marco Mussetta, and Kim Hung Le. A comparative study on controllers for improving transient stability of dfig wind turbines during large disturbances. *Energies*, 11(3):480, 2018.
- [34] Nancy E. Mayo, Sharon Wood-Dauphinee, Sara Ahmed, Gordon Carron, Johanne Higgins, Sara Mcewen, and Nancy Salbach. Disablement following stroke. *Disability and rehabilitation*, 21(5-6):258–268, 1999.
- [35] Russell Eberhart and James Kennedy. A New Optimizer Using Particle Swarm Theory. *Sixth International Symposium on Micro Machine and Human Science*, 0-7803-267:39–43, 1999.
- [36] Christopher Edwards and Sarah Spurgeon. *Sliding mode control: theory and applications*. Crc Press, 1998.
- [37] Aulia El Hakim, Hilwadi Hindersah, and Estiko Rijanto. Application of reinforcement learning on self-tuning pid controller for soccer robot multi-agent system. In *2013 joint international conference on rural information & communication technology and electric-vehicle technology (rICT & ICeV-T)*, pages 1–6. IEEE, 2013.
- [38] Sascha E. Engelbrecht and Juan Pablo Fernández. Invariant characteristics of horizontal-plane minimum-torque-change movements with one mechanical degree of freedom. *Biological Cybernetics*, 76(5):321–329, 1997.
- [39] Boudjema Fares. Robust Control via Sequential Semidefinite Programming. pages 2750–2755.
- [40] Susan E Fasoli, Hermano I Krebs, Joel Stein, Walter R Frontera, and Neville Hogan. Effects of robotic therapy on motor impairment and recovery in chronic stroke. *Archives of physical medicine and rehabilitation*, 84(4):477–482, 2003.

- [41] Yongfei Feng, Hongbo Wang, Yaxin Du, Fei Chen, Hao Yan, and Hongfei Yu. Trajectory planning of a novel lower limb rehabilitation robot for stroke patient passive training. *Advances in Mechanical Engineering*, 9(12):1–10, 2017.
- [42] Rolf Findeisen, Lars Imsland, Frank Allgower, and Bjarne A Foss. State and output feedback nonlinear model predictive control: An overview. *European journal of control*, 9(2-3):190–206, 2003.
- [43] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.
- [44] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.
- [45] Olof Garpinger, Tore Hägglund, and Karl Johan Åström. Performance and robustness trade-offs in pid control. *Journal of Process Control*, 24(5):568–577, 2014.
- [46] Gregory R. Gay, Paola Salomoni, and Silvia Mirri. E-Learning. *Encyclopedia of Internet Technologies and Applications*, 292:179–184, 2007.
- [47] Borna Ghannadi. Model-based control of upper extremity human-robot rehabilitation systems. 2017.
- [48] Borna Ghannadi and John McPhee. Optimal impedance control of an upper limb stroke rehabilitation robot. *ASME 2015 Dynamic Systems and Control Conference, DSCC 2015*, 1:1–7, 2015.
- [49] Borna Ghannadi, Naser Mehrabi, and John McPhee. Development of a human-robot dynamic model to support model-based control design of an upper limb rehabilitation robot. *Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics 2015, Multibody Dynamics 2015*, pages 999–1007, 2015.
- [50] Borna Ghannadi, Naser Mehrabi, Reza Sharif Razavian, and John McPhee. Non-linear model predictive control of an upper extremity rehabilitation robot using a two-dimensional human-robot interaction model. *IEEE International Conference on Intelligent Robots and Systems*, 2017-September:502–507, 2017.
- [51] Borna Ghannadi, Reza Sharif Razavian, and John McPhee. Upper extremity rehabilitation robots: A survey. *Handbook of Biomechatronics*, page 319, 2018.

- [52] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.
- [53] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.
- [54] Zenon Hendzel and Marcin Szuster. Discrete neural dynamic programming in wheeled mobile robot control. *Communications in Nonlinear Science and Numerical Simulation*, 16(5):2355–2362, 2011.
- [55] Marco Herrera, Andrés Cuaycal, Oscar Camacho, and David Pozo. Lqr discrete controller tuning for a twip robot based on genetic algorithms. In *2019 International Conference on Information Systems and Computer Science (INCISCOS)*, pages 163–168. IEEE, 2019.
- [56] Bryce Hosking. Modelling and Model Predictive Control of Power-Split Hybrid Powertrains for Self-Driving Vehicles by. 2018.
- [57] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.
- [58] Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Ontinuous learning control with deep reinforcement. 2016.
- [59] Irina Ioachim, Sylvie Gélinas, François Soumis, and Jacques Desrosiers. A Dynamic Programming Algorithm for the Shortest Path Problem with Time Windows and Linear Node Costs. *Networks*, 31(3):193–204, 1998.
- [60] P Travis Jardine, Michael Kogan, Sidney N Givigi, and Shahram Yousefi. Adaptive predictive control of a differential drive robot tuned with reinforcement learning. *International Journal of Adaptive Control and Signal Processing*, 33(2):410–423, 2019.
- [61] S.K. Jha et al. Comparative study of different classical and modern control techniques for the position control of sophisticated mechatronic system. *Procedia Computer Science*, 93:1038–1045, 2016.

- [62] Tor A Johansen. Introduction to nonlinear model predictive control and moving horizon estimation. *Selected topics on constrained and nonlinear control*, 1:1–53, 2011.
- [63] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.
- [64] Hadi Kalani, Alireza Akbarzadeh, S. Nader Nabavi, and Sahar Moghimi. Dynamic modeling and CPG-based trajectory generation for a masticatory rehab robot. *Intelligent Service Robotics*, 11(2):187–205, 2018.
- [65] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.
- [66] Matthew P. Kelly. Transcription Methods for Trajectory Optimization: a beginners tutorial. pages 1–14, 2017.
- [67] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.
- [68] Hermano Igo Krebs. *Twenty + years of robotics for upper-extremity rehabilitation following a stroke*. Elsevier Ltd., 2018.
- [69] N Senthil Kumar, V Sadasivam, and HM Asan Sukriya. A comparative study of pi, fuzzy, and ann controllers for chopper-fed dc drive with embedded systems approach. *Electric Power Components and Systems*, 36(7):680–695, 2008.
- [70] Gert Kwakkel, Boudewijn J Kollen, Jeroen van der Grond, and Arie JH Prevo. Probability of regaining dexterity in the flaccid upper limb: impact of severity of paresis and time since onset in acute stroke. *Stroke*, 34(9):2181–2186, 2003.
- [71] Clemente Laurettil, Francesca Cordella, Eugenio Guglielmelli, and Loredana Zollo. Learning by Demonstration for Planning Activities of Daily Living in Rehabilitation and Assistive Robotics. *IEEE Robotics and Automation Letters*, 2(3):1375–1382, 2017.
- [72] G. Lee, J. Kim, F. A. Panizzolo, Y. M. Zhou, L. M. Baker, I. Galiana, P. Malcolm, and C. J. Walsh. Reducing the metabolic cost of running with a tethered soft exosuit. *Science Robotics*, 2(6):1–3, 2017.

- [73] Frank L. Lewis and Draguna Vrabie. Adaptive dynamic programming for feedback control. *Proceedings of 2009 7th Asian Control Conference, ASCC 2009*, pages 1402–1409, 2009.
- [74] Yiqing Li, Wen Zhou, Yan Cao B, and Feng Jia. *Design of Embedded Structure Variable*, volume 1. Springer International Publishing, 2019.
- [75] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [76] M Patrice Lindsay, Bo Norrving, Ralph L Sacco, Michael Brainin, Werner Hacke, Sheila Martins, Jeyaraj Pandian, and Valery Feigin. World stroke organization (wso): global stroke fact sheet 2019, 2019.
- [77] Fengjin Liu, William W Hager, and Anil V Rao. Adaptive mesh refinement method for optimal control using nonsmoothness detection and mesh size reduction. *Journal of the Franklin Institute*, 352(10):4081–4106, 2015.
- [78] Jian-xun Liu, YH Li, Yong-gang CHEN, and Cheng WANG. Aeroengine lqr control based on fuzzy-neural networks. *Journal of Aerospace Power*, 19(6):838–843, 2004.
- [79] XY Liu and AJ Forsyth. Comparative study of stabilizing controllers for brushless dc motor drive systems. In *IEEE International Conference on Electric Machines and Drives, 2005.*, pages 1725–1731. IEEE, 2005.
- [80] Zemin Liu, Qingsong Ai, Yaojie Liu, Jie Zuo, Xiong Zhang, Wei Meng, and Shane Xie. An optimal motion planning method of 7-DOF robotic arm for upper limb movement assistance. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, 2019-July:277–282, 2019.
- [81] Cynthia Lyles-Scott. A slave by any other name: Names and identity in Toni Morrison’s *Beloved*. *Names*, 56(1):23–28, 2008.
- [82] Lalo Magni, Davide Martino Raimondo, and Frank Allgwer. Lecture Notes in Control and Information Sciences: Preface. *Lecture Notes in Control and Information Sciences*, 384(May 2014), 2009.
- [83] Anson Maitland. Nonlinear model predictive control reduction strategies for real-time optimal control. 2019.

- [84] Anson Maitland, Mohit Batra, and John McPhee. Nonlinear model predictive control reduction using truncated single shooting. In *2018 Annual American Control Conference (ACC)*, pages 3165–3170. IEEE, 2018.
- [85] Anson Maitland and John McPhee. Improving model predictive controller turnaround time using restricted lagrangians. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 3811–3816. IEEE, 2017.
- [86] Anson Maitland and John McPhee. Quasi-translations for fast hybrid nonlinear model predictive control. *Control Engineering Practice*, 97:104352, 2020.
- [87] Ian R. Manchester. Transverse dynamics and regions of stability for nonlinear hybrid limit cycles. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 44(1 PART 1):6285–6290, 2011.
- [88] Aitziber Mancisidor, Asier Zubizarreta, Itziar Cabanes, Pablo Bengoa, and Je Hyung Jung. New Trends in Medical and Service Robots. *New Trends in Medical and Service Robots Design, Analysis and Control*, 48(April 2017):117–130, 2018.
- [89] Laura Marchal-Crespo and David J Reinkensmeyer. Review of control strategies for robotic movement training after neurologic injury. *Journal of neuroengineering and rehabilitation*, 6(1):1–15, 2009.
- [90] Felipe N. Martins, Wanderley C. Celeste, Ricardo Carelli, Mário Sarcinelli-Filho, and Teodiano F. Bastos-Filho. An adaptive dynamic controller for autonomous mobile robot trajectory tracking. *Control Engineering Practice*, 16(11):1354–1363, 2008.
- [91] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Sokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.
- [92] William McNally, Kanav Vats, Alexander Wong, and John McPhee. Evopose2d: Pushing the boundaries of 2d human pose estimation using neuroevolution. *arXiv preprint arXiv:2011.08446*, 2020.
- [93] Mohit Mehndiratta, Efe Camci, and Erdal Kayacan. Automated tuning of nonlinear model predictive controller by reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3016–3021. IEEE, 2018.

- [94] Naser Mehrabi, Reza Sharif Razavian, and John McPhee. A Physics-Based Musculoskeletal Driver Model to Study Steering Tasks. *Journal of Computational and Nonlinear Dynamics*, 10(2):1–8, 2015.
- [95] Qiaoling Meng, Qiaolian Xie, Zhimeng Deng, and Hongliu Yu. *A general kinematics model for trajectory planning of upper limb exoskeleton robots*, volume 11745 LNAI. Springer International Publishing, 2019.
- [96] Michael A.Henson. Nonlinear model predictive control current states and future directions.pdf, 1998.
- [97] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.
- [98] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [99] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [100] Luke M Mooney, Elliott J Rouse, and Hugh M Herr. Autonomous exoskeleton reduces metabolic cost of human walking during load carriage. *Journal of neuroengineering and rehabilitation*, 11(1):1–11, 2014.
- [101] Manfred Morari and Jay H Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4-5):667–682, 1999.
- [102] Hafiz Muhammad Yasir Naeem and A Mahmood. Autonomous cruise control of car using lqr and h2 control algorithm. In *2016 International Conference on Intelligent Systems Engineering (ICISE)*, pages 123–128. IEEE, 2016.
- [103] Arne J. Nagengast, Daniel A. Braun, and Daniel M. Wolpert. Optimal control predicts human performance on objects with internal degrees of freedom. *PLoS Computational Biology*, 5(6), 2009.

- [104] Hirofumi Nakayama, Henrik Stig Jørgensen, Hans Otto Raaschou, and Tom Skyhøj Olsen. Recovery of upper extremity function in stroke patients: the copenhagen stroke study. *Archives of physical medicine and rehabilitation*, 75(4):394–398, 1994.
- [105] World Health Organization. *The world health report 2002: reducing risks, promoting healthy life*. World Health Organization, 2002.
- [106] World Health Organization et al. *The global burden of disease: 2004 update*. World Health Organization, 2008.
- [107] Lizheng Pan, Aiguo Song, Suolin Duan, and Xianchuan Shi. Study on motion performance of robot-aided passive rehabilitation exercises using novel dynamic motion planning strategy. *International Journal of Advanced Robotic Systems*, 16(5):1–15, 2019.
- [108] Michael A Patterson and Anil V Rao. Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1):1–37, 2014.
- [109] James L Patton, Mary Ellen Stoykov, Mark Kovic, and Ferdinando A Mussa-Ivaldi. Evaluation of robotic training forces that either enhance or reduce error in chronic hemiparetic stroke survivors. *Experimental brain research*, 168(3):368–383, 2006.
- [110] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.
- [111] David Pinto-Fernandez, Diego Torricelli, Maria del Carmen Sanchez-Villamanan, Felix Aller, Katja Mombaur, Roberto Conti, Nicola Vitiello, Juan C Moreno, and Jose Luis Pons. Performance evaluation of lower limb exoskeletons: a systematic review. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(7):1573–1583, 2020.
- [112] Francesca Pistoia, Simona Sacco, Cindy Tiseo, Diana Degan, Raffaele Ornello, and Antonio Carolei. The epidemiology of atrial fibrillation and stroke. *Cardiology clinics*, 34(2):255–268, 2016.
- [113] Matthias Plappert, Rein Houthoofd, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

- [114] Warren B. Powell, Abraham George, Belgacem Bouzaiene-Ayari, and Hugo P. Simao. Approximate dynamic programming for high dimensional resource allocation problems. *Proceedings of the International Joint Conference on Neural Networks*, 5:2989–2994, 2005.
- [115] S Joe Qin and Thomas A Badgwell. An overview of nonlinear model predictive control applications. In *Nonlinear model predictive control*, pages 369–392. Springer, 2000.
- [116] S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.
- [117] J. Randall Flanagan and David J. Ostry. Trajectories of human multi-joint arm movements: Evidence of joint level planning. *Experimental Robotics I*, pages 594–613, 2006.
- [118] Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.
- [119] G Rosati, G Volpe, and A Biondi. Trajectory planning of a two-link rehabilitation robot arm. *IFYolVIM World Congress*, 2007.
- [120] Muhammad Ahsan Saeed, Nisar Ahmed, Mujahid Hussain, and Adnan Jafar. A comparative study of controllers for optimal speed control of hybrid electric vehicle. In *2016 International Conference on Intelligent Systems Engineering (ICISE)*, pages 1–4. IEEE, 2016.
- [121] Yahaya Md Sam, Mohd Ruddin Hj Abdul Ghani, and Nasarudin Ahmad. LQR controller for active car suspension. *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, 1(February), 2000.
- [122] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.
- [123] Jean-Jacques E Slotine. Sliding controller design for non-linear systems. *International Journal of control*, 40(2):421–434, 1984.
- [124] David K. Smith and Dimitri P. Bertsekas. Dynamic Programming and Optimal Control. Volume 1 Dynamic Programming and Optimal Control. Volume 2. *The Journal of the Operational Research Society*, 47(6):833, 1996.

- [125] Chun-Yi Su, Subhash Rakheja, and Honghai Liu. *Intelligent Robotics and Applications Part 2*. 2012.
- [126] Liang Sun and Jiafei Gan. Researching of two-wheeled self-balancing robot base on lqr combined with pid. In *2010 2nd International Workshop on Intelligent Systems and Applications*, pages 1–5. IEEE, 2010.
- [127] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, 1998.
- [128] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.
- [129] M. Suzuki, Y. Yamazaki, N. Mizuno, and K. Matsunami. Trajectory formation of the center-of-mass of the arm during reaching movements. *Neuroscience*, 76(2):597–610, 1997.
- [130] Sadegh Tajeddin, Mahyar Vajedi, and Nasser L Azad. A newton/gmres approach to predictive ecological adaptive cruise control of a plug-in hybrid electric vehicle in car-following scenarios. *IFAC-PapersOnLine*, 49(21):59–65, 2016.
- [131] Shangjie Tang, Lin Chen, Michele Barsotti, Lintao Hu, Yongqiang Li, Xiaoying Wu, Long Bai, Antonio Frisoli, and Wensheng Hou. Kinematic synergy of multi-DOF movement in upper limb and its application for rehabilitation exoskeleton motion planning. *Frontiers in Neurorobotics*, 13(November), 2019.
- [132] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation. *Lecture Notes*, 2016.
- [133] Tatsuya Teramae, Tomoyuki Noda, and Jun Morimoto. EMG-Based Model Predictive Control for Physical Human-Robot Interaction: Application for Assist-As-Needed Control. *IEEE Robotics and Automation Letters*, 3(1):210–217, 2018.
- [134] Darryl G Thelen. Adjustment of muscle mechanics model parameters to simulate dynamic contractions in older adults. *J. Biomech. Eng.*, 125(1):70–77, 2003.
- [135] Christoph Tillmann and Hermann Ney. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, 29(1):97–133, 2003.

- [136] James Y Tung, Brent Stead, William Mann, Ba' Pham, and Milos R Popovic. Assistive technologies for self-managed pressure ulcer prevention in spinal cord injury: a scoping review. *J Rehabil Res Dev*, 52(2):131–46, 2015.
- [137] N Kemal Ure, M Ugur Yavas, Ali Alizadeh, and Can Kurtulus. Enhancing situational awareness and performance of adaptive cruise control through model predictive control and deep reinforcement learning. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 626–631. IEEE, 2019.
- [138] Yasaman Vaghei, Ahmad Ghanbari, and Sayyed Mohammad Reza Sayyed Noorani. Actor-critic neural network reinforcement learning for walking control of a 5-link bipedal robot. *2014 2nd RSI/ISM International Conference on Robotics and Mechatronics, ICRoM 2014*, pages 773–778, 2014.
- [139] R PS Van Peppen, Gert Kwakkel, Sharon Wood-Dauphinee, H JM Hendriks, Ph J Van der Wees, and Joost Dekker. The impact of physical therapy on functional outcomes after stroke: what's the evidence? *Clinical rehabilitation*, 18(8):833–862, 2004.
- [140] Oscar von Stryk and Roland Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, 37(1):357–373, 1992.
- [141] Ke Yi Wang, Peng Cheng Yin, Hai Peng Yang, and Xiao Qiang Tang. The man-machine motion planning of rigid-flexible hybrid lower limb rehabilitation robot. *Advances in Mechanical Engineering*, 10(6):1–11, 2018.
- [142] Lynne M Weber and Joel Stein. The use of robots in stroke rehabilitation: A narrative review. *NeuroRehabilitation*, 43(1):99–110, 2018.
- [143] Yue Wen, Jennie Si, Andrea Brandt, Xiang Gao, and He Helen Huang. Online reinforcement learning control for the personalization of a robotic knee prosthesis. *IEEE transactions on cybernetics*, 50(6):2346–2356, 2019.
- [144] Yue Wen, Jennie Si, Xiang Gao, Stephanie Huang, and He Helen Huang. A New Powered Lower Limb Prosthesis Control Framework Based on Adaptive Dynamic Programming. *IEEE Transactions on Neural Networks and Learning Systems*, 28(9):2215–2220, 2017.
- [145] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

- [146] Stephen Wright. On the convergence of the newton/log-barrier method. In *Preprint ANL/MCSP681-0897, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill.* Citeseer, 1997.
- [147] Jun Wu, Jian Huang, Yongji Wang, and Kexin Xing. RLS-ESN based PID control for rehabilitation robotic arms driven by PM -TS actuators. *2010 International Conference on Modelling, Identification and Control, ICMIC 2010*, pages 511–516, 2010.
- [148] Qingcong Wu, Bai Chen, and Hongtao Wu. Adaptive admittance control of an upper extremity rehabilitation robot with neural-network-based disturbance observer. *IEEE Access*, 7:123807–123819, 2019.
- [149] Fitri Yakub, Ahmad Zahran Md Khudzari, and Yasuchika Mori. Recent trends for practical rehabilitation robotics, current challenges and the future. *International Journal of Rehabilitation Research*, 37(1):9–21, 2014.
- [150] Deokwon Yun, Abdul Manan Khan, Rui Jun Yan, Younghoon Ji, Hyeyoun Jang, Junaid Iqbal, K. M. Zuhaib, Jae Yong Ahn, Jungsoo Han, and Changsoo Han. Handling subject arm uncertainties for upper limb rehabilitation robot using robust sliding mode control. *International Journal of Precision Engineering and Manufacturing*, 17(3):355–362, 2016.
- [151] Mingming Zhang, Andrew McDaid, Allan J Veale, Yuxin Peng, and Sheng Quan Xie. Adaptive trajectory tracking control of a parallel ankle rehabilitation robot with joint-space force distribution. *IEEE Access*, 7:85812–85820, 2019.
- [152] Shiyu Zhang, Andrea Maria Zanchettin, Renzo Villa, and Shuling Dai. Real-time trajectory planning based on joint-decoupled optimization in human-robot interaction. *Mechanism and Machine Theory*, 144:103664, 2020.
- [153] Jinghua Zhong. Pid controller tuning: A short tutorial. *Mechanical Engineering, Purdue University*, pages 1–10, 2006.

APPENDICES

Appendix A

Stability Proof of Sliding Mode Control

As mentioned in Section 5.2.3, we defined a sliding surface with one relative degree:

$$S = \dot{\epsilon} + \lambda\epsilon \quad (\text{A.1})$$

Considering the general system dynamics with model and disturbance uncertainties:

$$\ddot{x} = f_m(x, t) + d_m(t) + \Delta f(x, t) + \Delta d(t) + u \quad (\text{A.2})$$

The derivative of the sliding surface is as follows:

$$\dot{S} = f_m(x, t) + d_m(t) + \Delta f(x, t) + \Delta d(t) + u - \ddot{x}_d + \lambda\dot{\epsilon} \quad (\text{A.3})$$

The u is then used to cancel out the known terms:

$$u = -f_m(x, t) - d_m(t) + \ddot{x}_d - \lambda\dot{\epsilon} + \nu \quad (\text{A.4})$$

The term ν encompasses all the uncertain terms. Using the defined u , the derivative of the sliding surface is shown below:

$$\dot{S} = \Delta f(x, t) + \Delta d(t) + \nu \quad (\text{A.5})$$

Considering upper bounds on the uncertain terms:

$$\begin{aligned} |\Delta f(x, t)| &\leq \alpha(x, t) > 0 \\ |\Delta d(t)| &\leq \beta(t) > 0 \end{aligned} \quad (\text{A.6})$$

We select a Lyapunov function as below. The candidate is positive for all S :

$$V = \frac{S^2}{2} \quad (\text{A.7})$$

Now, we should assure that the derivative of the Lyapunov function is negative definite to confirm that the \dot{S} is asymptotically stable:

$$\dot{V} = S\dot{S} \quad (\text{A.8})$$

Hence, we need to calculate ν such that $\dot{V} = S\dot{S} < 0$:

$$S\dot{S} = S(\Delta f(x, t) + \Delta d(t) + \nu) \leq S(\alpha + \beta)\text{sign}(S) + S\nu \quad (\text{A.9})$$

The application of the $\text{sign}(s)$ function is because we have defined our bounds on the absolute value of the $\Delta f(x, t)$ and $\Delta d(t)$. Then we can set ν as:

$$\begin{aligned} \nu &= -(\zeta + \alpha + \beta)\text{sign}(S) \\ \zeta &> 0 \end{aligned} \quad (\text{A.10})$$

Defining ν as above results in a negative definite Lyapunov function:

$$S\dot{S} = -\zeta S\text{sign}(S) < 0 \quad (\text{A.11})$$

By defining a new weight $K = \zeta + \alpha + \beta$ and considering the boundary layer ψ , we reach the same formulation as in Section 5.2.3.

Appendix B

Controller Parameters

In this section, we present the controller parameters used in Chapter 5. Table B.1 shows the parameters of the simulation results presented in Section 5.3, and Table B.2 depicts the parameters for experimental implementations in Section 5.4.

Table B.1: Controller Parameters in Simulation

Controller	Parameters
PID	$k_p = 15$ $k_d = 7$ $k_i = 0$
Comp PID	$k_p = 5$ $k_d = 3$ $k_i = 0$
LQR	$Q = \text{Diag}(1000, 1000, 100, 100)$ $R = \text{Diag}(200, 200)$
SMC	$K = [1, 1]^T$ $\lambda = 5$ $\psi = 2$
CasADI-based NMPC	$Q = \text{Diag}(5000, 5000, 50, 50)$ $R = \text{Diag}(2.5, 2.5)$
Maple-based NMPC	$Q = \text{Diag}(250000, 250000, 500, 500)$ $R_1 = \text{Diag}(20, 20)$ $R_2 = \text{Diag}(0, 0)$

Table B.2: Controller Parameters in Hardware Implementation

Controller	Parameters
PID	$k_p = 15$ $k_d = 1.5$ $k_i = 2$
Comp PID	$k_p = 60$ $k_d = 10$ $k_i = 5$
LQR	$Q = \text{Diag}(12000, 12000, 300, 300)$ $R = \text{Diag}(50, 50)$
SMC	$K = [4.5, 4.5]^T$ $\lambda = 14$ $\psi = 2.5$
Maple-based NMPC	$Q = \text{Diag}(33000, 33000, 3300, 3300)$ $R_1 = \text{Diag}(50, 50)$ $R_2 = \text{Diag}(20, 20)$

Appendix C

TD3 Networks and Hyperparameters Information

In this section, we are presenting the TD3 tuner information. The neural network layers and hidden nodes are shown in Table C.1. The DRL configuration is depicted in the Table C.2.

Table C.1: Neural Network Parameters

Parameter	Value
Number of hidden layers	2
Hidden layer 1 nodes	400
Hidden layer 2 nodes	300

Table C.2: TD3 Hyperparameters

Parameter	Value
Actor learning rate	10e-5
Critic learning rate	10e-4
τ	0.005
γ	0.99
λ	0.01
Mini-batch size	100
Target noise	0.1