HYBRID FORCE-POSITION CONTROL OF A 4-DOF SCARA MANIPULATOR

COMMANDE HYBRIDE FORCE/POSITION D'UNE MANIPULATEUR SCARA AVEC QUATRE DEGRÉS DE LIBERTÉS

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Jack McDonald, BEng

Captain

In Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in Mechanical Engineering

October, 2022

To my wife, Zaneta and my daughter, Georgia.

Acknowledgements

I would like to thank my supervisor, Dr. Amor Jnifene, for first introducing me to control systems and robotics in the fourth year of my undergraduate degree, and then taking me in as a graduate student, and eventually a member of his faculty. His patience, enthusiasm, mentorship, fairness and expertise have been greatly appreciated.

I would also like to thank the RMC machine shop for the significant assistance with the construction of the physical model for this thesis. In particular, Brendan Freeman, Charles Sadiq and Dave Neumann were very helpful. Their advice, energy, and their patience with the numerous changes and iterations were very helpful.

Finally, I would like to thank my father-in-law, Lubomir Balaz for his critical assistance at multiple points throughout this project. His experience with robot design and mastery of Solidworks came in handy when significant obstacles were encountered.

Abstract

A wide range of robotic tasks ranging from metal grinding to minimally invasive surgery require fine control of both the position and the contact force of the robot's end effector. Force-position control theory for robotic manipulators has been thoroughly researched and and simulated in recent years, with both fundamental and advanced approaches being applied to a broad array of problems with success. Surgical skin cutting has been identified as one area that has not yet seen much work, and a 4 Degrees of Freedom (DOF) Selective Compliance Articulated Robot Arm (SCARA) manipulator is identified as a robot configuration that could be used for this application. A hybrid force-position controller works by running a position feedback loop in parallel with a force feedback loop, with a supervisory controller prioritizing between them. In this thesis, a hybrid force-position controller for a 4-DOF SCARA manipulator was developed. The mathematical model for the kinematics and dynamics of a SCARA robot were determined and verified using MatLab's Simulink and Simscape environments. A PID controller with feedback linearization was designed to control the position of the end effector, while a PI/Feedforward controller with velocity damping was designed to control the end effector's contact force with a simulated environment. Both controllers were combined in a hybrid force-position architecture and simulated with success.

Résumé

Plusieurs tâches robotiques incluant la chirurgie minimalement invasive et le meulage du métal, ont besoin de la régulation précise de la position et la force de contact de l'organe terminal. La théorie du contrôle force-position pour les manipulateurs robotiques a fait l'objet de recherches et de simulations approfondies ces dernières années, avec des approches fondamentales et avancées appliquées à beaucoup de problèmes avec succès. La coupe chirurgicale de la peau a été identifiée comme un domaine qui n'a pas encore vu beaucoup de travail, et un manipulateur de bras robotique articulé à conformité sélective à 4 degrés de liberté est identifié comme une configuration de robot qui pourrait être utilisée pour cette application. La commande hybride force/position des robots manipulateurs fonctionne en exécutant une boucle de rétroaction pour la position en parallèle avec une boucle de rétroaction pour la force, avec un contrôleur de supervision qui commute entre les deux. Cette étude vise à développer un régulateur hybride de la force et position pour un manipulateur SCARA avec quatre degrés de libertés. Le modèle mathématique de la cinématique et de la dynamique d'un robot SCARA a été développé et vérifié à l'aide des environnements Simulink et Simscape de MatLab. Un régulateur PID avec la linéarisation de la rétroaction était conçu pour contrôler la position de l'effecteur d'extrémité, ainsi qu'un régulateur PI/anticipatif avec l'amortissement de la rapidité pour la force de contact a été aussi développé et vérifié. Les deux contrôleurs ont été combinés dans une architecture hybride force-position et simulés avec succès.

Contents

List of Tables

List of Figures

List of Symbols

Symbol	Description	Unit
a_i	link length	m
A_i^{i-1}	homogeneous transform from frame $i-1$ to frame i	
d_i	link offset	m
f_{ext}	vector of external forces/torques acting on the end-effector	N or $\mathrm{N} \cdot \mathrm{m}$
F_N	normal force	N
g	acceleration due to gravity	$\frac{\mathrm{m}}{\mathrm{s}^2}$
I	moment of inertia	$\mathrm{kg} \cdot \mathrm{m}^2$
J	Jacobian	
K	gain matrix	
K_D	derivative gain	
K_{fi}	integral force control gain	
K_{fp}	proportional force control gain	
K_I	integral gain	
K_P	proportional gain	
K_{vd}	velocity damping force control gain	
l_{cix}	x-position of centre of gravity of link i	m
l_{ciy}	y-position of centre of gravity of link i	m
l_{ciz}	z-position of centre of gravity of link i	m
m	mass	kg
$M(\theta)$	mass matrix	

n	number of robot links	
O_i	origin of frame i	$[x_i,y_i,z_i]$
P_E	position of the end effector	$[x_E,y_E,z_E]$
q	joint variable vector	
P_i^{i-1}	position of point i with respect to frame $i-1$	
R_i^{i-1}	rotation matrix from frame $i-1$ to frame i	
$r(t)$	desired trajectory	
S	selection matrix	
t	time	s
α_i	link twist	rad
θ_i	joint angle	rad
τ	generalized joint force/torque vector	N or N $\cdot$ m
τ_C	command force/torque	N or N $\cdot$ m
τ_f	force controller command fore/torque vector	N or N $\cdot$ m
τ_p	position controller command fore/torque vector	N or N $\cdot$ m
ω_i	angular velocity	$\frac{\text{rad}}{\text{s}}$
z_{rs}	z coordinate of the reaction surface	m

1 Introduction

The use of robot manipulators is widespread throughout the modern industrial economy[1], and is rapidly expanding into new areas including health care[2], construction[3] and consumer retail[4]. One of the major drivers of this expansion is the increasing level of sophistication of the controllers behind the robots. The development of force-position controllers is one of the most important areas where this level of sophistication is increasing, and it is the focus of this thesis.

The term *robot* can be applied to a wide variety of things, ranging from bipedal humanoid robots to wheeled and aerial unmanned robotic vehicles. The force-position control concept is one generally used for robot manipulators - that is to say robots that have one end fixed to a static object with a series of articulated links, joints and motors leading towards an end effector. These are often also referred to as robot arms, as they mimic the role that a human's arm plays in the execution of tasks. Moving forward in this thesis, the term *robot* will refer specifically to a *robot manipulator*, and will thus exclude robotic vehicles or any other broader definition of the term.

Almost all robots require the position of their end effectors to be controlled. This is typically achieved with an onboard computer sending actuating signals to motors at each of the joints, and receiving information from sensors that measure and calculate the actual end effector position[5]. As the robot moves through free space, a properly designed position controller can account for the known dynamic properties of the robot, and apply actuating forces and torques at each of the joints that will force the end effector position to converge upon

a desired end effector trajectory[6]. However, when the robot must interact with the external environment in any significant way, the position controller alone may no longer be effective[7]. In many applications, the contact force that the robot exerts on its external environment must also be controlled.

For example, a robot designed to perform metal polishing tasks must control both the position of the end effector and the contact force it exerts on the metal. The position of the end effector must be controlled in order for the interaction between the robot's tool and the metal to take place, while the contact force will determine the rate at which the metal is polished, the rate at which the tool will wear and the heat generated during the operation. Some of the many other applications of force-position control are robotic surgery[63], human-robot interaction[9] and robotic assembly tasks [10][11].

Solutions to the problem of force-position control began with proposals by Salisbury and Craig of an active stiffness control method to control robot contact force using position sensor feedback [12][13]. This was followed by Hogan's development of indirect methods for force-position control using position and velocity feedback, referred to as impedance control or admittance control[7][14][15][16]. At around the same time, both Mason[17] as well as Craig and Raibert [18][19] developed a direct approach to force-position control known as hybrid force-position control. These advances all occurred in the early 1980's, such that by the arrival of the 1990's there existed a solid base of fundamental approaches towards force-position control[20].

These fundamental methods can be split into two categories: indirect methods and direct methods. Indirect methods (impedance and admittance control) are those which use a theoretical model to calculate the contact force from the position and velocity sensor data[21]. Direct methods (hybrid force-position control) use a force/torque sensor or observer to create feedback for an independent force control loop[22]. Additional methods that incorporated elements from both categories were developed including: dynamic hybrid control[23], parallel force-position control[24] and hybrid impedance control [25]. The stability, performance and suitability to different applications of all of these force-position controller types were the subject of considerable

analysis in subsequent research[20][26][27][28][29][30][31][32][33].

Throughout this initial explosion of force-position control theory, a parallel explosion in advanced methods of pure position control for robots was also underway. These new methods included adaptive control[34][35][36], sliding mode control[37][38] and robust control[39][40]. Ensuing papers applied these advanced methods to the force-position control problem with success[41][42][43].

The development of these advanced methods was followed by a second wave of new methods in the 1990's that took advantage of advances in artificial intelligence concepts. Control strategies based on neural networks[44][45][46], fuzzy logic[47][48] and learning algorithms[49] all began to emerge. Similar to the previous wave of advanced controllers, these techniques soon were applied to the problem of force-position control with success[50][51][52][53][54].

The rapid increase in new techniques for position control and force/-position control of robot manipulators started to slow in the mid-2000's, at which point the focus of the research began towards applying the new methods to real-world problems. Problems such as soil excavation[55], robot locomotion[56] and metal grinding[57], to name a few, have been areas where force-position control has been successfully applied. Within the field of health care in particular, there have been many different areas where the use of force-position control has been applied, including limb rehabilitation[58][59], needle/catheter insertion[60][61] and robotic surgery[62].

It is worth noting that the majority of the recent research into advanced force/position control techniques and their application to real-world problems is limited to simulations, with only a few exceptional papers validating their controllers experimentally[21][57][58].

Robotic surgery is a particularly promising field for the use of force/-position controlled robotics. The widespread use of the Da Vinci robot in laproscopic surgery is evidence of this[64], but there also exist many other non-laproscopic surgical tasks that robotics can assist with. One surgical task that has not yet received significant attention from robotics researchers is the cutting of skin. This task is an important part of many different surgical

operations and clearly involves fine control of both the position and applied force of a tool. Some work by Duchemin et al. in the early 2000's successfully applied hybrid force/position control to the related problem of skin harvesting for reconstructive surgery[65][66][67], however there has been little attention to the area since then.

The application of force/position control to the problem of skin cutting in robotic surgery was thus identified as a potential area for development. Using simulations to test developed controllers would be a beneficial contribution in the area, but an even stronger contribution would be to validate developed controllers through physical experimentation. The focus of this thesis was set with this area for contribution in mind.

Figure 1.1: Photo of existing 3-DOF SCARA manipulator

An existing Selective Compliance Articulated Robot Arm (SCARA) manipulator with 3 Degrees of Freedom (DOF), shown in Figure 1.1, was con-

sidered to be a good base to build off of into a robot that could be used in a surgical cutting application. In order for it to be made suitable for surgical skin cutting, it would need a fourth DOF to be added, allowing for control of the orientation of the cutting tool relative to the direction of its travel along the skin. Thus, the development of a force/position controller for a 4-DOF SCARA manipulator was set as the focus of this thesis. It would be developed and tested through simulations, as is done in the majority of the literature, but with a specific focus on eventual experimental validation on a modified version of the existing 3-DOF SCARA manipulator. Efforts to make the required modifications to the robot were also pursued in parallel to the development of the controllers, and are discussed in Appendix A.

It is also worth mentioning that although the state of the art in modern hybrid force/position control is in the application of advanced (adaptive, sliding mode, robust, etc...) controllers and artificial intelligence techniques (neural networks, fuzzy logic, learning algorithms, etc...) to the problem - the fundamental control methods (PID control, feedback linearization, feedforward control, etc...) are often able to match their performance with a much simpler approach. An example of this is in [66], where Dombre et al. use PID control to achieve effective hybrid force/position control in their experimentally validated skin harvesting robot. Thus, the focus of this thesis was the application of fundamental control methods to the problem of hybrid force/position control.

This thesis will be organized into six chapters. This introduction constitutes the first, and a more detailed technical discussion into the background of hybrid force/position control will follow in the second chapter. The third chapter will discuss the development of the mathematical model and simulations of the 4-DOF SCARA manipulator. The fourth chapter will discuss the development of the position controller for the robot, and the fifth chapter will discuss the development of the force/position controller. Finally, the sixth chapter will be a conclusion to the thesis. Two appendices are also included, one containing information on the construction of the physical robot, and one containing all of the thesis's source code.

2 Background

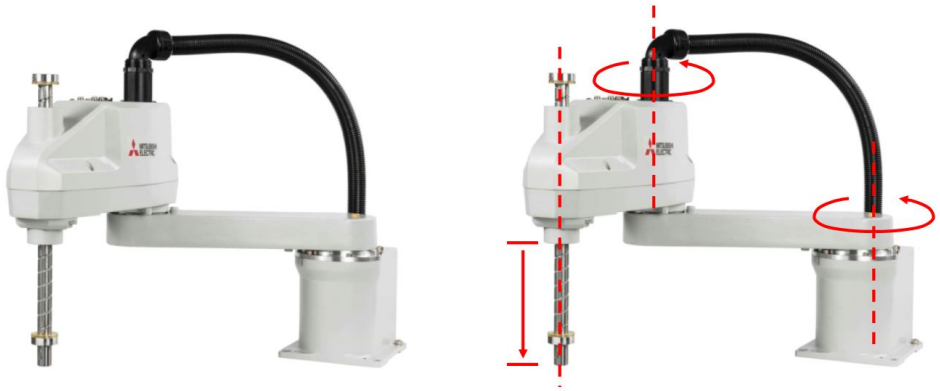

Figure 2.1: Typical SCARA robot shown with and without joint identification[68]

The robot configuration that was investigated for this thesis was the Selective Compliance Articulated Robot Arm (SCARA), which is serial manipulator that has three degrees of freedom (DOF) in its most basic format. Moving from the fixed base of the robot out towards its end effector, the first two joints are revolute, while the last is prismatic. This configuration is useful due to its inherent simplicity and maneuverability. In this thesis, a fourth DOF is added to the configuration to make it more applicable the the task of surgical skin cutting. Many generalized conclusions about the control or modelling of the 4-DOF SCARA robot from this study will also be applicable to any other serial robot manipulator configuration, aside from the implications of

increased/lowered mathematical complexity involved in different robot configurations (i.e. computing power, accumulation of uncertainties, etc...). Figure 2.1 below shows a typical 3-DOF SCARA robot with its three joints identified.

2.1 Position Control

While this study focuses specifically on hybrid force-position control, position control of the end effector forms the basis on which the force-position controller must be built upon, and therefore must be introduced independently.

Position controllers for robot manipulators require a desired trajectory of the end effector (both the position and orientation of the end effector expressed as a time series) as input. The job of the position controller is to take this desired trajectory, and to apply actuating forces and torques at each of the robot's joints in a coordinated manner, such that the position and orientation of the end effector will converge upon the desired trajectory over time. Typically, the robot is equipped with optical encoders at each joint that measure the actual angles of rotation for revolute joints and the joint displacements for prismatic joints. These sensors allow for closed-loop feedback control, where the controller is constantly comparing its actual position/orientation to the desired trajectory, and uses the difference between the two to decide on what signals to send to the system's actuators. A block diagram of a typical position controller is shown below in Figure 2.2.

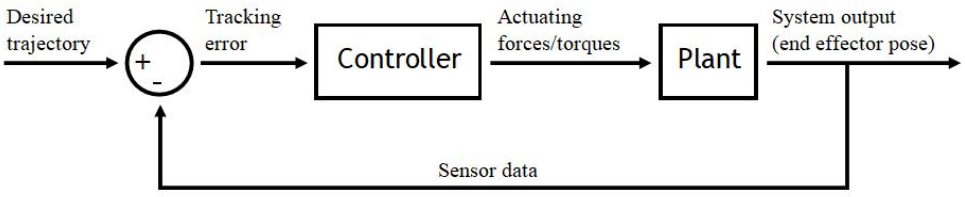

Figure 2.2: Block diagram of a typical position controller

The simplest control strategy for a robot manipulator is one that treats each joint as an independent system, and with a separate controller for each.

This is known as a decentralized control strategy, as there is no centralized controller that coordinates the movements of the joints with each other. These types of controllers are easy to understand, build and troubleshoot, but lack in performance because they fail to deal with coupling between joints. The motion of one motor in, say, the third joint of a robot may create reaction forces that affect the dynamics of the the first joint of the robot. A decentralized controller cannot account for these dynamics, which will ultimately lead to increased tracking error and reduced performance[6].

However, if near-zero tracking error is not required, and the controller designer is willing to accept a reduced performance in favour of simplicity - a decentralized control strategy may be best. If this is the case, a commonly used controller type for each of the individual joints is the Proportional Derivative (PD) controller. This controller uses two fixed gain constants, K_P and K_D, to calculate the command forces/torques to be sent to the motors using the control law shown below:

$$\tau_C = K_P q_e + K_D \dot{q_e} \tag{2.1}$$

Where τ_C represents the command force/torque sent to the joint, and q_e represents the tracking error, that is to say the difference between the measured joint variable (from the joint sensor) and the desired joint variable (from the desired trajectory).

If the control gains are properly selected, an independent joint PD controller can force a robot to successfully track a desired trajectory. However, in the presence on non-linearities and coupling between joints, there will always be a significant tracking error, and even at steady-state there will be an error due to gravity. The size of these errors will be inversely proportional to the size of the control gains, but there are functional limits on the control gains based on the motor saturation limits[69].

A typical solution to the problem of the tracking error is to introduce a third control action into the PD controller called the Integral action. Such

a controller is thus referred to as a PID controller. The control gain K_I is introduced into the previous control as shown below:

$$\tau_C = K_P q_e + K_D \dot{q_e} + K_I \int_0^t q_e dt \tag{2.2}$$

This additional term turns the closed loop system into a third orders system, which allows the controller to reject step input (i.e. constant) disturbances, most notably the effects of gravity[69]. If the nonlinearities of the robot system are known to be bounded, that is to say it is known that they will not exceed a certain maximum, the system can be modelled with a step disturbance of such a size and the PID gains can be set to guarantee convergence. This has a dramatic positive effect on the performance of the system without adding much complexity to the controller. It does not, however, take account of the coupling between the joints.

In order to account for the coupling, a centralized controller must be used. The simplest version of a centralized controller is the state feedback controller. This controller assumes that the robot system is linear (although it is not), and represents it in state space form shown below:

$$\dot{x} = Ax + Bu \tag{2.3}$$

Where x represents the $n \times 1$ state vector of the linearized system, A represents the $n \times n$ state matrix that defines the linear relationships between the states, B represents the $n \times m$ input matrix defining the linear relationships between the system's states and the system's input variables, and u represents the $m \times 1$ input variable vector. Let n represent the order of the system (i.e. the number of states), and m represent to number of input variables. Note that the linearized representation of a robot manipulator in state space will always be of an order (n) that is double the number of joints (one state for each joint's variable, and one for the first derivative thereof), and that the number of input variables (m) will be equal to the number of joints (the command torque sent to each joint).

The control law shown below can be used to stablize the system[70]:

$$u = -Kx + x_d \tag{2.4}$$

Where K represents a $m \times n$ matrix of appropriately selected control gains, and x_d represents an $n \times 1$ vector of desired state variable values (taken from the given desired trajectory). When Equation 2.4 is substituted into Equation 2.3 the following closed loop dynamics result:

$$\dot{x} = Ax + B(-Kx + x_d) \tag{2.5}$$

$$\dot{x} = Ax - BKx + Bx_d \tag{2.6}$$

$$\dot{x} = (A - BK)x + Bx_d \tag{2.7}$$

Note that the closed loop dynamics in Equation 2.7 take the same form as the base state space representation in Equation 2.3, just with the new state space matrix being $A - BK$ and the system input changed to x_d. State space control theory dictates that if the system in Equation 2.3 is *controllable*, that is to say that it satisfies the following inequality[69]:

$$\det \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \neq 0 \tag{2.8}$$

... then there exists a control gain matrix K that, when used in the control law 2.4, can force the poles of the closed-loop system to anywhere in the complex plane. This means that as long as the natural limits of the system's actuators are respected (i.e. the motor saturation limits), the tracking performance of the robot manipulator can be made to be asymptotically stable [71]. Typically, an optimization strategy known as the Linear Quadratic Regulator is used to determine the pole locations in the complex plane that will maximize system tracking performance without over-saturating the motors[71].

Much like the PD controller, a state feedback controller has two states (one for joint position and one for joint velocity) for each of the joints, and

so it will always be able to at least match the performance of the PD control action. In applications where some linear coupling exists between joints (such as in a 4-DOF SCARA manipulator), it will out-perform the decentralized PD controller, as it will be equipped to deal with these dynamics (due to the effect of the non-diagonal elements of the system's A matrix).

The state feedback controller does however fall short in the same main way that the PD controller does, which is in dealing with nonlinearities. The asymptotic stability of the system is only guaranteed insofar as the system can be accurately modelled as purely linear time-invariant (LTI) [70]. Robotic manipulators unfortunately do not fall into this category. In fact, if a system's linear coupling is not significant, a decentralized PID controller will outperform the state feedback controller, as the integral action will be able to eliminate some or all of the steady-state error that comes from the unmodeled nonlinear system elements.

The previously mentioned controllers are all linear controllers, and thus will not properly stabilize the system in the presence of nonlinear dynamics. In order to deal with the nonlinear dynamics, a *feedforward* control scheme can be adopted. In order to demonstrate how such a scheme works, the manipulator dynamic system can be represented in the form shown below:

$$\tau = g(q, \dot{q}, \ddot{q}) + h(q, \dot{q}) \tag{2.9}$$

Where τ represents the forces/torques experienced at each joint, q represents a vector of all of the joint variables, g represents all the of the system's linear dynamics, and h represents all of the system's nonlinear dynamics. The feedforward control approach uses $h(q_d, \dot{q_d})$ as an estimation of the system's nonlinear dynamics and adds them into one of the linear control laws mentioned above that can stabilize the remaining linear dynamics ($g(q, \dot{q}, \ddot{q})$). The control law below shows a PID inner loop controller with a feedforward compensator for the nonlinear dynamics:

$$\tau_C = h(q_d, \dot{q_d}) + K_P q_e + K_D \dot{q_e} + K_I \int_0^t q_e dt \tag{2.10}$$

Where q_d represents a vector of desired joint variables. This controller is efficient because the nonlinear dynamics of the desired trajectory can be pre-calculated as soon as the dynamic model of the system and the desired trajectory are known, reserving online computing power for the simpler linear controller. This method of pre-calculating the expected forces/torques from the nonlinear dynamic terms based on the known desired trajectory is also sometimes referred to as the computed torque method. When the dynamic model is an accurate representation of the system, this controller is very effective, but because modelling error and uncertainty are inevitable, its performance will always be negatively affected by them[72]. Figure 2.3 below shows a block diagram of a feedforward control strategy:

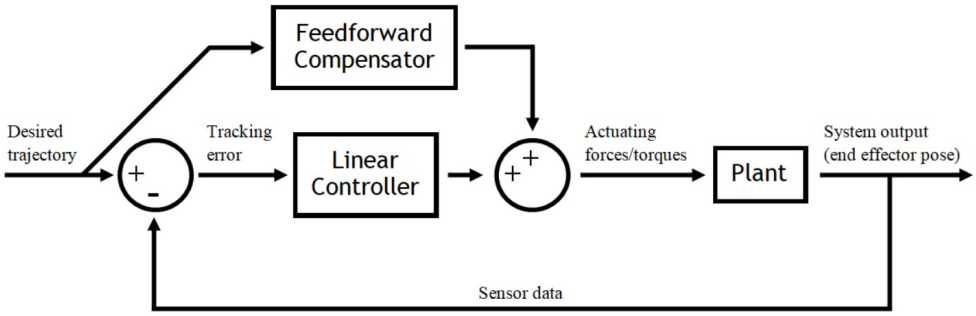

Figure 2.3: Block diagram of a feedforward controller

Because the feedforward element of the controller is using only the desired trajectory in its calculations, whenever the robot's pose differs from the desired pose, the nonlinear elements of the system dynamics will not be completely cancelled, which will act as a disturbance on the linear controller. In order to overcome this issue, the actual measured joint variables are used in Equation 2.11 in lieu of the desired joint variables. This control law is shown below:

$$\tau_C = h(q, \dot{q}) + K_P q_e + K_D \dot{q_e} + K_I \int_0^t q_e dt \tag{2.11}$$

This control strategy is often known as inverse dynamics control[69], and a block diagram of it is shown below in Figure 2.4:

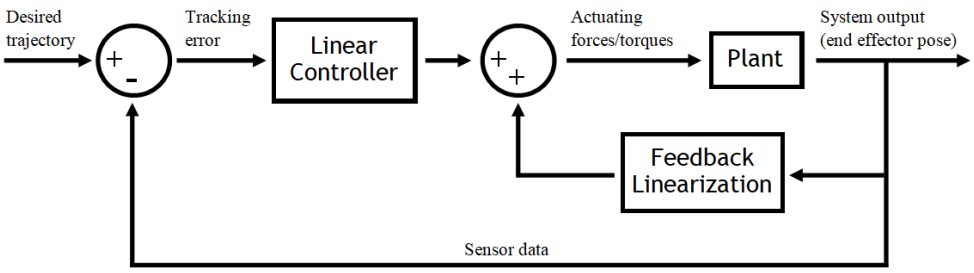

Figure 2.4: Block diagram of an inverse dynamics controller

The inverse dynamics controller will represent an increase in performance when compared to the feedforward controller, but will come at the expense of an increased requirement for online processing power, as the nonlinear dynamics of the robot must be calculated using live sensor data from the robot[72].

Both the feedforward and inverse dynamics controllers use a nonlinear control method known as feedback linearization, where the a nonlinear system is manipulated with an outer feedback loop to attempt to cancel the nonlinearities and convert it into a linear system. This "linearized" system can be controlled using the PD/PID/state space method described earlier, and if the linearization is exact, the system will be asymptotically stable[70].

Beyond these feedback linearization methods, there exist more advanced controllers that can deal with the nonlinearities inherent in robot manipulators. In particular, sliding mode control[37][38], adaptive control[34][35][36] and robust control[39][40] have been proven to be very useful when applied to end-effector position control. These control methods are significantly different than the linear control and feedback linearization methods that have been previously discussed, as their theoretical formulation is based off of the

Lyapunov Stability of the system which is applicable to all dynamic systems, whether linear or nonlinear. The use of such methods is helpful when seeking to achieve strong trajectory-tracking in the presence of uncertainties, or unmodeled dynamics in a system. These requirements are important, but outside the scope of this thesis, so further details of these advanced position control methods will not be discussed.

2.2 Force Control

First, it should be noted that the term "force control" can refer not only to the control of the forces exerted by a robot manipulator on its external environment, but also to the control of torques exerted on its external environment. Much like how "position control" refers to the control of both the position *and* the orientation of the end effector, the same linear-rotational duality exists in the force/torque domain.

Force control of robot manipulators is a significantly less discussed topic than that of position control. Almost every robot manipulator requires end point position control, and while many also require force control, many do not. This means that there are many robots that require only position controllers, and there are very few robots that require only force controllers. There are very few applications where force/torque (but not position/orientation) needs to be controlled, and a robot manipulator is the tool chosen to complete the task. When searching for an example of such a task, one might think of tightening of a bolt. The external torque that must be applied cannot exceed a certain amount so as not to damage the bolt, but it also must reach a certain minimum for the bolt to be properly installed. However, in order for the bolt to be tightened, it must be rotated in place, which means that the orientation of the end-effector of a robot performing this task would in fact also need a position controller.

If a force needs to be controller but the position does not, it is unlikely that a robot manipulator will be selected to perform the task. As such, it is important to understand that force control for robot manipulators always

exists within the context of position control. This relationship is ultimately the focus of this thesis, and will be discussed at length later, but at the moment force control will be separated from position control in order to discuss its theory independently.

Force control refers to controlling the forces and torques that a robot manipulator applies to its external environment. These forces and torques can be represented by a 6×1 combined force/torque vector f_{ext} shown below:

$$f_{ext} = \begin{bmatrix} f_{ext,x} \\ f_{ext,y} \\ f_{ext,z} \\ \tau_{ext,x} \\ \tau_{ext,y} \\ \tau_{ext,z} \end{bmatrix} \tag{2.12}$$

Where $f_{ext,i}$ represents a component of the force that the end effector exerts on the environment in the i direction, and $\tau_{ext,i}$ represents a component of the torque that the end effector exerts on the environment in the i direction. Note that while x, y and z were selected as the coordinate system, any orthogonal, right-hand coordinate frame can be used. Typically, the coordinate frame is selected based on the task that the force control is being used for. For example, if the task assigned to the robot manipulator is to apply a force normal to a surface, the coordinate frame might be set up to have the z axis perpendicular to the surface. Another frame that could be used for the coordinate system of the external force vector would be the frame associated with the end effector (i.e. the tool frame). The force/torque sensor used to generate feedback for the force controller is usually installed at or near to the origin of the the tool frame, and there is convenience associated with having matching coordinate systems for the measured external forces/torques and the desired forces/torques[29].

The goal of a force controller is to take a desired force/torque vector (f_d) as an input, and then come up with a command force/torque to apply at each

of the robot's motors that will cause the actual external forces/torques that the robot applies to its environment (f_{ext}) to converge to f_d. It does this by taking measurements of f_{ext} from a force/torque sensor mounted on the end effector of the robot, and then plugging them into a force control law to calculate a command force/torque (τ_C).

The development of the force control law is highly dependent on the external force model that the designer of the controller adopts, and this model will be highly dependent on the task that the robot is being used for. For position control of the robot, the system model generally will always take the same form, whereas in force control there is no real limit to the number of different models that can be used. For instance, a robot designed to polish metal using a high-speed rotary brush will experience significantly different forces and torques when in contact with its environment than a robot designed to make a surgical incision would.

The modelling of the external forces for this study will be discussed in detail in later on, so for now they will only be treated in general terms. A block diagram of a closed-loop force controlled system is shown below in Figure 2.5:

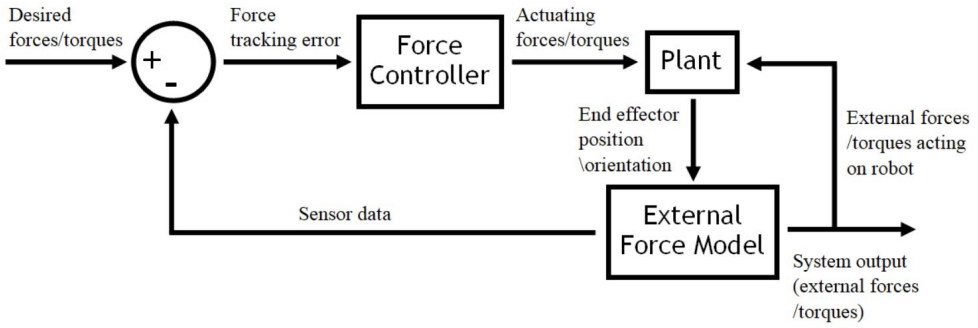

Figure 2.5: Block diagram of a closed-loop force controlled system

Typically the types of force controllers that are used are made up of similar types of actions as those previously discussed in Section 2.1, that is to say linear PID controllers with feedforward or computed torque elements to achieve

feedback linearization. One exception does exist, however, as a result of the nature of the measurements made by the force/torque sensor. Typically, these measurements are extremely noisy, which means that when numerical derivatives are taken, the signal is almost completely useless. Thus, the derivative control action is rarely used, and the linear controllers are left simply with their PI (Proportional and Integral) actions[19].

2.3 Force-Position Control

Controlling both the position and the contact forces exerted by the end effector simultaneously is not possible using the previously discussed position controllers or force controllers alone. In order to achieve proper tracking of both a desired position/orientation trajectory and a desired contact force trajectory, the two objectives must somehow be merged and controlled together.

Force-Position control methods can be grouped into two categories, the first being *indirect methods* and the second being *direct methods*. In the first category, the contact force between the robot's end-effector and the environment is modelled as a function of the end-effector's position (and time derivatives thereof)[73]. The inverse of this function allows the controller to determine an adjusted position/orientation trajectory that, when used as input for a simple position controller, will achieve both the desired position/orientation trajectory and also the desired force/torque output.

The most widely-used of these *indirect methods* is known as *impedance control*, where the relationship between the position and the contact force is modelled as a second order mass-spring-damper system with adjustable parameters. The resistance posed by this mass-spring-damper system, also known as the *mechanical impedance* of the robot's external environment, is simply added into the robot's internal dynamic model, and a position controller can be built to stabilize the system[14][15][16].

While impedance control and other indirect methods of force-position control represent an elegant simplification of the problem, they can only guarantee contact force tracking insofar as the external force model is accurate[28]. In

reality, external forces acting on robot manipulators are often very complex, time-variant, or unknown. If more deliberate force tracking is desired, the *direct methods* of force-position control are more appropriate. In these methods, an explicit closed force feedback loop is used to directly control the output forces exerted by the manipulator. Hybrid force/position control falls firmly into this category[73].

In hybrid force/position control, two feedback loops run in parallel to each other. One controls the position/orientation of the end effector, and the other controls the forces/torques exerted by the end effector on the external environment. Both controllers produce command signals that would achieve stability for their respective goals, and the two signals are both sent to a supervisory controller. The supervisory controller prioritizes the competing goals of position tracking and force tracking, and then sends the system's actuators a single command signal that is built from the two signals it received[19]. The structure is shown below in Figure 2.6.

It is at this point that one of the fundamental problems in force/position control should be mentioned. The problem is that it is the desired forces/-torques and the desired trajectories are not guaranteed to be compatible with each other, that is to say it may be impossible to achieve both targets at once. Imagine a robot tasked with erasing a chalkboard, and that the plane of the chalkboard is normal to the x axis of a global coordinate frame. The position controller will attempt to force the robot to follow a trajectory with a constant value for the x position of the end effector (while moving along the changing paths in the y and z directions), while the force controller will attempt to force the robot to exert a constant force in the x direction. If there is some compliance in the chalkboard, the exertion of a constant force against it will cause it to deform in the x direction, which means that in order for the robot to maintain its desired force, it will need to deviate from its target of holding the end effector position constant in the x direction. In this case, there is no possible way to achieve both targets, because the targets are mutually exclusive.

Indeed, unless the two trajectories are perfectly compatible, it is impossible

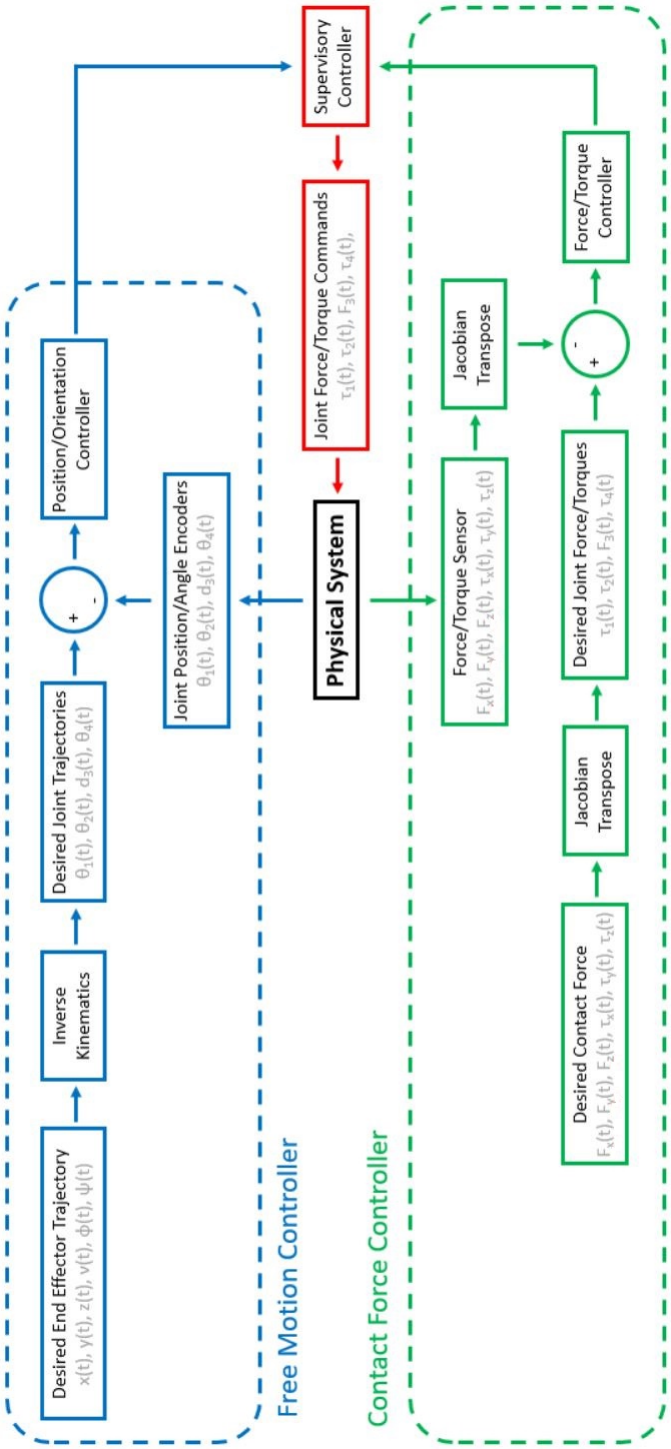

Figure 2.6: Structure of a hybrid force/position controller

to build a controller that can track both force and positions perfectly. If the user of the robot knows enough about the environment that it operates in to be able to come up with desired position and force trajectories that are perfectly compatible, they could simply build their controller using this knowledge and use a single closed-loop feedback system to track one of the trajectories alone. This is a fundamental problem in hybrid force-position control: the desired position and force inputs can often contain conflicting requirements, and the controller needs to be able to prioritize the correct requirements depending on the situation.

The solution to this problem lies in a deeper understanding of the nature of the tasks assigned to the force/position controlled robot manipulator. In [17], Mason explains that if the six degrees of freedom of an end effector (three dimensions of translation and three axes of rotation) are analyzed independently, each one can be assigned as either having *natural constraints* or *artificial constraints*. The natural constraints are those that are imposed by the environment, while the artificial constraints are imposed by the robot's controller. These constraints can either limit the position/orientation of the end effector, or the forces/torques that it can exert on its environment in that direction.

To continue with the example of the robot manipulator tasked with erasing the chalkboard, where the chalkboard is aligned with the $y-z$ plane, the task can be broken down into natural and artificial constraints in the following manner:

Table 2.1: Natural and artificial constraints for a chalkboard-erasing robot

Natural Constraints	**Artificial Constraints**
$v_x = 0$	$f_x = f_{ext,x}$
$f_y = 0$	$v_y = \dot{y}_E$
$f_z = 0$	$v_z = \dot{z}_E$
$\tau_x = 0$	$\omega_x = \omega_{E,x}$
$\omega_y = 0$	$\tau_y = 0$
$\omega_z = 0$	$\tau_z = 0$

Note that v_i represents the velocity of the eraser in the i direction, f_i represents the contact force between the eraser and its environment in the i direction, $f_{ext,i}$ represents the desired contact force between the eraser and its environment, and $\dot{y}_E$ and $\dot{z}_E$ represent the desired velocity of the eraser in the y and z directions, respectively. Also, ω_i represents the rotational velocity of the eraser about the i axis, $\omega_{E,i}$ represents the desired rotational velocity of the eraser about the i axis, and τ_i represents the contact torque about the i axis between the eraser and the chalkboard.

It is important to recognize that the natural constraints of f_y, f_z and τ_x assume a frictionless sliding interaction as the eraser rotates and moves laterally along the chalkboard. This is obviously an oversimplification, and these constraints could be more realistically modelled as functions of the normal force between the chalkboard and the eraser, the velocity of the eraser in the y and z directions, and the rotational velocity of the eraser about the x axis. If this sort of model were to be used, the expressions that represent the respective friction forces and torques would be set equal to f_y, f_z and τ_x and they would remain as natural constraints.

In this breakdown of constraints based on the nature of the task, there are twelve constraints representing the position, orientation, forces and torques each broken down into the three axes of the task frame. For most tasks that involve a robot interacting with stiff, immobile solids, a similar breakdown can be made[17]. With these natural and artificial constraints defined, the problem of conflicting requirements can be resolved.

The solution to the problem is to ask the controller to only impose the artificial constraints on to the robot, while allowing the environment to impose the natural constraints on the robot. As long as there no more than six artificial constraints and they are all mutually orthogonal (i.e. no linear artificial constraints in the same direction as each other, and no rotational artificial constraints about the same axis as each other), there will be no conflicting requirements that the controller needs to prioritize between.

The supervisory controller in the hybrid force/position control scheme is

responsible for taking the two independent sets of command signals sent by the position controller and the force controller, and then deciding upon a single set of command signals to be sent to the robot's actuators. It does this by breaking the task down into its natural and artificial constraints, and determining whether the force command or the position command is the appropriate signal to be used for each of the artificial constraints[19]. Continuing on with the chalkboard-erasing robot example, the supervisory controller would send the command signals from the force/torque control loop for the linear x direction and the rotational y and z directions, and it would send the command signals from the position/orientation control loop for the linear y and z directions and the rotational x direction.

Decomposing and rebuilding these command signals from global coordinate frames, to task frames, and then mapping them into signals to be sent to the robot's actuators, is a complex tasks that requires significant processing power and knowledge of the robot model. However, as these transformations rely heavily on the robot model, the environmental force model, and the selection of the task frame, their explanation will be discussed later in Chapter 5.

Hybrid force/position control will be the focus of the remainder of this thesis. The next chapter will deal with the development of the mathematical model of the robot that was investigated

3 Model Development

The first step in the control system design was the development of a theoretical model of the robot. This process included the creating a kinematic model of the robot using the Denavit-Hartenburg (D-H) Convention, deriving the inverse kinematics and robot workspace, using differential kinematics to determine the singular positions, creating the dynamic model using the Lagrange method, and finally recreating the entire model in MatLab/Simulink.

3.1 Kinematic Model

The development of the kinematic model of the robot was the critical foundation for the remainder of the project. Any mistakes made here might not have been immediately apparent, but would have had dramatic effects on later results. It was also important that the frames were assigned and nomenclature selected in an intuitive manner to allow for more straightforward analysis of results and troubleshooting of problems.

The kinematic modelling for the robot was performed without any specific physical parameters of any of the links or motors. All parameters were left as variables. The starting point of the development was simply the 4-DOF SCARA manipulator configuration. This approach made it so that design choices on the physical parameters of the robot could be made with more knowledge of how they would effect its performance.

3.1.1 Frame Assignment

The first step in the development of the kinematic model was the assignment of frames of reference to each link using the Denavit-Hartenberg (D-H) Convention[74]. In the the D-H Convention, each link is numbered starting from the fixed base of the arm, which is called link 0. The moving link attached to link 0 is called link 1, and so on all the way until the end effector is reached. Likewise, the joints of the robot are also numbered such that joint i forms the connection between link $i-1$ and link i.

The z axis of each frame i (i.e. z_i) is set parallel to the joint axis of the joint $i+1$. Next, the x_i axis is set as the common normal between the z_{i-1} and the z_i axes. When the z_{i-1} and z_i axes are parallel (as is commonly the case), the x_i axis can be set arbitrarily to wherever is most convenient for the frame assigner. With the z_i and x_i axes established, the origin of the frame, O_i, is set as the intersection of the two, and the y_i axis is set as direction normal to the x_i-z_i plane that originates from O_i in the direction compliant with the right-hand rule for frames of reference.

The origin of frame i is set at the intersection of the common normal of the z_i and z_{i+1} axes (i.e. a_{i+1}) and the z_i axis. T Note that if the axes z_i and z_{i+1} are parallel (as is commonly the case), there are an infinite number of common normals between them, and thus an infinite number of locations for the origin of frame i. In this case, the origin may be set wherever is most convenient.

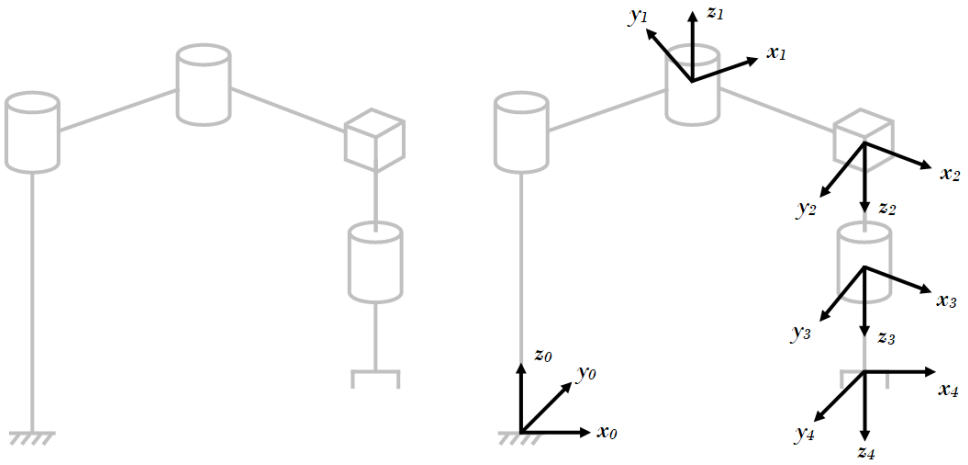

Figure 3.1: Schematic of SCARA manipulator with frame assignments

A special case exists for the assignment of the base frame of the robot (i.e. frame 0). As with all other frames, the z_0 axis must align with the axis of joint 1. However, because there exists no frame before it, no common normal can be used to set the x_0 axis. Thus, the x_0 axis (and furthermore the origin and y_0 axis) are all set arbitrarily. In this case, they should be set to align the origin with some known reference point in the environment wherein the robot operates. The base frame is the only fixed frame, so it will be used to define the motion of the robot's end-effector in its operating environment. Thus, it is best to set the origin at some known reference point in the environment that coincides with the z_0 axis. A good choice for this is the point where the fixed arm of the robot is attached to its operating environment (i.e. the ground, or a platform on which it is operating).

A second special case exists for the assignment of frame n, where n represents the number of joints in the robot. Because there is no joint $n+1$, the z_n axis must be set arbitrarily. Typically, the z_n axis is chosen to be set to be aligned with z_{n-1}, or to be aligned with the approach vector of the end effector. Once z_n is set, the same rules apply for setting the location of O_n, and the x_n and y_n axes.

Figure 3.1 shows a schematic of the SCARA manipulator and the frames that were assigned to each joint using the D-H Convention.

3.1.2 Link Connection Definitions

Once frames are assigned to each joint, the D-H Convention uses four variables to describe the transformations between successive frames (i.e. the geometric characteristics of the robot links). These four variables are called link length, link offset, link twist and joint angles (represented by a_i, d_i, α_i and θ_i, respectively).

The link length, a_i, (as previously described) represents the length of the common normal between the z_{i-1} and z_i axes. The link twist, α_i, represents the angle of rotation about the x_i axis that is required to bring the z_{i-1} axis parallel to the z_i axis. Once the transformations defined by a_i and α_i are done, the link offset, d_i, represents the distance along the z_i axis that O_{i-1} must be translated in order for it to be coincident with O_i. Finally, once these three transformations are done, the joint angle, θ_i, represents the rotation about the z_i axis required to align the x_{i-1} and x_i axes.

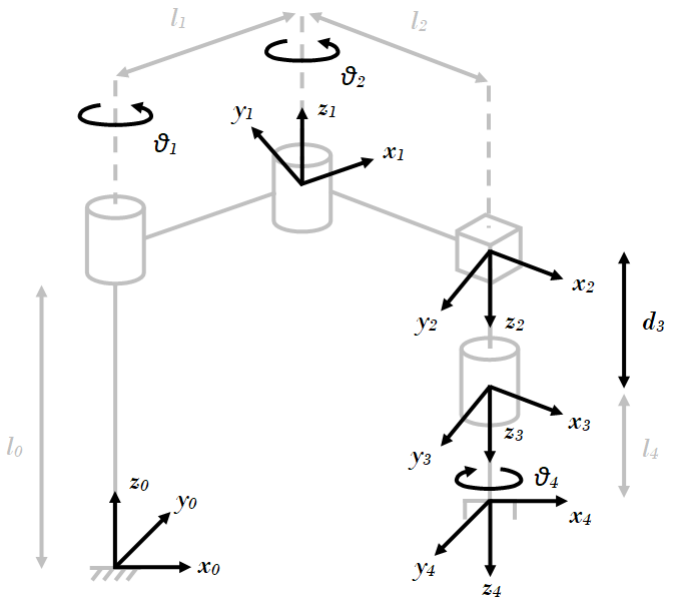

Figure 3.2: Schematic of SCARA manipulator showing link connection variables

Figure 3.2 shows an updated schematic of the 4-DOF SCARA manipulator with the important link connection variables labelled. These variables were collected and then listed in Table 3.1. It is important to note that the variables θ_1, θ_2, d_3 and θ_4 could vary over time, as they were the variables associated with the movement of each joint in the RRPR configuration. The variables l_0, l_1, l_2 and l_4 represent the fixed lengths of each link in the robot. They were left as variables so that they could be parameterized during any future design process.

Table 3.1: D-H Table for 4-DOF SCARA manipulator

Link	$\boldsymbol{a_i}$	$\boldsymbol{d_i}$	$\boldsymbol{\alpha_i}$	$\boldsymbol{\theta_i}$
1	l_1	l_0	0	θ_1
2	l_2	0	π	θ_2
3	0	d_3	0	0
4	0	l_4	0	θ_4

3.1.3 Forward Kinematics

With the link connection variables defined, the D-H Convention provides a straightforward path to kinematic equations by using homogeneous transformation matrices. These 4×4 matrices are a mechanism that can be used to convert a point or a vector from one frame into another, both by rotating it and translating it. The homogeneous transformation that converts a vector from i to frame $i-1$ is represented by A_i^{i-1}, and can be calculated using the following formula:

$$A_i^{i-1} = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Note that the notations s and c are shorthand for sine and cosine, s_i and c_i are shorthand for $\sin\theta_i$ and $\cos\theta_i$, s_{ij} and c_{ij} are shorthand for $\sin(\theta_i + \theta_j)$ and $\cos(\theta_i + \theta_j)$, and finally that $s_{i\bar{j}}$ and $c_{i\bar{j}}$ are shorthand for $\sin(\theta_i - \theta_j)$ and $\cos(\theta_i - \theta_j)$. Also note that moving forward, the vector q shall represent a vector of the four joint variables, that is to say $q = \begin{bmatrix} \theta_1 & \theta_2 & d_3 & \theta_4 \end{bmatrix}^{\mathrm{T}}$.

Homogeneous transforms can be concatenated by post-multiplying them, thus yielding the overall homogeneous transform from the end effector to the base frame. Thus, the overall homogeneous transform for the 4-DOF SCARA robot was calculated as follows:

$$A_4^0(q) = A_1^0(q)\, A_2^1(q)\, A_3^2(q)\, A_4^3(q) \tag{3.2}$$

$$A_4^0(q) = \begin{bmatrix} c_1 & -s_1 & 0 & l_1 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & l_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2 & s_2 & 0 & l_2 \\ s_2 & -c_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$A_4^0(q) = \begin{bmatrix} c_{12\bar{4}} & s_{12\bar{4}} & 0 & l_1c_1 + l_2c_{12} \\ s_{12\bar{4}} & -c_{12\bar{4}} & 0 & l_1s_1 + l_2s_{12} \\ 0 & 0 & -1 & l_0 - l_4 - d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

This homogeneous transform contains all the information needed to complete the kinematic model. Rows 1 through 3 of columns 1 through 3 make up the rotation matrix (i.e. a description of the orientation of the end effector with respect to the base frame), and rows 1 through 3 of column 4 make up the position vector of the end effector in the base frame. The rotation matrix will be represented by R_4^0 and the position vector will be represented by P_4^0:

$$R_4^0(q) = \begin{bmatrix} c_{12\bar{4}} & s_{12\bar{4}} & 0 \\ s_{12\bar{4}} & -c_{12\bar{4}} & 0 \\ 0 & 0 & -1 \end{bmatrix} \qquad P_4^0(q) = \begin{bmatrix} l_1c_1 + l_2c_{12} \\ l_1s_1 + l_2s_{12} \\ l_0 - l_4 - d_3 \end{bmatrix} \tag{3.5}$$

These results made intuitive sense for a number of reasons. First, the z component of P_4^0 is $l_0 - l_4 - d_3$. Just by inspecting Figure 3.2 it can be seen that this should be the case. The x and y components of P_4^0 are functions of only θ_1 and θ_2, which also makes intuitive sense. Additionally, θ_4 has no effect on the position of the end effector, and d_3 has no effect on the orientation of it. These intuitive observations confirmed the validity of the D-H convention methods and that they were effectively employed in this application.

With these results, the position and orientation of the end effector could be determined if the state of each joint variable was known. For control of the

end effector, however, it was important to be able to determine the required set of joint variables that would result in a given end effector position and orientation. This is called inverse kinematics, and was the next step in the full development of the model.

3.1.4 Inverse Kinematics

Inverse kinematics are developed by taking the forward kinematic equations, and simply re-configuring the equations to solve for the joint variables, instead of the end effector position and orientation. This process is complicated by the fact that multiple solutions often exist.

Before going further, it is necessary to establish the convention that the position and orientation of the end effector will be described with. The position of the end effector will be relatively simple. The vector P_E will represent the location of the end effector in the base frame, and will be made up of the three components x_E, y_E and z_E. The orientation of the end effector is more complex to describe.

There exist many ways to describe transformations in orientations, but the one that will be used here will be roll-pitch-yaw. In this method, three successive elementary rotations are made. The first is the "roll" rotation, of an angle ψ about the x_0 axis. The next is the "pitch" rotation, of an angle ν about the y_0 axis, and the final is the "yaw" rotation of an angle ϕ about the z_0 axis. Because the axes of rotation for all three are all part of a fixed frame, the determination of the overall rotation matrix can be found using pre-multiplication:

$$R_E = R_z(\phi)\, R_y(\nu)\, R_x(\psi) \tag{3.6}$$

$$R_E = \begin{bmatrix} c\phi & -s\phi & 0 \\ s\phi & c\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\nu & 0 & s\nu \\ 0 & 1 & 0 \\ -s\nu & 0 & c\nu \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\psi & -s\psi \\ 0 & s\psi & c\psi \end{bmatrix} \tag{3.7}$$

$$R_E = \begin{bmatrix} c\phi c\nu & c\phi s\nu s\psi - s\phi c\psi & c\phi s\nu c\psi + s\phi s\psi \\ s\phi c\nu & s\phi s\nu s\psi + c\phi c\psi & s\phi s\nu c\psi - c\phi s\psi \\ -s\nu & c\nu s\psi & c\nu c\psi \end{bmatrix} \tag{3.8}$$

The goal of the inverse kinematics will be to produce a solution for each joint variable when given a set of ZYX Euler Angles, along with a location of the end effector.

The first and simplest component to solve for in this case was for the third joint, d_3, as it was only present in one part of the forward kinematic solution, z_E:

$$z_E = l_0 - l_4 - d_3 \tag{3.9}$$

$$d_3 = l_0 - l_4 - z_E \tag{3.10}$$

Next, the ZYX Euler Angle rotation matrix from Equation 3.8 was compared with R_4^0 from 3.5 to come up with obsevations regarding ν and ψ:

$$R_4^0(q) = \begin{bmatrix} c_{12\bar{4}} & s_{12\bar{4}} & 0 \\ s_{12\bar{4}} & -c_{12\bar{4}} & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad R_E = \begin{bmatrix} c\phi c\nu & c\phi s\nu s\psi - s\phi c\psi & c\phi s\nu c\psi + s\phi s\psi \\ s\phi c\nu & s\phi s\nu s\psi + c\phi c\psi & s\phi s\nu c\psi - c\phi s\psi \\ -s\nu & c\nu s\psi & c\nu c\psi \end{bmatrix} \tag{3.11}$$

Looking at r_{31} in both matrices, it could be seen that $s\nu = 0$, which meant that ν would always be either 0 or π. Substituting this result into r_{32} and r_{33} showed that ψ would be π when ν was 0, or it would be 0 when ν was π. In

the ZYX Euler Angle framework, a rotation of π about either the fixed x axis or the fixed y axis produced the same result, so there will exist no solution that differentiates between the two possibilities. Therefore, it was arbitrarily chosen that ν would be set to 0 and that ψ would be set to π. With this set, comparing r_{11}, r_{21}, r_{12} and r_{22} between the two rotation matrices showed that ϕ would always be equal to $\theta_1 + \theta_2 - \theta_4$.

These results made sense, as there was no way that any of the joints could have been manipulated to change the orientation of the end effector about the x_0 or y_0 axes, and the orientation of the end effector about the z_0 axis was simply a linear combination of the joint angles 1, 2 and 4.

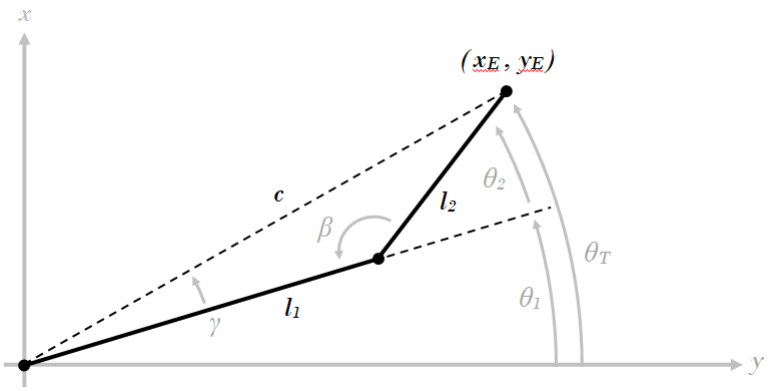

Figure 3.3: Two dimensional representation of links 1 and 2

In order to solve for θ_1 and θ_2, some new geometric intermediate variables were required. Figure 3.3 shows them all and what they represent. The first solution that will be shown is for θ_2, using cosine law:

$$c^2 = a^2 + b^2 - 2ab\cos\beta \tag{3.12}$$

$$\left(\sqrt{{x_E}^2 + {y_E}^2}\right)^2 = {l_1}^2 + {l_2}^2 - 2l_1 l_2 \cos\left(180 - \theta_2\right) \tag{3.13}$$

$${x_E}^2 + {y_E}^2 = {l_1}^2 + {l_2}^2 + 2l_1 l_2 \cos\theta_2 \tag{3.14}$$

$$\theta_2 = \pm\arccos\left(\frac{{x_E}^2 + {y_E}^2 - {l_1}^2 - {l_2}^2}{2l_1 l_2}\right) \tag{3.15}$$

It is important to note that there are two possible solutions for θ_2: one where it is positive and one where it is negative. These two cases correspond to the "elbow-up" and "elbow-down" configurations of the second joint. This kinematic ambiguity had major implications in the path-planning and control of the robot.

In order to calculate θ_1, sine law was used:

$$\frac{\sin\gamma}{l_2} = \frac{\sin\beta}{c} \tag{3.16}$$

$$\frac{\sin\left(\theta_T - \theta_1\right)}{l_2} = \frac{\sin\left(180 - \theta_2\right)}{\sqrt{{x_E}^2 + {y_E}^2}} \tag{3.17}$$

$$\sin\left(\arctan2(y_E, x_E) - \theta_1\right) = l_2 \frac{\sin\theta_2}{\sqrt{{x_E}^2 + {y_E}^2}} \tag{3.18}$$

$$\theta_1 = \arctan2(y_E, x_E) - \arcsin\left(\frac{l_2 \sin\theta_2}{\sqrt{{x_E}^2 + {y_E}^2}}\right) \tag{3.19}$$

At this point the two calculated values of θ_2 from Equation 3.15 could be substituted into Equation 3.19 to determine the two corresponding values for θ_1. Thus, the inverse kinematic equations of the robot were:

$$\theta_1 = \arctan2(y_E, x_E) - \arcsin\left(\frac{l_2 \sin\theta_2}{\sqrt{{x_E}^2 + {y_E}^2}}\right) \tag{3.20}$$

$$\theta_2 = \pm\arccos\left(\frac{{x_E}^2 + {y_E}^2 - {l_1}^2 - {l_2}^2}{2l_1 l_2}\right) \tag{3.21}$$

$$d_3 = l_0 - l_4 - z_E \tag{3.22}$$

$$\theta_4 = \theta_1 - \theta_2 - \phi \tag{3.23}$$

3.1.5 Workspace

With the forward and inverse kinematic models fully developed, the shape of the robot's workspace could be determined. Given the physical parameters of the robot (link lengths, joint limits), the workspace could be plotted and visualized.

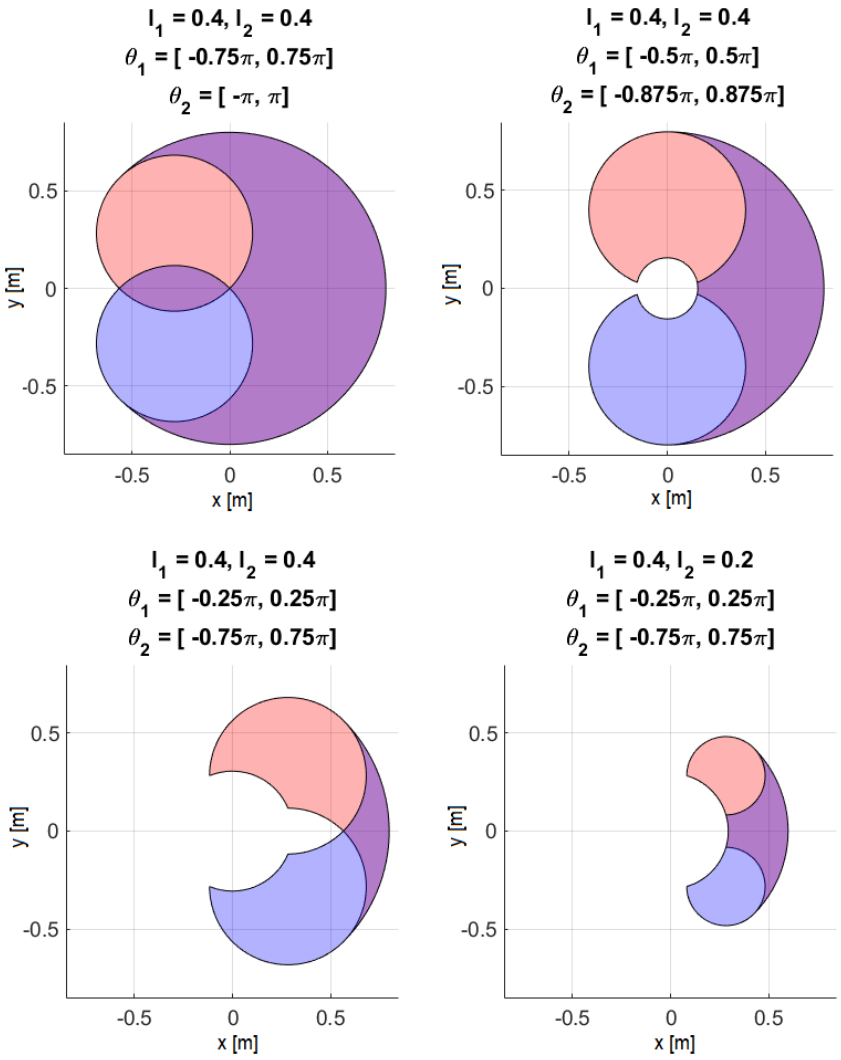

Figure 3.4: 2D representation of robot workspace with varied physical parameters

As the forward kinematic model of the robot found in Section 3.1.3 showed, the fourth joint contributed only to the orientation of the end effector, and not to its position. Additionally, Equation 3.9 showed that the third joint (and the only the third joint) contributed only to the z_E position of the end effector. This meant that the determination of the workspace was largely a two-variable problem, governed by the limits of joints 1 and 2. Using the same logic, the only two link connection variables that could change the shape of the workspace were l_1 and l_2.

The workspace was plotted using a number of arbitrary values for l_1, l_2 and for the upper and lower limits of θ_1 and θ_2. These plots are shown in Figure 3.4. The red areas represent the regions of the workspace that were accessible only in the "elbow-up" configuration, while the blue areas represent those only accessible in the "elbow-down" configuration. The purple areas represent the regions that were accessible with both configurations.

These images of course only show a two dimensional cross-section of the workspace over the x-y plane. Figure 3.5 below shows a full three dimensional representation of the workspace.

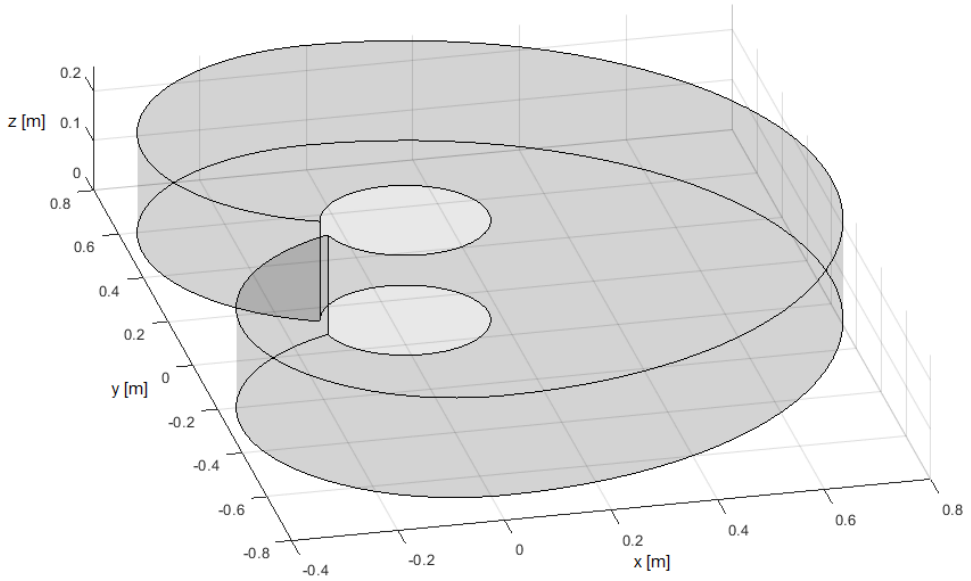

Figure 3.5: 3D representation of robot workspace

With the workspace, the forward kinematics and the inverse kinematics all determined, the kinematic model of the robot could be considered complete.

3.2 Differential Kinematics

The next step in the development of the robot model was to determine the differential kinematic equations that represented the movement of its joints and links through space and time. By taking the kinematic equations for each joint from Equation 3.4 and differentiating them once with respect to time, the relationship between each of the joint velocities and the end effector velocity could be ascertained. This relationship between joint and end effector velocities is represented by a matrix called the Jacobian, which is used in many subsequent levels of robot model development.

3.2.1 Computing the Jacobian

To find this Jacobian, the generalized velocity of the end effector first had to be calculated. The generalized velocity vector shall be represented by v, and shall be a combination of the linear velocity ($\dot{P}$) and angular velocity (ω):

$$v_4^0 = \begin{bmatrix} \dot{P}_4^0 \\ \omega_4^0 \end{bmatrix} \tag{3.24}$$

The linear and angular velocities of the end effector were calculated iteratively, by calculating the linear and angular velocities of each successive joint. The following generalized equations were used to calculate each frame's velocities once the velocities of the previous frame were known:

$$\dot{P}_i^0 = \dot{P}_{i-1}^0 + \dot{P}_{i-1,i}^0 + \left(\omega_{i-1}^0 \times P_{i-1,i}^0\right) \tag{3.25}$$

$$\omega_i^0 = \omega_{i-1}^0 + \omega_{i-1,i}^0 \tag{3.26}$$

In these equations, the subscripts and superscripts can be explained as follows. Using the example of $\dot{P}_{2,3}^0$, the superscript 0 represents the frame of reference that the variable (in this case the linear velocity) was being compared to. The subscript 2,3 means that the velocity that was being measured was the linear velocity of the origin of frame 3 with respect to the origin of frame 2. If the subscript is a single number, then it simply means that the variable in question is being measured in absolute terms, not as one frame with respect to another.

These two equations could be expressed more clearly when it is known whether the joint i is a prismatic or revolute joint. For prismatic joints, it is known that $\omega_{i-1,i}^0 = 0$ and that $\dot{P}_{i-1,i}^0 = \dot{d}_i z_{i-1}^0$, where z_{i-1}^0 represents the third (i.e. right-most) column of the rotation matrix R_{i-1}^0. Thus, Equations 3.25 and 3.26 could be simplified to the following in the case of prismatic joints:

$$\dot{P}_i^0 = \dot{P}_{i-1}^0 + \dot{d}_i z_{i-1}^0 + \left(\omega_{i-1}^0 \times P_{i-1,i}^0\right) \tag{3.27}$$

$$\omega_i^0 = \omega_{i-1}^0 \tag{3.28}$$

For revolute joints, it is known that $\omega_{i-1,i}^0 = \dot{\theta}_i z_{i-1}^0$ and that $\dot{P}_{i-1,i}^0 = \omega_{i-1,i}^0 \times P_{i-1,i}^0$. Substituting this into Equation 3.25 yielded the following:

$$\dot{P}_i^0 = \dot{P}_{i-1}^0 + \left(\omega_{i-1,i}^0 \times P_{i-1,i}^0\right) + \left(\omega_{i-1}^0 \times P_{i-1,i}^0\right) \tag{3.29}$$

$$\dot{P}_i^0 = \dot{P}_{i-1}^0 + \left(\omega_{i-1,i}^0 + \omega_{i-1}^0\right) \times P_{i-1,i}^0 \tag{3.30}$$

$$\dot{P}_i^0 = \dot{P}_{i-1}^0 + \omega_i^0 \times P_{i-1,i}^0 \tag{3.31}$$

And substituting into Equation 3.26 yielded the following:

$$\omega_i^0 = \omega_{i-1}^0 + \dot{\theta}_i z_{i-1}^0 \tag{3.32}$$

Using Equations 3.27, 3.28, 3.31 and 3.32, one can calculate the generalized velocity vector for each frame if the generalized velocity vector of the previous frame is already known. By starting at the base frame (whose generalized velocity vector was 0 in all directions), one could calculate the vector for each successive frame until finally, the end effector was reached. Each successive joint made the equations longer and more complex, which made sense based on the increasing effects of coupling between the joints moving away from the base frame.

Using the MatLab script shown in Appendix A, the end effector generalized velocity vector was found to be:

$$v_4^0 = \begin{bmatrix} \dot{P}_4^0 \\ \omega_4^0 \end{bmatrix} = \begin{bmatrix} -l_2\left(\dot{\theta}_1 + \dot{\theta}_2\right)\sin\left(\theta_1 + \theta_2\right) - l_1\dot{\theta}_1\sin\theta_1 \\ l_2\left(\dot{\theta}_1 + \dot{\theta}_2\right)\cos\left(\theta_1 + \theta_2\right) + l_1\dot{\theta}_1\cos\theta_1 \\ -\dot{d}_3 \\ 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 - \dot{\theta}_4 \end{bmatrix} \tag{3.33}$$

These results confirmed the validity of the MatLab script, as $\dot{P}_4^0$ is equal to the first derivative of P_4^0 (from Equation 3.5) with respect to time. Additionally, in Equation 3.5, it was clear that the rotation matrix was just an elementary rotation of $\theta_1+\theta_2-\theta_4$ about the base frame's z-axis. These results confirmed that ω_4^0 is also equal to the first derivative with respect to time of such a rotation.

The Jacobian of the robot, represented by a $6 \times n$ matrix $J(q)$, can primarily defined as the matrix that maps the joint velocities, $\dot{q}$, to the robot's end effector generalized velocity vector v:

$$v = J(q)\dot{q} \tag{3.34}$$

Looking at the results in Equation 3.33, the Jacobian for the 4-DOF SCARA manipulator was found to be:

$$J(q) = \begin{bmatrix} -l_2\sin\left(\theta_1 + \theta_2\right) - l_1\sin\theta_1 & -l_2\sin\left(\theta_1 + \theta_2\right) & 0 & 0 \\ l_2\cos\left(\theta_1 + \theta_2\right) + l_1\cos\theta_1 & l_2\cos\left(\theta_1 + \theta_2\right) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix} \tag{3.35}$$

Note that the Jacobian can be broken down into the translational Jacobian J_P and the rotational Jacobian J_O:

$$J = \begin{bmatrix} J_P \\ J_O \end{bmatrix} \tag{3.36}$$

$$J_P = \begin{bmatrix} -l_2 s_{12} - l_1 s_1 & -l_2 s_{12} & 0 & 0 \\ l_2 c_{12} + l_1 c_1 & l_2 c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad J_O = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix} \tag{3.37}$$

3.2.2 Kinematic Singularities

By analyzing the Jacobian from Equation 3.37 it was possible to confirm some things that were already known about the robot's kinematics, and also possible to learn some new information.

The first observation of importance was that the fourth and fifth rows of the matrix were all zeroes. Looking back at Equation 3.33, this meant that there was no way to change the angular velocities of the end effector about the base frame's x or y axes of rotation, and thus no way to change the angular orientation of the end effector about those same axes. This was a manifestation of the fact that the SCARA arm only had four degrees of freedom, and so was not a new piece of information.

The third and fourth columns of the Jacobian clearly showed that the linear velocity of the end effector in the z-direction and the angular velocity of the end effector about the z axis of rotation could both be controlled at all configurations of the robot directly by the velocities of the third and fourth joints, respectively. This confirmed the existing understanding of the robot's kinematic model.

The more interesting conclusion that could be derived from the Jacobian was in consideration of the first two columns. These columns represented the effects that changes in the velocities of joints 1 and 2 had on the generalized velocity of the end effector. It could be seen that the third, fourth and fifth rows of these two columns were zeroes, and the sixth row was just one. This meant that the first two joint velocities had no effect on the z-position, the x-orientation nor the y-orientation of the end effector. It also meant that

the effect it had on the z-orientation of the end effector was simply a linear combination of the effect that the fourth joint had. However, the first two rows of these first two columns show expressions that represent the only means that the robot had of controlling the x or y positions of the end effector (because the first two rows of the third and fourth columns were all zeroes). These four elements are shown in a smaller matrix below, which will be referred to as J_{12}:

$$J_{12} = \begin{bmatrix} -l_2 s_{12} - l_1 s_1 & -l_2 s_{12} \\ l_2 c_{12} + l_1 c_1 & l_2 c_{12} \end{bmatrix} \tag{3.38}$$

Since J_{12} represented the whole connection between the robot's joint velocities and the end effector's linear velocity in two directions (x and y), it can be said that if J_{12} ever became rank-deficient (i.e. the columns became linearly-dependent), the robot end effector would be in a position of reduced mobility of the end effector (in the x-y plane). To be more specific, this would mean that a finite velocity of the end effector in a certain direction would require an infinite joint velocity for at least one of the joints. These configurations of reduced mobility are referred to as "kinematic singularities" or "singular positions". To find them, the determinant of J_{12} was set to zero:

$$\det(J_{12}) = 0 = (-l_2 s_{12} - l_1 s_1)(l_2 c_{12}) - (-l_2 s_{12})(l_2 c_{12} + l_1 c_1) \tag{3.39}$$

$$0 = -{l_2}^2 s_{12} c_{12} - l_1 l_2 s_1 c_{12} + {l_2}^2 s_{12} c_{12} + l_1 l_2 c_1 s_{12} \tag{3.40}$$

$$0 = l_1 l_2 \big(c_1 s_{12} - s_1 c_{12}\big) \tag{3.41}$$

$$0 = l_1 l_2 \big(\sin(\theta_1 + \theta_2 - \theta_1)\big) \tag{3.42}$$

$$0 = \sin(\theta_2) \tag{3.43}$$

$$\theta_2 = 0 \quad \text{or} \quad \theta_2 = \pi \tag{3.44}$$

This meant that if the second joint was set to either 0 or π radians, the robot would be in a singular position. This made sense, because if the second joint was at 0, the first two links would be fully extended, meaning that

the robot would no longer be able to move in the direction parallel to the first two links. If the second joint was at π radians that would mean that the first two joints would be parallel again, and the robot would be at the internal boundary of its workspace. These kinematic singularities are known as boundary singularities.

3.3 Dynamic Model

The final step in the development of the mathematical model of the robot was to incorporate information on the masses and moments of inertia of each link. Doing so would progress the model from a kinematic model to a dynamic model. This was extremely important, because it would allow for an analysis of the forces acting on the robot, and in turn to analyze the forces (or torques) required by each joint's motor in order to control the robot.

The Lagrange Method was used to generate the dynamic model of the robot. In this method, the potential and kinetic energies of each link are considered and modelled mathematically. The resulting expressions can be differentiated with respect to time and to the the joint variables, and combined to create a general set of dynamic equations.

First, expressions had to be built for the positions, rotational velocities ($\omega^0_{CG_i}$) and translational velocities ($\dot{P}^0_{CG_i}$) of the centres of mass of each link i, expressed in frame 0. Equations 3.25 and 3.26 were used to come up with the following expressions:

$$\dot{P}^0_{CG_1} = \begin{bmatrix} -\dot{\theta}_1\big(l_1 s_1) + l_{c1x} s_1) + l_{c1y} c_1)\big) \\ \dot{\theta}_1\big(l_1 c_1) + l_{c1x} c_1) - l_{c1y} s_1)\big) \\ 0 \end{bmatrix}$$

$$\dot{P}^0_{CG_2} = \begin{bmatrix} \big(\dot{\theta}_1 + \dot{\theta}_2\big)\big(l_{c2y} c_{12} - l_{c2x} s_{12}\big) - \dot{\theta}_1\big(l_1 s_1 + l_2 s_{12}\big) - \dot{\theta}_2 l_2 s_{12} \\ \big(\dot{\theta}_1 + \dot{\theta}_2\big)\big(l_{c2x} c_{12} + l_{c2y} s_{12}\big) + \dot{\theta}_1\big(l_1 c_1 + l_2 c_{12}\big) + \dot{\theta}_2 l_2 c_{12} \\ 0 \end{bmatrix}$$

$$\dot{P}^0_{CG_3} = \begin{bmatrix} \big(\dot{\theta}_1 + \dot{\theta}_2\big)\big(l_{c3y} c_{12} - l_{c3x} s_{12}\big) - \dot{\theta}_1\big(l_1 s_1 + l_2 s_{12}\big) - \dot{\theta}_2 l_2 s_{12} \\ \big(\dot{\theta}_1 + \dot{\theta}_2\big)\big(l_{c3x} c_{12} + l_{c3y} s_{12}\big) + \dot{\theta}_1\big(l_1 c_1 + l_2 c_{12}\big) + \dot{\theta}_2 l_2 c_{12} \\ -\dot{d}_3 \end{bmatrix}$$

$$\dot{P}^0_{CG_4} = \begin{bmatrix} \big(\dot{\theta}_1 + \dot{\theta}_2 - \dot{\theta}_4\big)\big(l_{c4y} c_{12\bar{4}} - l_{c4x} s_{12\bar{4}}\big) - \dot{\theta}_1\big(l_1 s_1 + l_2 s_{12}\big) - \dot{\theta}_2 l_2 s_{12} \\ \big(\dot{\theta}_1 + \dot{\theta}_2 - \dot{\theta}_4\big)\big(l_{c4x} c_{12\bar{4}} + l_{c4y} s_{12\bar{4}}\big) + \dot{\theta}_1\big(l_1 c_1 + l_2 c_{12}\big) + \dot{\theta}_2 l_2 c_{12} \\ -\dot{d}_3 \end{bmatrix}$$

$$\omega^0_{CG_1} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix}$$

$$\omega^0_{CG_2} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix}$$

$$\omega^0_{CG_3} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix}$$

$$\omega^0_{CG_4} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 - \dot{\theta}_4 \end{bmatrix}$$

Note that the variables in the form l_{cix} or l_{ciy} represent the x and y coordinates of the location of the centre of gravity of link i, expressed with respect

to the frame i.

The next step in the Lagrange Method was to build expressions that represented the kinetic energy (k_i) and the gravitational potential energy (u_i) of each link i in the robot. The formulae to calculate these expressions is shown below:

$$k_i = \frac{1}{2} m_i (\dot{P}^0_{CG_i})^{\mathrm{T}} \dot{P}^0_{CG_i} + \frac{1}{2} (\omega^0_{CG_i})^{\mathrm{T}} I_{C_i} \omega^0_{CG_i} \tag{3.45}$$

$$u_i = -m_i g^{\mathrm{T}} P_{C_i} \tag{3.46}$$

In these equations, m_i represented the mass of link i in kilograms, g represented the acceleration due to gravity in metres per square second, expressed as vector with respect to frame 0, and P_{C_i} represented the position of the centre of mass of link i, expressed with respect to frame 0. I_{C_i} represented the inertia tensor of link i at its centre of mass, which took the form:

$$I_{C_i} = \begin{bmatrix} I_{xx_i} & -I_{xy_i} & -I_{xz_i} \\ -I_{xy_i} & I_{yy_i} & -I_{yz_i} \\ -I_{xz_i} & -I_{yz_i} & I_{zz_i} \end{bmatrix} \tag{3.47}$$

In this tensor, I_{xx_i}, I_{yy_i} and I_{zz_i} represented the mass moments of inertia of the link, while I_{xy_i}, I_{xz_i} and I_{yz_i} represented the mass products of inertia. Each of them was measured in kilogram square metres. When the inertia tensor is measured at the centre of mass of the link, the mass products of inertia will all equal zero. The entire inertia tensor was a physical property of each link that was held constant as long as the geometry and the density of the link does not change.

$$k(t) = k_1 + k_2 + k_3 + k_4 \tag{3.48}$$

$$u(t) = u_1 + u_2 + u_3 + u_4 \tag{3.49}$$

Once k_i and u_i were calculated for each link, they were added all together to produce expressions that represented the total kinetic ($k(t)$) and gravitational potential ($u(t)$) energies of the robot as a function of the robot's geometric and inertial constant properties, as well as the time-variant joint positions and velocities. Because the joint positions and velocities could change over time, the total kinetic and gravitational potential energies of the robot could be expressed as functions of time. Unfortunately, these expressions were too long to be shown on paper, but they can be re-calculated using the MatLab scripts shown in Appendix A.

The expressions for $k(t)$ and $u(t)$ were then plugged into Lagrange's equation below to solve for the dynamic equations of the robot:

$$\tau_i = \frac{d}{dt}\left(\frac{\partial k}{\partial \dot{\theta}_i}\right) - \frac{\partial k}{\partial \theta_i} + \frac{\partial u}{\partial \theta_i} \tag{3.50}$$

In this formula, τ_i represents the force or torque (in Newtons or Newton metres) required at joint i to maintain the kinetic/potential energies that were plugged into the right-hand side of the equation. The generalized joint force/-torque vector $\tau = \begin{bmatrix}\tau_1 & \tau_2 & \tau_3 & \tau_4\end{bmatrix}^{\mathrm{T}}$ could be created by stacking Equation 3.50 calculated for each joint on top of one another. The expression that was calculated to be equal to the vector τ represented the force/torque required at each joint to counteract the inertial, centrifugal, Coriolis, and gravitational forces/torques acting on the robot. It did not, however, represent the forces/-torques required to overcome any frictional forces (in the motors or otherwise), any electrical forces internal to the motors, or any external applied forces/-torques acting on the end effector.

The generalized force/torque vector τ could be broken down into the three types of forces/torques that it represented, the mass/inertial, the centrifugal/-Coriolis and the gravitational forces/torques:

$$\tau = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) \tag{3.51}$$

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \\ f_3 \\ \tau_4 \end{bmatrix} \tag{3.52}$$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ d_3 \\ \theta_4 \end{bmatrix} \tag{3.53}$$

In this form, $M(\theta)$ represents the mass matrix, $C(\theta, \dot{\theta})$ represents the vector of centrifugal and Coriolis terms, and $G(\theta)$ represents the vector of gravitational forces. While the entire expression for τ was so large that it does not show well in one piece, if it was broken down into these individual elements it can be shown:

$$M(\theta) = \begin{bmatrix} M_{11} & M_{12} & 0 & M_{14} \\ M_{21} & M_{22} & 0 & M_{24} \\ 0 & 0 & m_3 + m_4 & 0 \\ M_{41} & M_{42} & 0 & m_4({l_{c4x}}^2 + {l_{c4y}}^2) + I_{zz_4} \end{bmatrix} \tag{3.54}$$

$$\begin{aligned} M_{11} = {} & I_{zz_1} + I_{zz_2} + I_{zz_3} + I_{zz_4} + m_1\big((l_1 + l_{c1x})^2 + {l_{c1y}}^2\big) \\ & + m_2\big({l_1}^2 + (l_2 + l_{c2x})^2 + {l_{c2y}}^2 + m_3\big({l_1}^2 + (l_2 + l_{c3x})^2 + {l_{c3y}}^2\big) \\ & + m_4\big({l_1}^2 + {l_2}^2 + {l_{c4x}}^2 + {l_{c4y}}^2\big) + \big(m_2 + m_3 + m_4\big)l_1 l_2 c_2\big) \\ & + m_2 l_1\big(l_{c2x}c_2 + l_{c2y}s_2\big) + m_3 l_1\big(l_{c3x}c_2 + l_{c3y}s_2\big) \\ & + m_4\big(l_{c4x}(l_1 c_{2\bar{4}}) + 2l_2 c_4) + l_{c4y}(l_1 s_{2\bar{4}} - 2l_2 s_4)\big) \end{aligned} \tag{3.55}$$

$$\begin{aligned} M_{12} = M_{21} = {} & I_{zz_2} + I_{zz_3} + I_{zz_4} + m_2\big((l_2 + l_{c2x})^2 + {l_{c2y}}^2\big) \\ & + m_3\big((l_2 + l_{c3x})^2 + {l_{c3y}}^2\big) + m_4\big({l_2}^2 + {l_{c4x}}^2 + {l_{c4y}}^2\big) \\ & + \big(m_2 + m_3 + m_4\big)l_1 l_2 c_2 + m_2 l_1\big(l_{c2x}c_2 + l_{c2y}s_2\big) \\ & + m_3 l_1\big(l_{c3x}c_2 + l_{c3y}s_2\big) + m_4\big(l_{c4x}(l_1 c_{2\bar{4}} + 2l_2 c_4) \\ & + l_{c4y}(l_1 s_{2\bar{4}} - 2l_2 s_4)\big) \end{aligned} \tag{3.56}$$

$$\begin{aligned} M_{22} = {} & I_{zz_2} + I_{zz_3} + I_{zz_4} + m_2\big((l_2 + l_{c2x})^2 + {l_{C2y}}^2\big) \\ & + m_3\big((l_2 + l_{c3x})^2 + {l_{c3y}}^2\big) \\ & + m_4\big({l_2}^2 + {l_{c4x}}^2 + {l_{c4y}}^2 + 2l_2(l_{c4x}c_4 + l_{c4y}s_4)\big) \end{aligned} \tag{3.57}$$

$$\begin{aligned} M_{14} = M_{41} = {} & -m_4\big(l_{c4x}(l_{c4x} + l_2 c_4 + l_1 c_{2\bar{4}}) + l_{c4y}(l_{c4y} - l_2 s_4 + l_1 s_{2\bar{4}})\big) \\ & - I_{zz_4} \end{aligned} \tag{3.58}$$

$$M_{24} = M_{42} = -m_4\big({l_{c4x}}^2 + {l_{c4y}}^2 + l_2(l_{c4x}c_4 + l_{c4y}s_4)\big) - I_{zz_4} \tag{3.59}$$

Note that many of the terms in the mass matrix came in pairs, because the mass matrix must always be a positive semi-definite matrix.

$$C(\theta,\dot{\theta}) = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} \tag{3.60}$$

$$C_1 = 0 \tag{3.61}$$

$$\begin{aligned} C_2 = {} & l_1\dot{\theta}_1\big(\big(\dot{\theta}_1+\dot{\theta}_2\big)\big((m_2+m_3+m_4)l_2s_2+(m_2l_{c2x}+m_3l_{c3x})s_2\big) \\ & - l_1\dot{\theta}_1\big(\big(\dot{\theta}_1+\dot{\theta}_2\big)(m_2l_{c2y}+m_3l_{c3y})c_2 \\ & + \big(\dot{\theta}_1+\dot{\theta}_2-\dot{\theta}_4\big)\big(m_4l_{c4x}s_{2\bar{4}}-m_4l_{c4y}c_{2\bar{4}}\big)\big) \end{aligned} \tag{3.62}$$

$$C_3 = 0 \tag{3.63}$$

$$\begin{aligned} C_4 = {} & m_4\big(\dot{\theta}_1+\dot{\theta}_2-\dot{\theta}_4\big)\big(l_2(\dot{\theta}_1+\dot{\theta}_2)(l_{c4y}c_4+l_{c4x}s_4) \\ & + m_4\big(\dot{\theta}_1+\dot{\theta}_2-\dot{\theta}_4\big)l_1\dot{\theta}_1(l_{c4y}c_{2\bar{4}}-l_{c4x}s_{2\bar{4}})\big) \end{aligned} \tag{3.64}$$

And finally the vector of gravitational forces:

$$G(\theta) = \begin{bmatrix} 0 \\ 0 \\ g(m_3+m_4) \\ 0 \end{bmatrix} \tag{3.65}$$

Once the expressions for $M(\theta)$, $C(\theta,\dot{\theta})$ and $G(\theta)$ were found, the dynamic equations for the robot were nearly complete. However, as mentioned earlier, Equation 3.51 did not take into account any external forces/torques applied at the end effector. In order for the robot to perform any useful tasks (lifting, carrying, etc...) the model had to be be built to account for these external forces/torques. To account for these additional strains on the motors, a simple term was added to the end of the equation:

$$\tau = M(\theta)\ddot{\theta} + C(\theta,\dot{\theta}) + G(\theta) + J^{\mathrm{T}}f_{ext} \tag{3.66}$$

Where f_{ext} represented the generalized end-effector force-torque vector (i.e. the x, y and z forces stacked on top of the x, y and z torques). These forces/torques had to be expressed with respect to frame 0.

This additional term at the end takes advantage of a convenient property of the robot's Jacobian. Not only could it be used to map the end effector velocity to the joint velocities, it could also be used to map the end effector force/torque vector to the joint forces/torques due to the concept of virtual work[69].

3.4 MatLab/Simulink Model

With a mathematical model representing the robot in place, the model had to be implemented virtually to enable simulations. The model had to be built as a system that took applied joint forces/torques (τ) as inputs, and produced an end-effector trajectory as output ($x_E(t), y_E(t), z_E(t)$). The individual joint variables ($q(t)$) and joint velocities ($\dot{q}(t)$) would be internal states of the system.

The first step in the development of the virtual model was the assignment of shapes and weights to each of the links so that their inertial properties could be calculated. In order to do this, each of the four links, along with a base piece were all built in Solidworks. Images of these models are shown below in Figure 3.6:

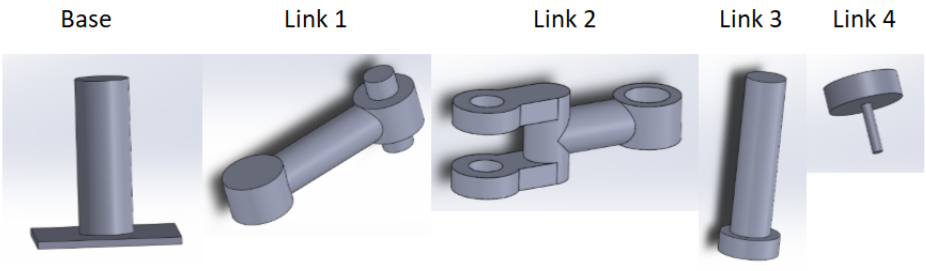

Figure 3.6: Solidworks models of all four robot links and the base

Next, the parts were all imported into MatLab's Simscape Multibody App, and linked together. Simscape Multibody allows users to assign densities to solid objects, and so all parts were assigned densities of 1000 $\frac{\text{kg}}{\text{m}^3}$. Figure 3.7 below shows the assembled configuration of all the parts:

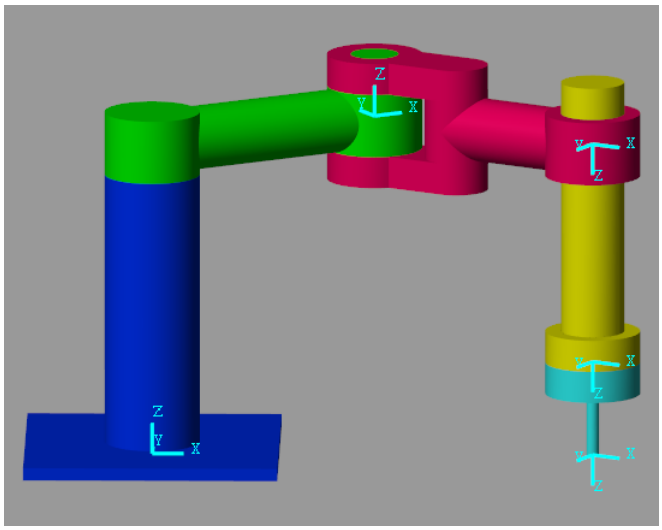

Figure 3.7: Simscape Multibody Assembly of Robot Links

With the densities and geometries of all the parts known, the inertial properties could all be calculated. The results of these calculations are shown below in Table 3.2:

Table 3.2: Inertial parameters for robot model links

Link	**Link 1**	**Link 2**	**Link 3**	**Link 4**
Length (m)	0.4	0.4	d_3	0.15
Mass (kg)	6.01	5.37	4.03	0.91
Density $\left(\frac{\text{kg}}{\text{m}^3}\right)$	1000	1000	1000	1000
l_{cix} (m)	-0.185	-0.224	0	0
l_{ciy} (m)	0	0	0	0
l_{ciz} (m)	0	0	-0.201	-0.122
I_{xx_i} $(\text{kg} \cdot \text{m}^2)$	0.0132	0.0234	0.0802	0.0016
I_{yy_i} $(\text{kg} \cdot \text{m}^2)$	0.1810	0.1261	0.0802	0.0016
I_{zz_i} $(\text{kg} \cdot \text{m}^2)$	0.1807	0.1558	0.0064	0.0025

An important observation from this table is that all of the links had centres of mass that were displaced from their associated frames of reference in only one of the three dimensions, due to rotational symmetry. This simplified the dynamic equations considerably, as many of the terms disappearred when l_{cix}, l_{ciy} or l_{ciz} were zero. With these values substituted in, the M matrix was thus simplified to:

$$M(\theta) = \begin{bmatrix} 2.34\cos\theta_2 + 3.23 & 1.17\cos\theta_2 + 1.12 & 0 & -0.0025 \\ 1.17\cos\theta_2 + 1.12 & 1.12 & 0 & -0.0025 \\ 0 & 0 & 4.94 & 0 \\ -0.0025 & -0.0025 & 0 & 0.0025 \end{bmatrix} \tag{3.67}$$

... and the C and G vectors were simplified to:

$$C(\theta, \dot{\theta}) = \begin{bmatrix} 0 \\ 1.17\dot{\theta_1}\big(\dot{\theta_1} + \dot{\theta_2}\big)\sin\theta_2 \\ 0 \\ 0 \end{bmatrix} \qquad G(\theta) = \begin{bmatrix} 0 \\ 0 \\ -48.4 \\ 0 \end{bmatrix} \tag{3.68}$$

The only variables that made any changes to the dynamic equations of motion were θ_2, $\dot{\theta_1}$ and $\dot{\theta_2}$. This made sense, because the angle θ_2 was the only joint variable that could affect the configuration of the robot in a way that would have an effect on the other joints (if the elbow was extended the entire arm would require more torque at joint 1 to rotate than if it was retracted). Additionally, it made sense that $\dot{\theta_1}$ and $\dot{\theta_2}$ would affect the C matrix, as they would increase the centrifugal/Coriolis forces acting on joint 2.

With the physical properties of the robot model established, it became possible to model the dynamic system in MatLab's Simulink. The dynamic system was modelled to take torque commands as inputs, and to produce joint angles/positions, velocities, and accelerations as output. In order to do so, Equation 3.51 was rearranged as follows:

$$\tau = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) + J^{\mathrm{T}} f_{ext} \tag{3.69}$$

$$M(\theta)\ddot{\theta} = \tau - C(\theta, \dot{\theta}) - G(\theta) + J^{\mathrm{T}} f_{ext} \tag{3.70}$$

$$\ddot{\theta} = M(\theta)^{-1}\big(\tau - C(\theta, \dot{\theta}) - G(\theta) + J^{\mathrm{T}} f_{ext}\big) \tag{3.71}$$

Using Simulink, the calculated $\ddot{\theta}$ signal could then have two numerical integrations performed on it to come up with $\dot{\theta}$ and θ signals. These two signals could be fed back into the equation through a feedback loop to continue the simulation, and could also be used as the system output. Put together in Simulink, the model of the system dynamics is shown below in Figure 3.8.

Note that the MatLab function block `M_inv` had the following internal code:

```
function Minv = M__inv(theta2)

Minv =  [-0.0027974/(0.0034132*cos(theta2)^2 -
    0.0058971), (0.0029211*cos(theta2) + 0.0027974)
    /(0.0034132*cos(theta2)^2 - 0.0058971), 0,
    (0.0029211*cos(theta2))/(0.0034132*cos(theta2)^2 -
    0.0058971); (0.0029211*cos(theta2) + 0.0027974)
    /(0.0034132*cos(theta2)^2 - 0.0058971),
    -(1.0*(0.0058422*cos(theta2) + 0.0080676))
    /(0.0034132*cos(theta2)^2 - 0.0058971), 0,
    -(1.0*(0.0029211*cos(theta2) + 0.0052703))
    /(0.0034132*cos(theta2)^2 - 0.0058971); 0, 0,
    0.20243, 0; (0.0029211*cos(theta2))/(0.0034132*cos(
    theta2)^2 - 0.0058971), -(1.0*(0.0029211*cos(theta2)
    + 0.0052703))/(0.0034132*cos(theta2)^2 - 0.0058971),
    0, (1.3653*cos(theta2)^2 - 2.3641)/(0.0034132*cos(
    theta2)^2 - 0.0058971)];
```

... and the MatLab function block `C` was made up of the following code:

```
function C = C(theta2,theta1dot,theta2dot)
```

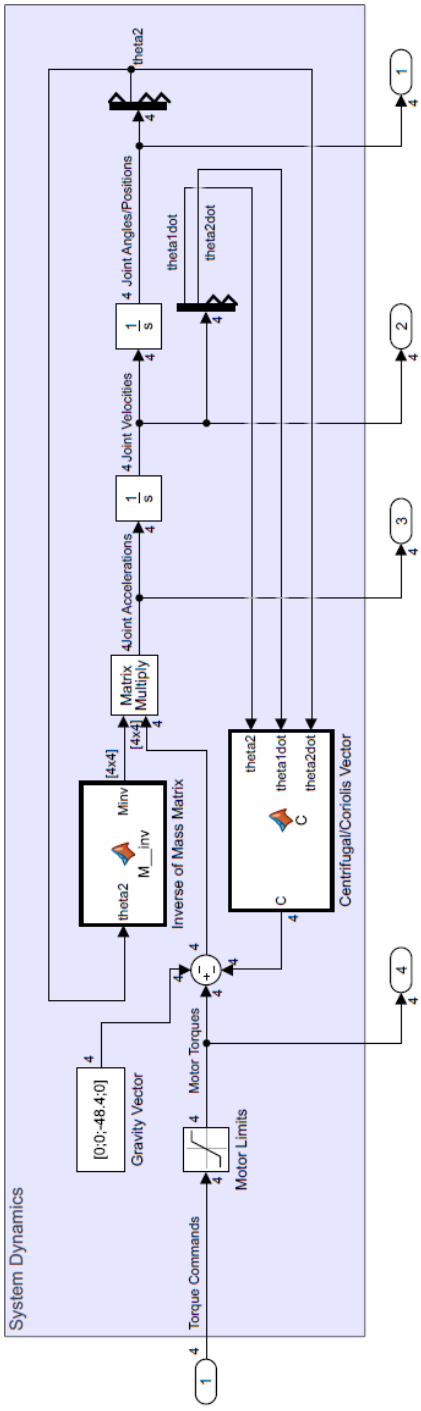

Figure 3.8: Simulink Model of System Dynamics

```
C =   [ 0; (18257*sin(theta2)*theta1dot^2)/15625 +
    (18257*theta2dot*sin(theta2)*theta1dot)/15625; 0; 0];
```

With this model established, controllers could be developed and tested in Simulink. An additional model was built in Simscape Multibody that could be used in parallel to the pure Simulink model. This would allow for a visual representation of the robot arm movement, as well as a troubleshooting/comparison aide for the Simulink model. Figure 3.9 below shows the Simscape Multibody model of the robot that was developed.

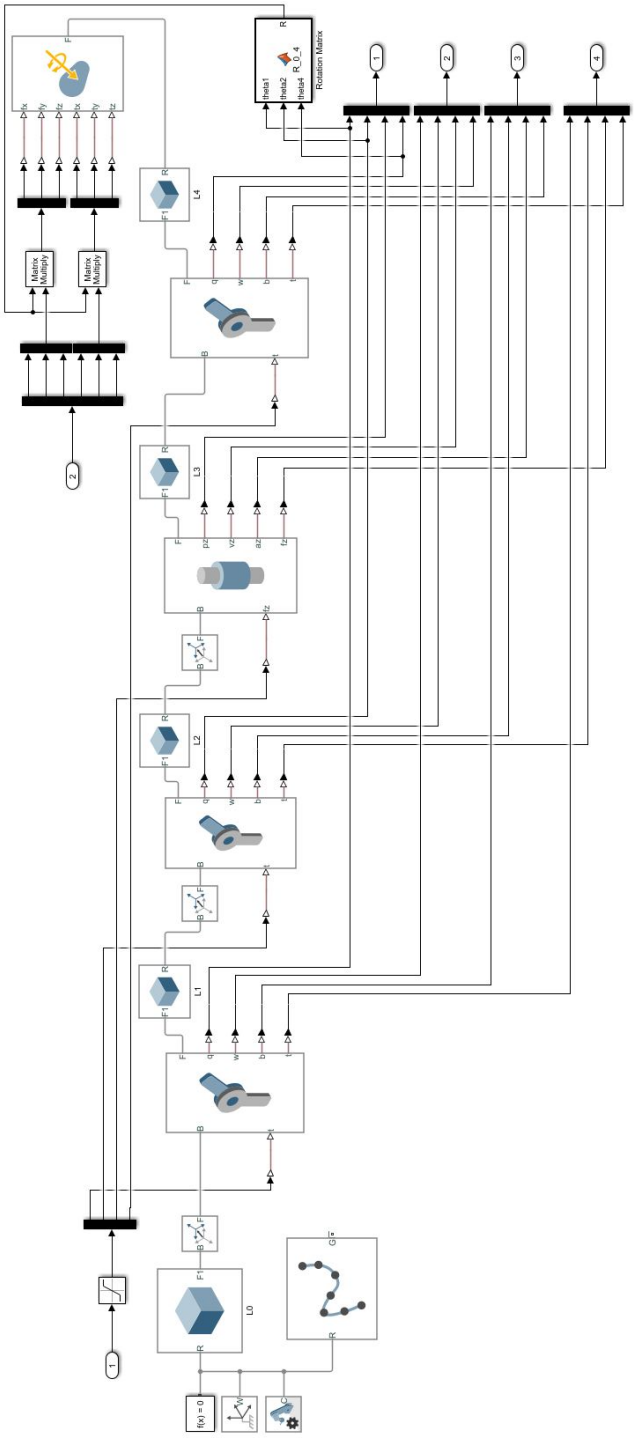

Figure 3.9: SimScape Multibody Model of System Dynamics

4 Position Control

With the model of the robot fully developed both theoretically and in simulations, a set of command signals to send to each of the joints was required in order to test it. In order to come up with a set of command forces/torques, a controller had to be developed.

The focus of this project was to develop a hybrid force/position controller, and the approach to this development was to move forward incrementally through each level of complexity. The first and simplest level of the controller that could be developed was the position controller.

Position control of the end effector of a robot is a well-developed science and can be achieved through a wide variety of different methods. Some of these methods are simple, and some are very complex. Generally, the simplicity of the controller comes at a cost of lower performance, while the more complex controllers can achieve stronger performance. To be specific, the term "performance" in this case refers to the ability of the controller to force the robot's end effector to follow a given trajectory within its workspace with minimal tracking error.

4.1 Decentralized PID Control

A Decentralized Proportional-Integral-Derivative controller was among the simplest options that could effectively be used to control the position of the end effector. PID is a commonly used type of linear controller known for its simplicity and robustness (in comparison to other linear controllers). The

functionality of a PID controller can be described as follows:

1. A desired trajectory is provided as input for the controller. This trajectory must be provided as a set of desired positions as a function of time.
2. The first time derivative and integrals of the desired trajectory are taken numerically and stored.
3. Measurements of the state of the controlled variables are taken using sensors, and numerical derivatives and integrals are taken of these signals as well.
4. The difference between the measured state and the desired state is multiplied by the proportional gain (K_P), and added to the difference between the derivatives of the measured and desired states multiplied by the derivative gain (K_D), which is then added to the difference between the integrals of the measured and desired states multiplied by the integral gain (K_I).
5. This resulting signal is then applied as command signal to the system actuator(s).

This algorithm can be represented with the following simple control law:

$$\tau = K_D\dot{e}(t) + K_P e(t) + K_I \int_0^t e(t)dt \tag{4.1}$$

... where $e(t)$ represents the difference between the desired trajectory and the actual position of each joint.

For any linear system, it is known that there exists some set of controller gains (K_P, K_I and K_D) that will cause the actual state of the controlled variable to converge to certain types of desired trajectories. For step input trajectories, a properly designed PID controller will yield asymptotic stability. For linear ramp input trajectories, a properly designed PID controller will yield stability, albeit with a non-zero steady-state error [71].

Unfortunately, the robot system in question was highly nonlinear. The M matrix and C vector contained multiple sinusoidal functions that themselves

were nonlinear, and the C vector also contained polynomials that made it more nonlinear. In order to linearize the system for the PID controller, it could be assumed that the robot was always in its fully extended (i.e. $\theta_2 = 0$) configuration. This configuration would be the most mechanically intensive for the controller to move, so this assumption would be more likely to cause the controller to be over-powered than to be under-powered, which was good for performance. The new M matrix and C vector were shown below with the $\theta_2 = 0$ assumption in place:

$$M(\theta) = \begin{bmatrix} 5.57 & 2.29 & 0 & -0.0025 \\ 2.29 & 1.12 & 0 & -0.0025 \\ 0 & 0 & 4.94 & 0 \\ -0.0025 & -0.0025 & 0 & 0.0025 \end{bmatrix} \qquad C(\theta, \dot{\theta}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.2}$$

As can be seen below in Equation 4.3, if gravity was ignored, the $\theta_2 = 0$ assumption fully linearized the system. It must be noted of course, that using this assumption would certainly introduce tracking error to the system. This was because the system's nonlinearities would act as unmodeled disturbances that the linearized PID controller was not equipped to deal with. Luckily, due to the PID controller's robustness (in particular, the higher order introduced through the integral term), as long as these "disturbances" remained bounded over time, that is to say they would not continue to increase as t approaches infinity, the error that they initially caused could eventually be reduced to zero.

$$\tau = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) = \begin{bmatrix} 5.57\ddot{\theta_1} + 1.29\ddot{\theta_2} - 0.0025\ddot{\theta_4} \\ 1.29\ddot{\theta_1} + 1.12\ddot{\theta_2} - 0.0025\ddot{\theta_4} \\ 4.94\ddot{d_3} \\ -0.0025\ddot{\theta_1} - 0.0025\ddot{\theta_2} + 0.0025\ddot{\theta_4} \end{bmatrix} \tag{4.3}$$

To further simplify the system, all of the coupling between joints would be ignored and each joint would only be controlled as if it were the only

one moving. Thus, each joint would effectively be running its own separate controller, operating independently of the other three joint controllers. It was for this reason that this strategy was called *Decentralized* PID Control. This would certainly reduce the performance of the robot, but as long as the speeds of the system were low the effects would be small. By ignoring coupling, the system dynamics could be reduced to:

$$\tau = \begin{bmatrix} 5.57\ddot{\theta}_1 \\ 1.12\ddot{\theta}_2 \\ 4.94\ddot{d}_3 \\ 0.0025\ddot{\theta}_4 \end{bmatrix} \tag{4.4}$$

The transfer functions of these four equations were thus:

$$\frac{\theta_1(s)}{\tau(s)} = \frac{1}{5.57s^2} \tag{4.5}$$

$$\frac{\theta_2(s)}{\tau(s)} = \frac{1}{1.12s^2} \tag{4.6}$$

$$\frac{D_3(s)}{\tau(s)} = \frac{1}{4.94s^2} \tag{4.7}$$

$$\frac{\theta_4(s)}{\tau(s)} = \frac{1}{0.0025s^2} \tag{4.8}$$

Each of these transfer functions was then put into the block diagram shown below in Figure 4.1:

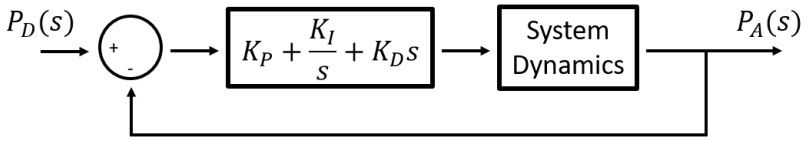

Figure 4.1: General PID controller block diagram

Note that in this diagram, $P_D(s)$ represented the desired trajectory of the joint variable being controlled, while $P_A(s)$ represented the actual trajectory

of the joint variable. The block diagram was then simplified to find the closed-loop system characteristic equations below:

Joint 1:

$$s^3 + \frac{K_D}{5.57}s^2 + \frac{K_P}{5.57}s + \frac{K_I}{5.57} = 0 \tag{4.9}$$

Joint 2:

$$s^3 + \frac{K_D}{1.12}s^2 + \frac{K_P}{1.12}s + \frac{K_I}{1.12} = 0 \tag{4.10}$$

Joint 3:

$$s^3 + \frac{K_D}{4.94}s^2 + \frac{K_P}{4.94}s + \frac{K_I}{4.94} = 0 \tag{4.11}$$

Joint 4:

$$s^3 + \frac{K_D}{0.0025}s^2 + \frac{K_P}{0.0025}s + \frac{K_I}{0.0025} = 0 \tag{4.12}$$

Next, the target poles of the system were arbitrarily set to -8, -9 and -10. Poles this far into the left-hand side of the complex plane would guarantee stability for the linearized system. If the system was not performing well enough, these could be pushed further left, and if it was performing well but requiring unrealistic command torques and forces from the motors, these could be pushed further to the right. If these two trade-offs were proving difficult to compromise between, the Linear Quadratic Regulator (LQR) could be used to optimize the best location for the poles, and if this didn't work a different controller type would be selected. With these poles, the target characteristic equation was worked out to be:

$$(s+8)(s+9)(s+10) = 0 \tag{4.13}$$

$$s^3 + 27s^2 + 242s + 720 = 0 \tag{4.14}$$

Which yielded final control gains of:

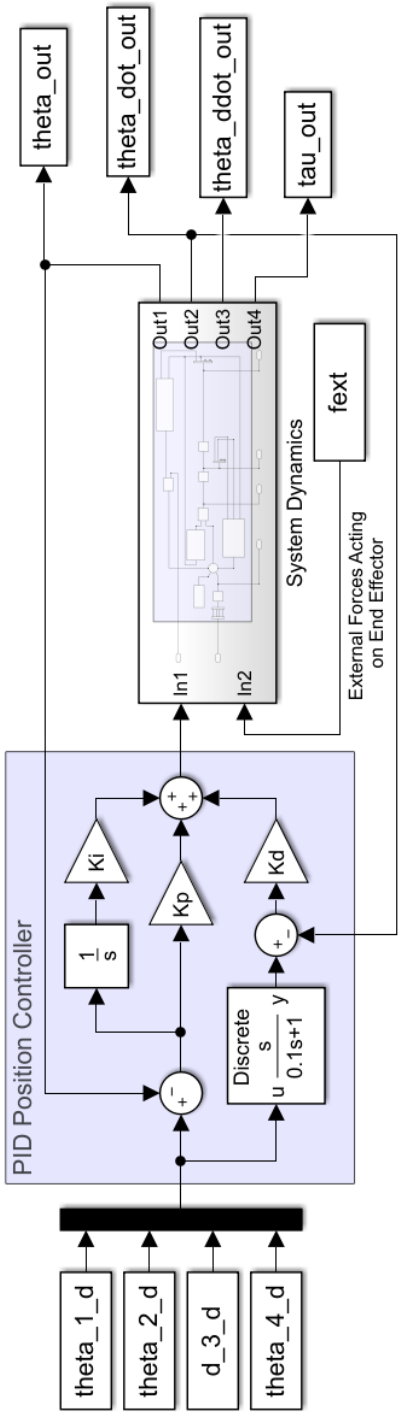

Figure 4.2: Simulink model of PID position controller

$$K_P = \begin{bmatrix} 1347.94 \\ 271.04 \\ 1195.48 \\ 0.605 \end{bmatrix} \qquad K_I = \begin{bmatrix} 4010.4 \\ 806.4 \\ 3556.8 \\ 1.8 \end{bmatrix} \qquad K_D = \begin{bmatrix} 150.39 \\ 30.24 \\ 133.38 \\ 0.0675 \end{bmatrix} \tag{4.15}$$

Figure 4.2 shows the assembled PID position controller in Simulink. Note that the same exact controller was used for both the Simulink and the Simscape Multibody models.

To test the performance of the Decentralized PID controller, a three-dimensional trajectory was built using a combination of sinusoidal curves. The desired trajectory $r(t)$ was arbitrarily set to be:

$$r(t) = \begin{bmatrix} x_d \\ y_d \\ z_d \\ \psi_d \\ \nu_d \\ \phi_d \end{bmatrix} = \begin{bmatrix} 0.8 - 0.4\frac{t}{T} - 0.05\sin\left(\frac{4\pi t}{T}\right) \\ -0.2\frac{t}{T} + 0.15\sin\left(\frac{3\pi t}{2T}\right) \\ 0.25 - 0.1\frac{t}{T} - 0.005\sin\left(\frac{15\pi t}{T}\right) \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.16}$$

Where T represented the total trial time for the simulation (in this case set to 10 seconds). This trajectory was chosen because to challenge the controller to follow a non-linear end effector path without violating the workspace boundaries. It also stayed within the zone that was accessible using both the elbow-up and elbow-down configurations (i.e. the purple areas from Figure 3.4).

Figure 4.3 below shows both the reference trajectory $r(t)$ in black, and the Decentralized PID controller's output trajectory in a rainbow colour, starting at blue and ending in red. From this image alone we can see that the Decentralized PID controller was able to track the reference trajectory, although with a relatively large tracking error at the beginning of the trial. Figures 4.4 and 4.5 show the performance of each individual joint's controller, again

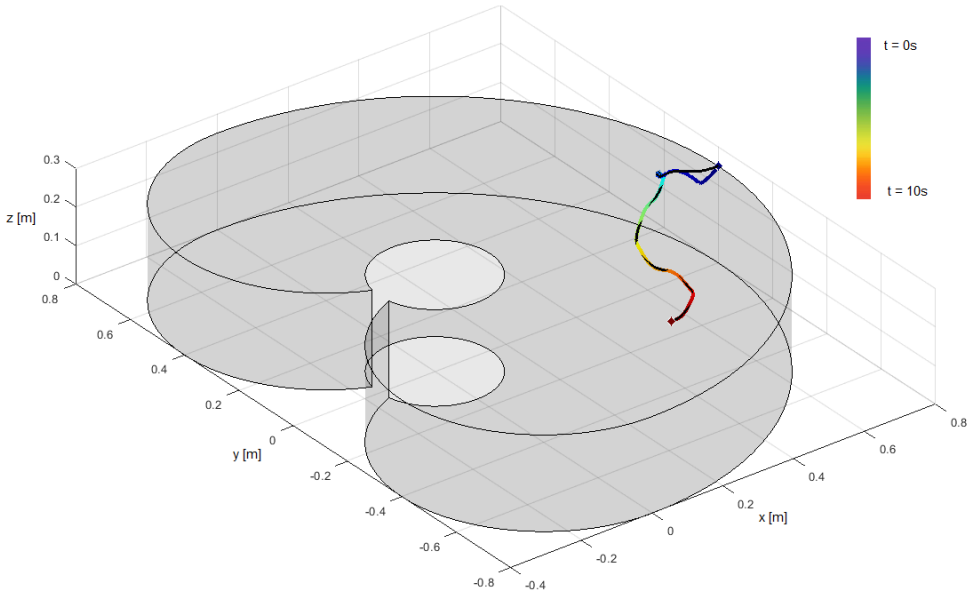

Figure 4.3: Decentralized PID controller 3D tracking results

showing that the reference trajectory was well-tracked, but with significant tracking error during the first second of the trial.

The tracking errors (defined as the difference between the desired trajectory and the actual trajectory) are shown in Figure 4.6. Note that both the error from the main Simulink simulation and from the Simscape Multibody trials are shown. Because the errors were relatively stable after 2 seconds, the final 8 seconds of the trial are not shown. allowing a better focus on the first second of the trial.

An important observation from Figure 4.6 is that although the results from the Simulink trials and the Simscape Multibody trials were similar, they were not exactly the same. They followed the same trends, but they did not always have the exact same magnitudes. In fact, this slight difference in performance was a good indicator that the Simulink model that was built from scratch is valid. Because there was a non-zero difference in performance, we know that there were two separate sets of solvers being used, but because the results

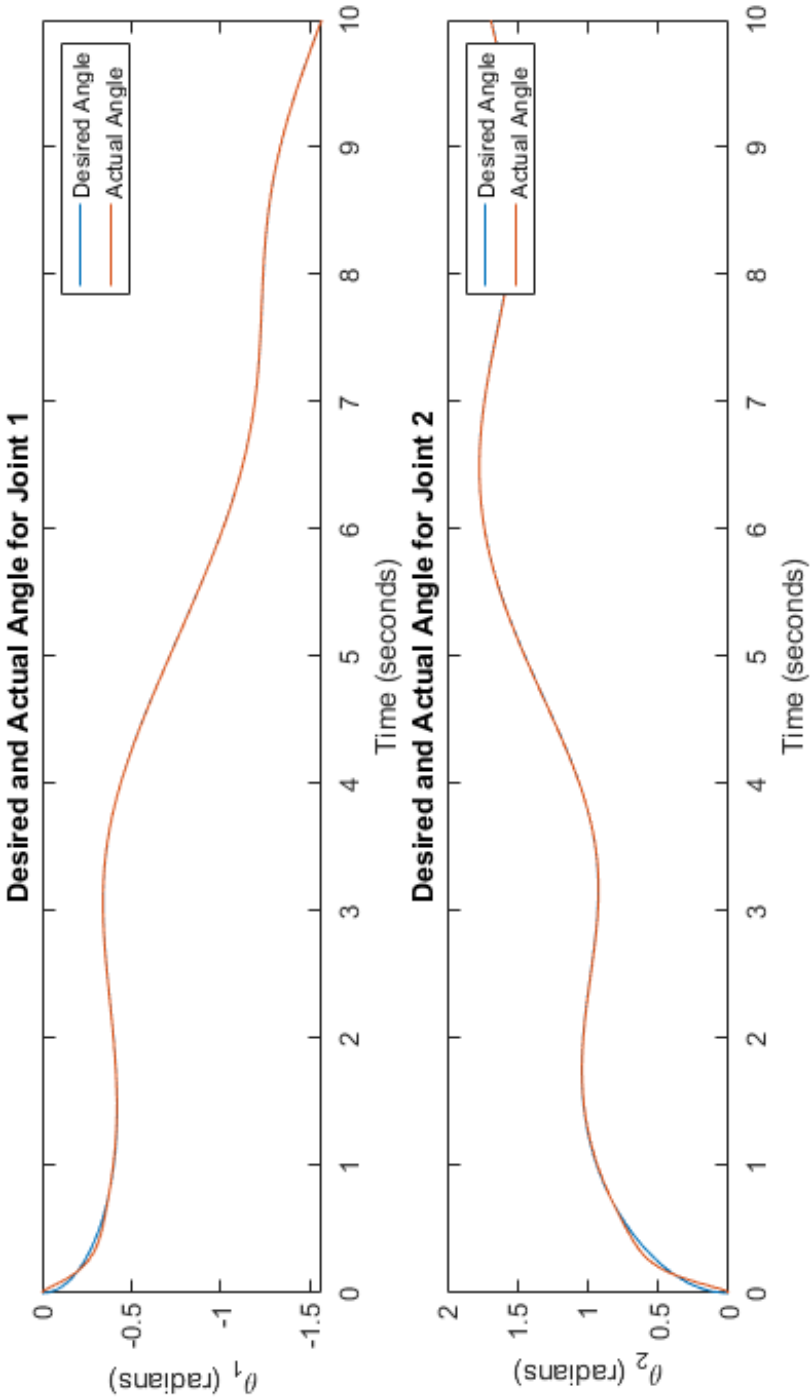

Figure 4.4: Joint 1 and 2 tracking performance for Decentralized PID controller

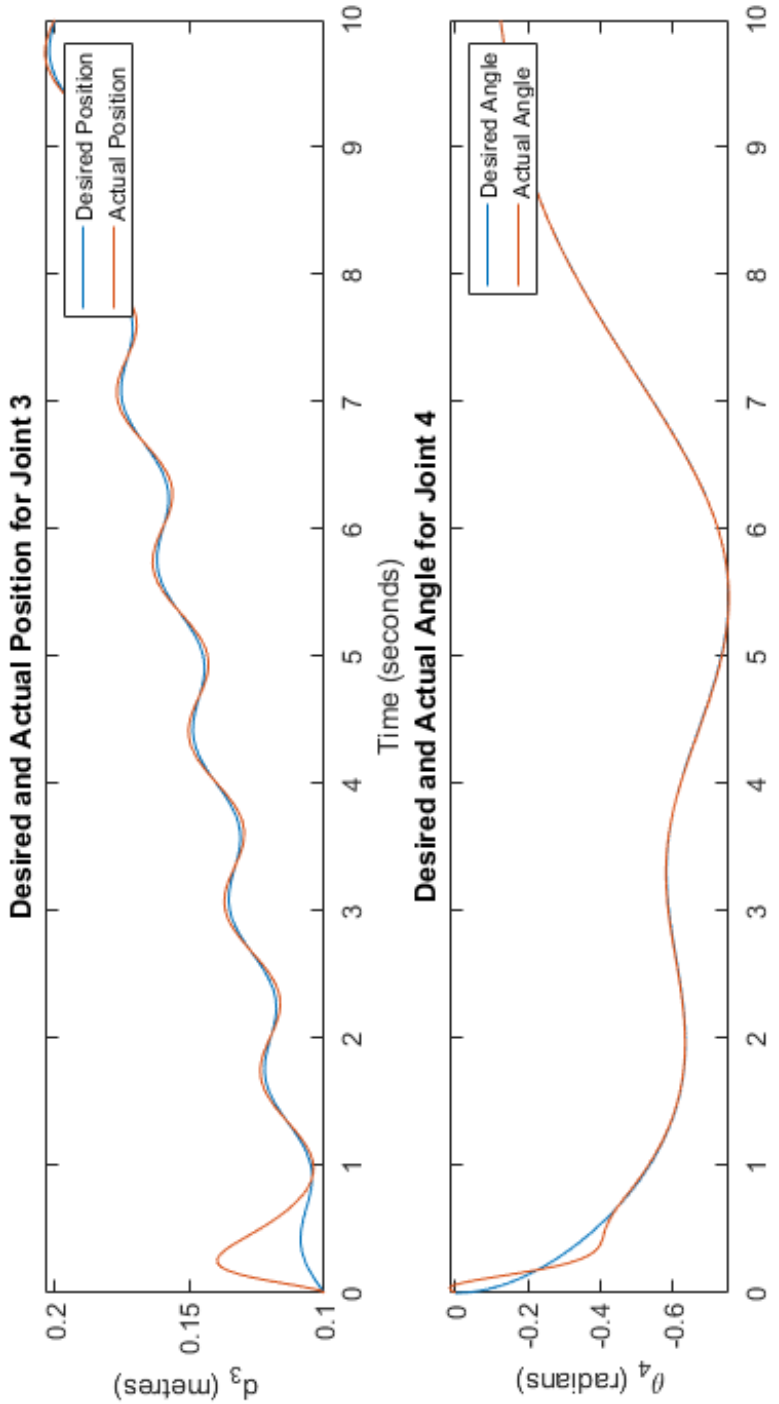

Figure 4.5: Joint 3 and 4 tracking performance for Decentralized PID controller

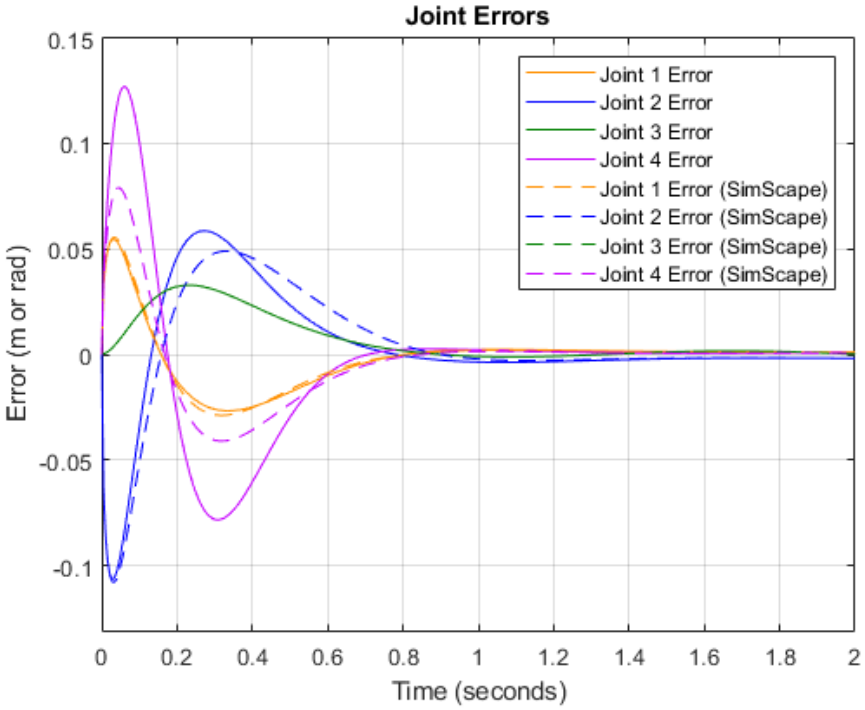

Figure 4.6: Decentralized PID controller joint errors

were very similar, we know that the model built from scratch replicated the dynamic model that formed the basis of the Simscape Multibody platform.

Another important observation is that the discrepancy between the Simulink results and the Simscape Multibody results did not seem to exist for Joint 3. In Figure 4.6, the Joint 3 Simscape Multibody line was never visible because it was exactly the same as the Joint 3 Simulink line. This was initially concerning, because it suggested that there was be some unintended link between the two models, or perhaps that the Joint 3 Simscape Multibody data simply didn't exist. After further investigation however, it was found that the two data sets were in fact distinct, they were just so similar that it was impossible to see the difference in the plot. The reason that the Joint 3 data sets were so much more similar than the other three joints, was because the scaling for Joint 3 was necessarily different. Since Joint 3 was the only prismatic joint,

it was the only Joint whose error was denominated in metres, not radians. Additionally, because Joint 3 was the only joint that was affected by gravity (see Equation 3.65), the performance of its controller was significantly worse than that of the other three joints. This meant that error for Joint 3 was much larger than the other three joints (but the fact that it was the only prismatic joint meant that it would be scaled on the plot to match the other three), and therefore the difference between the Simulink and Simscape Multibody errors would be dramatically less noticeable.

While the Decentralized PID controller was able to successfully track the trajectory, it did so with some significant tracking error. In order to increase the position tracking performance prior to moving forward with the force control, an increased layer of sophistication was added.

4.2 Decentralized PID Control with Feedback Linearization

The majority of the tracking error for the Decentralized PID Controller came from the system's nonlinearities that were ignored during the linearization process. To reduce these errors, an internal feedback loop could be added to the control law to linearize the system dynamically, that is to say to linearize it with more appropriate assumptions than simply assuming the robot was in its fully extended configuration at all times.

First, we will recall the system's dynamic equations (3.66), and rewrite them in the following form:

$$\tau = M_{linear}\ddot{\theta} + M_{nonlinear}(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) + J^{\mathrm{T}} f_{ext} \tag{4.17}$$

Where M_{linear} represented the linear, time-invariant parts of the M matrix, and $M_{nonlinear}(\theta)$ represented the nonlinear/time-variant parts of the M matrix. From Equation 3.67 we can see that they were be equal to:

$$M_{linear} = \begin{bmatrix} 3.23 & 1.12 & 0 & -0.0025 \\ 1.12 & 1.12 & 0 & -0.0025 \\ 0 & 0 & 4.94 & 0 \\ -0.0025 & -0.0025 & 0 & 0.0025 \end{bmatrix} \tag{4.18}$$

$$M_{nonlinear}(\theta) = \begin{bmatrix} 2.34\cos\theta_2 & 1.17\cos\theta_2 & 0 & 0 \\ 1.17\cos\theta_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.19}$$

Using the following change of variables from τ to u:

$$u = \tau - M_{nonlinear}(\theta)\ddot{\theta} - C(\theta, \dot{\theta}) - G(\theta) \tag{4.20}$$

... and again assuming that $f_{ext} = 0$ because we were still only modelling the free movement of the robot through space, we were left with the following linearized system dynamics:

$$u = M_{linear}\ddot{\theta} \tag{4.21}$$

Using the same process as was used in the development of the previous controller, a set of new PID gains (K_{P2}, K_{I2} and K_{D2}) was developed to ensure convergence. Using the same desired poles of -8. -9 and -10, and again ignoring the coupling (i.e. non-diagonal) elements in M_{linear} to maintain the decentralized control strategy, the following new PID gains were found:

$$K_{P2} = \begin{bmatrix} 781.66 \\ 271.04 \\ 1195.48 \\ 0.605 \end{bmatrix} \qquad K_{I2} = \begin{bmatrix} 2325.6 \\ 806.4 \\ 3556.8 \\ 1.8 \end{bmatrix} \qquad K_{D2} = \begin{bmatrix} 87.21 \\ 30.24 \\ 133.38 \\ 0.0675 \end{bmatrix} \tag{4.22}$$

Using the same algorithm as from Equation 4.1, the simple dynamics of Equation 4.21 could be stabilized with the following control law:

$$u = K_{D2}\dot{e}(t) + K_{P2}e(t) + K_{I2}\int_0^t e(t)dt \tag{4.23}$$

This PID control law could then be combined with the feedback linearization inner loop from Equation 4.24 to create the following overall control law:

$$\tau = K_{D2}\dot{e}(t) + K_{P2}e(t) + K_{I2}\int_0^t e(t)dt + M_{nonlinear}(\theta)\ddot{\theta} + C(\theta,\dot{\theta}) + G(\theta) \tag{4.24}$$

A rough block diagram showing the implementation of a Feedback Linearization/PID strategy is shown in Figure 4.7.

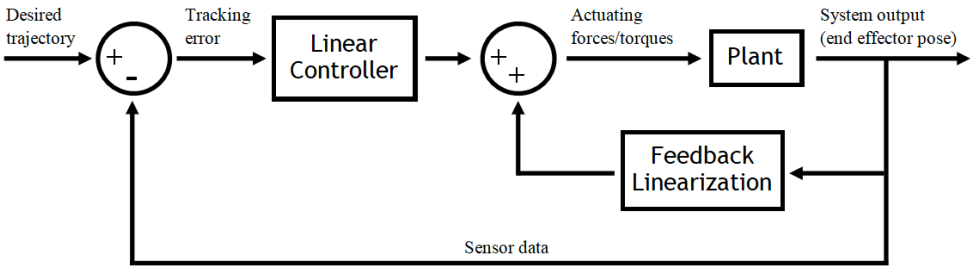

Figure 4.7: Feedback Linearization/PID control block diagram

Note that in order to implement this law, the controller had to have access to θ, $\dot{\theta}$ and $\ddot{\theta}$ which would either require velocity and acceleration sensors (on top of the already-necessary position/angle sensors), or the use of numerical derivatives (potentially even double derivatives) that could be significantly noisy and prone to measurement error. This would not make a major effect on the performance in the simulations, as we had easy access to these values, but it would certainly be an important factor to consider in the actual physical controller. The Simulink block diagram for the Decentralized PID with Feedback Linearization controller is shown in Figure 4.8.

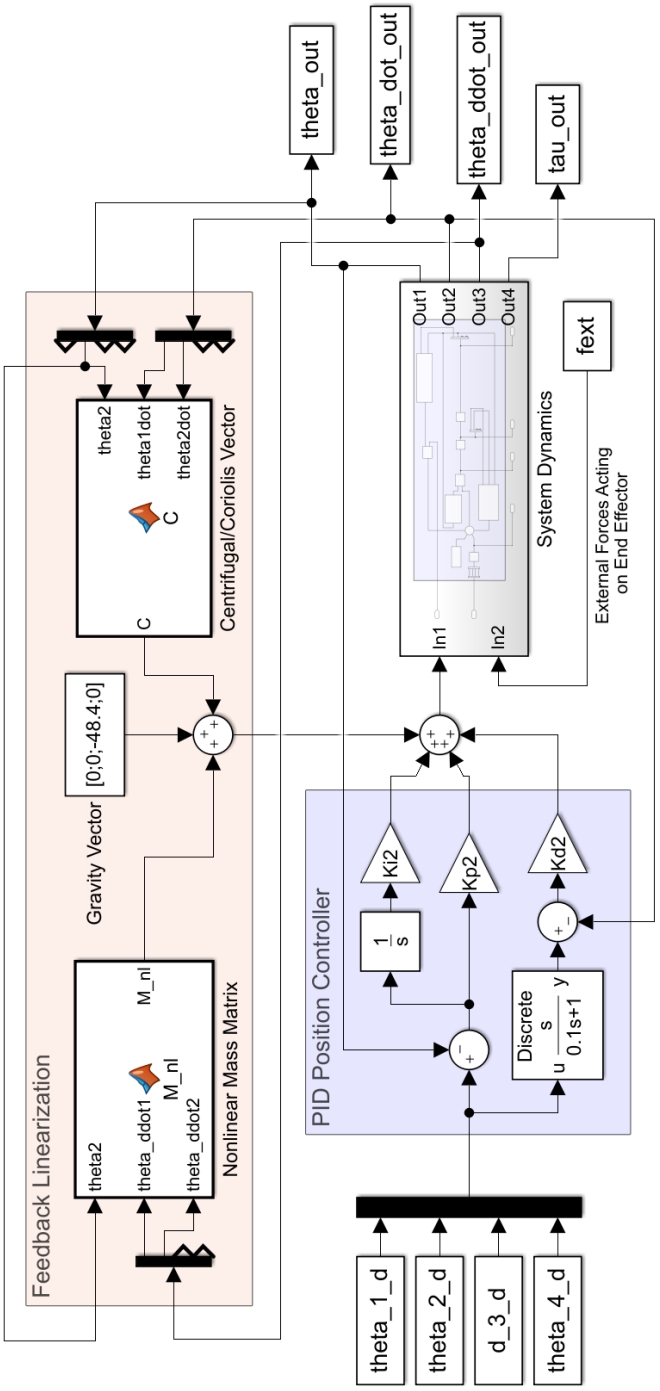

Figure 4.8: Simulink model of PID position controller with Feedback Linearization

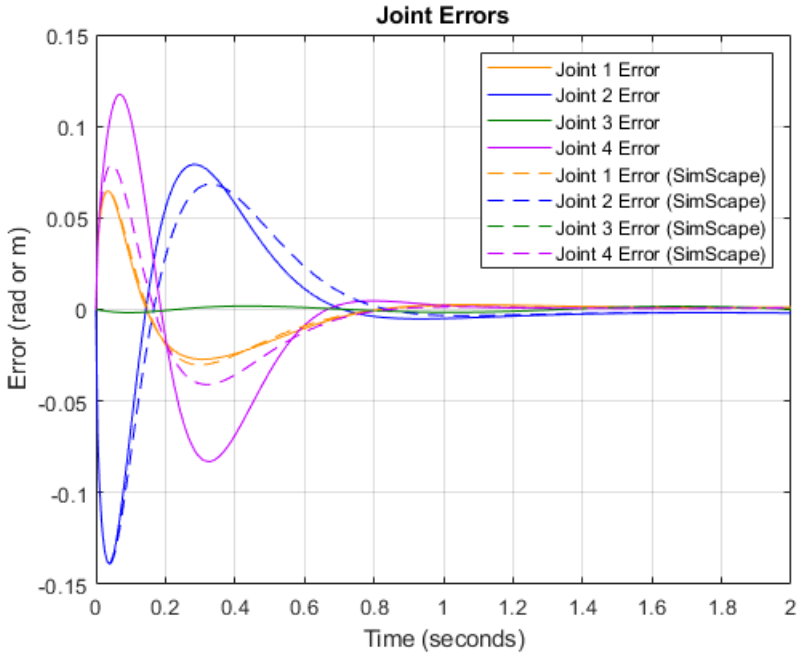

Figure 4.9: Joint errors for Decentralized PID/Feedback Linearization controller

When implemented, the Decentralized PID/Feedback Linearization controller out-performed its predecessor, as expected. The resulting errors for each joint are shown again in Figure 4.9:

The significant improvement in performance showed up almost exclusively in Joint 3. The reason for this was because Joint 3 had previously been affected significantly by gravity, and the new controller included a gravity compensation term. The resultant error for Joint 3 was very minimal, only slight periodic undulations as a result of the sinusoidal nature of the desired trajectory. Notably, Joint 3 was the only joint that did not have a significant error at the start of the trial that was slowly cancelled over time. This was because Joint 3's dynamics being completely decoupled (i.e. independent) of the other three joints, as can be seen in Equations 3.67 and 3.68. Because of this decoupling, the Decentralized PID controller with Feedback Linearization did not need to ignore any dynamics. In the other three joints, the coupling had to be ignored, and thus was a source of error. The third position controller that was built was an attempt to reduce that error.

As previously mentioned, this improved performance came at a cost of increased complexity, and an increased requirement for online computation and/or sensors. These simulations did not accurately reflect this cost, and so it must always be mentioned when discussing the results. If this computational/sensor cost were too much of a problem, the control law could be slightly modified to reduce it. Instead of using online measured data from the robot's sensors, the values for θ, $\dot{\theta}$ and $\ddot{\theta}$ could have been pre-computed using the values of θ from the desired trajectory. This pre-computing would reduce the computational burden on the controller at the cost of slightly increased error, due to the fact that the desired values of θ would not always be guaranteed to be equal to the actual value of θ during the trial. This control strategy is known as the *Computed Torque* method.

4.3 Centralized Control with Feedback Linearization

The largest remaining source of error in the previous controller came from ignoring the coupling between the joints in pursuit of a decentralized approach. In order to reduce those errors, the logical solution was to use a centralized controller that could account for the dynamic coupling between the joints. A full state feedback control law is a classic example of a centralized controller for linear systems, and combining it with an inner feedback linearization loop would allow for the benefits of centralized control to be applied to a nonlinear, time-variant system.

We will start with Equation 4.21, which was used in the development of the previous controller. The same feedback linearization was be used in this controller, but instead of using PID control to stabilize the system, full state feedback was used. Full state feedback requires the system to be represented in state-space form. To achieve this, two state variables, x_1 and x_2 were be introduced, each one representing a 4×1 vector in itself:

$$x_1 = \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ d_3 \\ \theta_4 \end{bmatrix} \qquad\qquad x_2 = \dot{\theta} = \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \\ \dot{d_3} \\ \dot{\theta_4} \end{bmatrix} \tag{4.25}$$

These two states were then used to represent Equation 4.21 in state space:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & I_4 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix} u \tag{4.26}$$

Note that here, the A matrix was 8×8 and the B matrix was 8×4, and were equal to:

$$A = \begin{bmatrix} 0 & I_4 \\ 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix} \tag{4.27}$$

...and that M^{-1} was the inverse of M_{linear}:

$$M^{-1} = \begin{bmatrix} 0.4739 & -0.4739 & 0 & 0 \\ -0.4739 & 1.3688 & 0 & 0.8949 \\ 0 & 0 & 0.2024 & 0 \\ 0 & 0.8949 & 0 & 400.8949 \end{bmatrix} \tag{4.28}$$

If the rank of the controllability matrix, M_C (Equation 2.8), was greater than the number of states (in this case 8), then it is said that the system is fully controllable. Using MatLab's `ctrb()` function, M_C was found to have a rank of 8, meaning that the system was fully controllable. This meant that a control law in the following form:

$$u = -K \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \theta_d \tag{4.29}$$

...could force the closed-loop poles to any desired location in the s-plane. Note that K represents a 4×8 gain matrix. Again, arbitrarily setting the locations of desired poles to -8.0, -8.1, -8.2, -8.3, -8.4, -8.5, -8.6 and -8.7, MatLab's `place()` function yielded the following K matrix:

$$K = \begin{bmatrix} K_1 & K_2 \end{bmatrix} \tag{4.30}$$

$$K_1 = \begin{bmatrix} 228.9897 & 76.5359 & 0.7344 & 1.6829 \\ 78.7720 & 78.4412 & -0.2282 & 1.3943 \\ 3.8678 & -6.2185 & 339.3889 & -6.0836 \\ -0.1742 & -0.1743 & -0.0019 & 0.1664 \end{bmatrix} \tag{4.31}$$

$$K_2 = \begin{bmatrix} 54.4107 & 18.5250 & 0.0906 & 0.1828 \\ 18.7917 & 18.7500 & -0.0260 & 0.1455 \\ 0.4604 & -0.7347 & 81.9093 & -0.7304 \\ -0.0418 & -0.0418 & -0.0002 & 0.0408 \end{bmatrix} \tag{4.32}$$

The complete control law for the full state feedback linearization controller was thus:

$$\tau = -K \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \theta_d + M_{nonlinear}(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) \tag{4.33}$$

Figure 4.10 below shows the Simulink implementation of the Full State Feedback Linearization controller. Note that the full state feedback controller was significantly more simple than the PID controller in this form, due to the simplicity of its control law.

When the Full State Feedback Linearization model was tested, the results were very poor. After some extensive troubleshooting, it became clear that the model itself was not faulty, and that the only thing likely to be causing the problem was an error in the K gain matrix itself. The controller clearly was working to force the end effector to follow the desired trajectory, it was just not very successful at doing so. One hint as to why the blame could be attributed to a sub-optimal K matrix comes from investigating its third row. Referencing Equation 4.29 we can see that this third row would be multiplied by the vectors in Equation 4.25 to yield the following formula for the control signal u_3:

$$u_3 = 3.8678\theta_1 - 6.2185\theta_2 + 339.3889d_3 - 6.0836\theta_4 \tag{4.34}$$

$$+ 0.4604\dot{\theta_1} - 0.7347\dot{\theta_2} + 81.9093\dot{d_3} - 0.7304\dot{\theta_4} \tag{4.35}$$

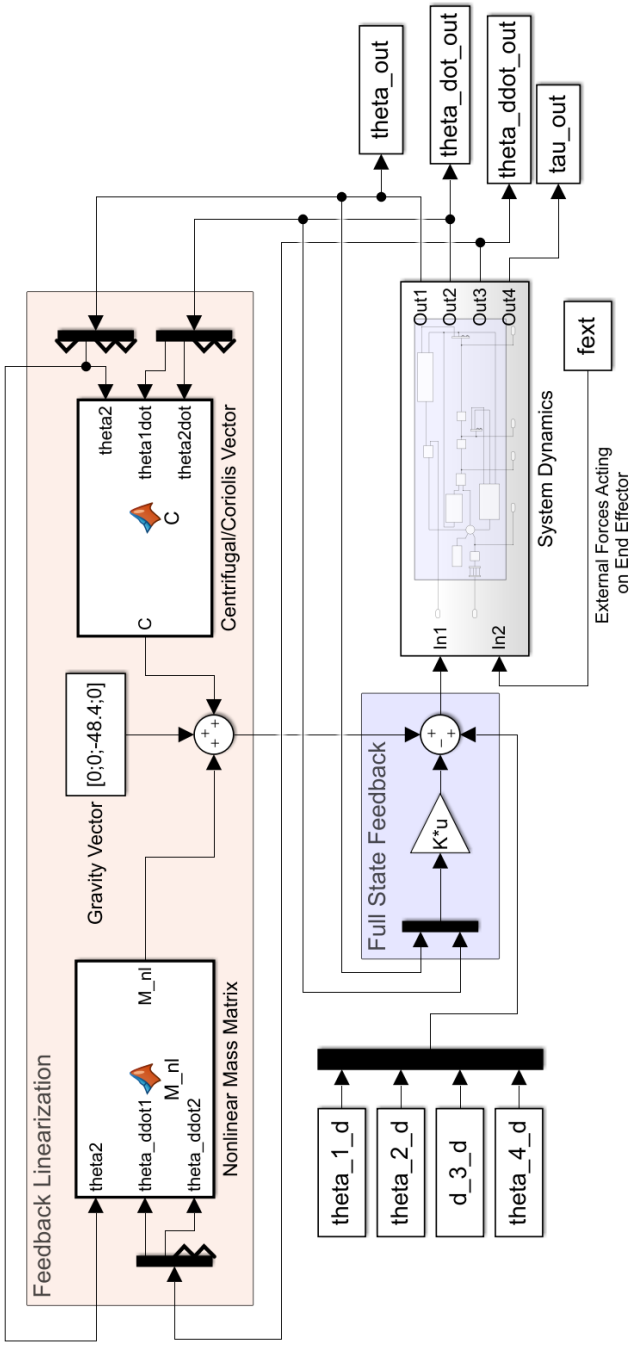

Figure 4.10: Simulink model of Full State Feedback Linearization controlled system

Recalling from Equation 4.17 that d_3 had no dynamic coupling with any of the other three joints (due to it operating purely in the z direction while the other joints operated exclusively on the x-y plane), it did not make sense for the control signal u_3 to have any dependence on θ_1, θ_2, θ_4, $\dot{\theta_1}$, $\dot{\theta_2}$ or $\dot{\theta_4}$. Equation 4.35 showed us that this K matrix did include contributions from these other joints, which although significantly lower in magnitude than the contributions of d_3 and $\dot{d_3}$, were certainly not optimal. This points to potential flaws in MatLab's `place()` function.

When thinking about how to manually determine a better K matrix, the problem became more clear. The K matrix was the matrix which would force the matrix $A - BK$ to have eigenvalues equal to the desired poles - in this case -8.0, -8.1, -8.2, -8.3, -8.4, -8.5, -8.6 and -8.7. To give an idea of the complexity of this task, the matrix $A - BK$ (with significant rounding to allow for it to fit) is shown below:

$$A - BK = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0.5K_{11} & -0.5K_{12} & 0 & 0 & 0.5K_{15} & -0.5K_{16} & 0 & 0 \\ -0.5K_{21} & 1.4K_{22} & 0 & 0.9K_{24} & -0.5K_{25} & 1.4K_{26} & 0 & 0.9K_{28} \\ 0 & 0 & 0.2K_{33} & 0 & 0 & 0 & 0.2K_{37} & 0 \\ 0 & 0.9K_{42} & 0 & 401K_{44} & 0 & 0.9K_{46} & 0 & 401K_{48} \end{bmatrix}$$

The eigenvalues of this matrix were all equal to very long expressions which made it difficult to solve for the individual elements of the K matrix. They were certainly too long to solve for by hand, and it seemed that MatLab's `place()` function used a numerical optimization to do so, so clearly the dimensionality (i.e. $n = 8$) of this optimization was too great for its effective application to this problem. An attempt was made to improve the performance of the system by using desired poles that were 10 times more negative (i.e. more stable), even though this improvement would come at the known cost of significantly better system actuators. The results yielded no improvement.

Another attempt was made to improve the results by using MatLab's `lqr()` function, which employs the Linear Quadratic Regulator to choose an optimal set of poles and a corresponding gain matrix K according to a user-defined set of priorities on error vs. actuation requirement minimization. The gain matrix that it came up with is shown below:

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 & 2.6837 & 0.5078 & 0 & -0.0011 \\ 0 & 1 & 0 & 0 & 0.5078 & 1.7268 & 0 & -0.0016 \\ 0 & 0 & 1 & 0 & 0 & 0 & 3.2987 & 0 \\ 0 & 0 & 0 & 1 & -0.0011 & -0.0016 & 0 & 1.0025 \end{bmatrix} \tag{4.36}$$

This change clearly solved the problem of the motion of Joints 1, 2 and 4 affecting u_3, which was a good sign. When tested, it did make an improvement to the performance of the controller, but not anywhere close to enough to replicate the performance of the PID-based controllers. Figures 4.11 and 4.12 show the performance of the controller using the new LQR-generated gain matrix.

Clearly, the performance was not satisfactory. The controller succeeded in pushing each joint in the correct direction, but was not able to accurately track the reference trajectory anywhere close to as well as the PID-based controllers.

Given that two separate methods of finding an appropriate gain matrix yielded unsatisfactory results, it was clear that either the implementation of the Full State Feedback Linearization controller was done incorrectly, or the it was an inferior controller type for this application. Assuming the latter, one possible explanation for its inferior performance was that it was comparable to a PD controller, not a PID controller. The control law from Equation 4.29 can be rewritten as:

$$u = K_1\theta + K_2\dot{\theta} + \theta_d \tag{4.37}$$

This was not dissimilar to the control law for a Proportional-Derivative (PD) controller, with the gain matrices K_1 and K_2 serving in similar roles to the K_P and K_D gains. Given that the previous controllers were based on a PID action, it made sense that reverting to a PD action (and thus losing the benefits of the integral action) would lead to a decrease in performance. In a PID controller, the integral action is known to reduce tracking error at the expense of stability, while the derivative action is known to increases the damping of the system (i.e. reduce oscillations) at the expense of the tracking error. In a trial such as this where the system was being asked to track a constantly-changing reference trajectory, it made sense that losing the integral action could be responsible for the significant decrease in performance for the Full State Feedback Linearization controller.

While unfortunately the Full State Feedback Linearization controller proved to be a failure, there was a second option available to deal with the coupling in the

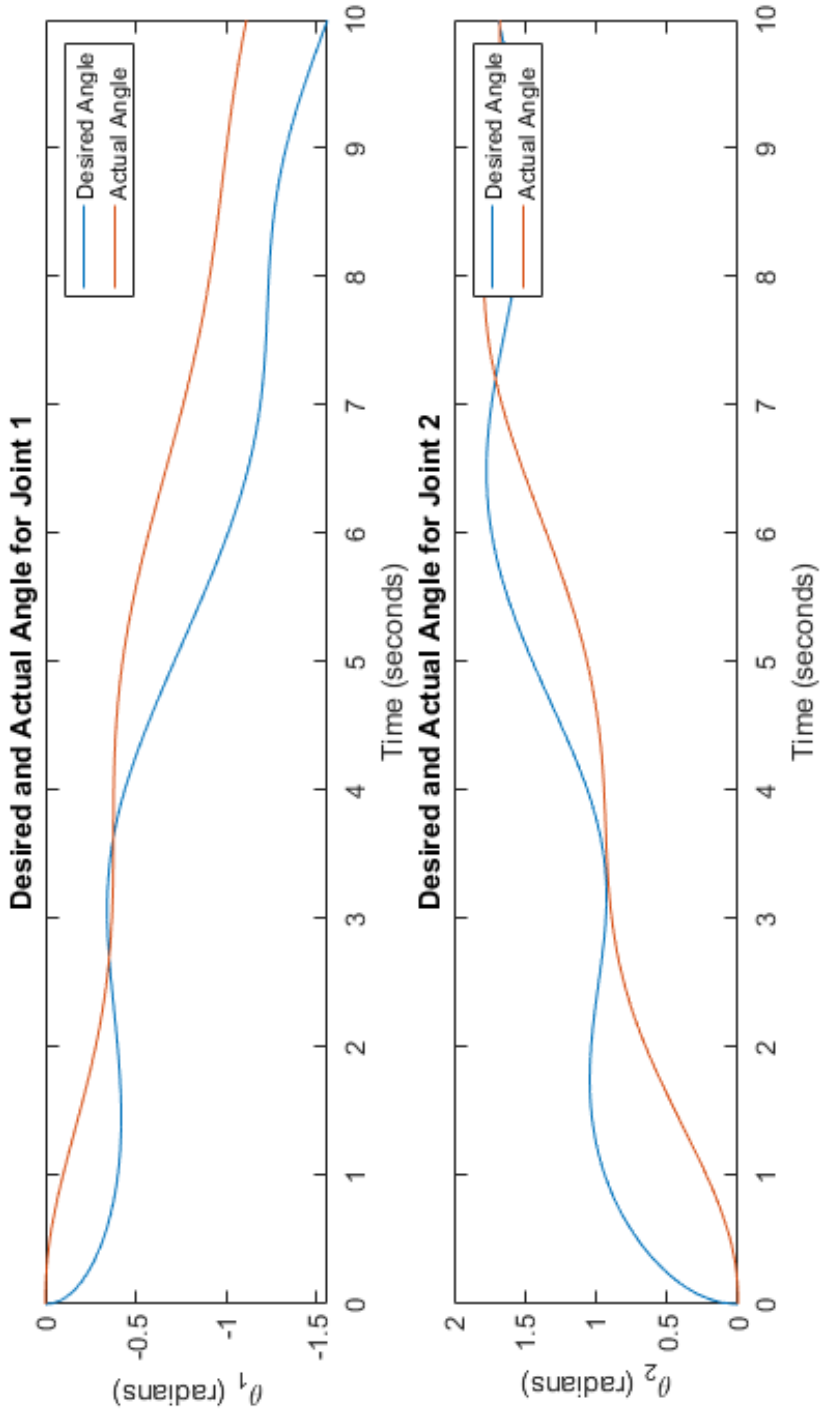

Figure 4.11: Joints 1 and 2 Full State Feedback Linearization controller results using LQR-generated gain matrix

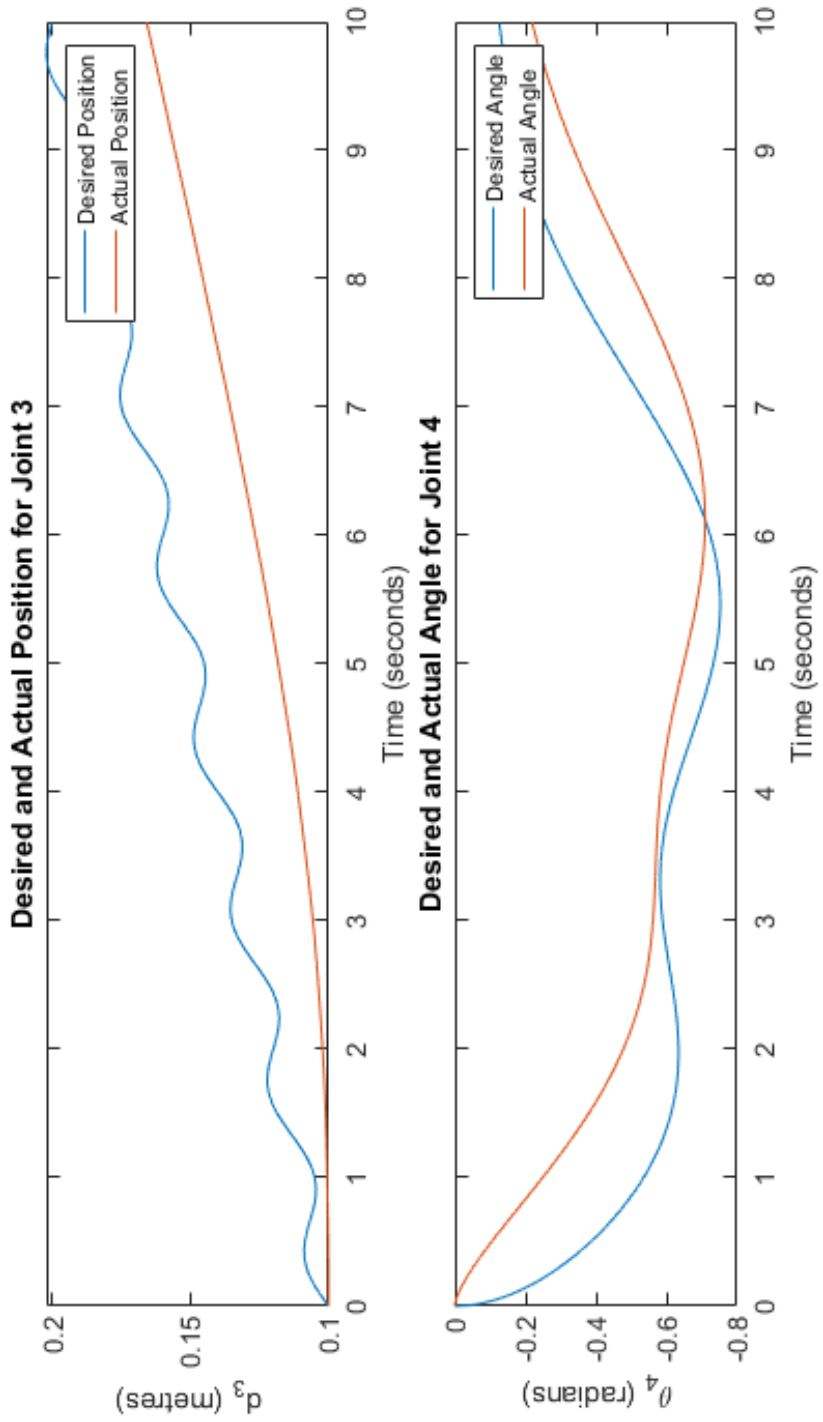

Figure 4.12: Joints 3 and 4 Full State Feedback Linearization controller results using LQR-generated gain matrix

system, and to hopefully create a more centralized control strategy. This option would simply see a transfer of all the non-diagonal terms from the M_{linear} matrix to the $M_{nonlinear}(\theta)$ matrix, and then using the same Feedback Linearization/PID strategy that was previously successful. This way, the coupling would be counteracted with the Feedback Linearization, and the PID controller would be left with (theoretically) linear dynamics. In this strategy, the new breakdown between the two M matrices would be:

$$M_{linear} = \begin{bmatrix} 3.23 & 0 & 0 & 0 \\ 0 & 1.12 & 0 & 0 \\ 0 & 0 & 4.94 & 0 \\ 0 & 0 & 0 & 0.0025 \end{bmatrix} \tag{4.38}$$

$$M_{nonlinear}(\theta) = \begin{bmatrix} 2.34\cos\theta_2 & 1.17\cos\theta_2 + 1.12 & 0 & -0.0025 \\ 1.17\cos\theta_2 + 1.12 & 0 & 0 & -0.0025 \\ 0 & 0 & 0 & 0 \\ -0.0025 & -0.0025 & 0 & 0 \end{bmatrix} \tag{4.39}$$

Plugging these two new matrices into the same control law as shown in Equation 4.24, and also the same Simulink model as shown in Figure 4.8, yielded the errors shown in Figure 4.13. Comparing these results to the Decentralized PID/Feedback Linearization controller, we saw a slight reduction in the tracking error for Joint 4, no change for Joint 3, and slight increases in the tracking error for Joints 1 and 2. Given that the performance of Joints 1 and 2 were more consequential for force position control than the performance of Joint 4 was (they controlled the end effector position and orientation, while Joint 4 only controlled its orientation), it was determined that the Centralized Feedback Linearization/PID Controller actually slightly underperformed its predecessor.

The reasons behind this under-performance likely were that the coupling between the joints was relatively insignificant while operating at low speeds, and because the tracking errors in both controllers were probably largely the result of the non-constant nature of the desired trajectory.

This represented the end of the trials of the different position controllers, and from this point on, the position controller that was used was the Decentralized Feedback Linearization/PID Controller due to its superior performance.

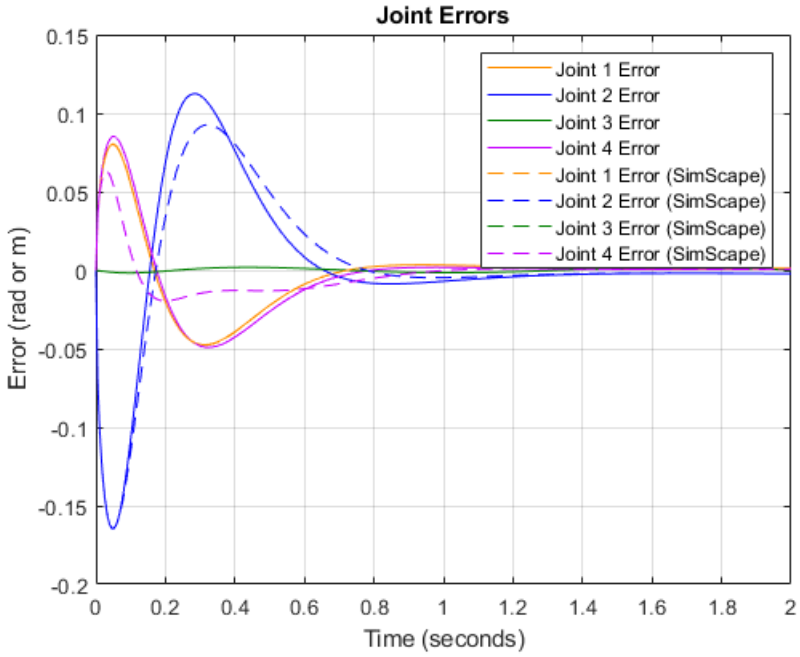

Figure 4.13: Joint errors for Centralized PID/Feedback Linearization controller

4.4 Position Control Workspace Trial

As discussed in Section 3.1.5, there existed three zones within the robot's workspace that were each accessible by a different set of possible robot configurations. One zone was accessible only in the elbow-up configuration (i.e. $\theta_2 > 0$), another was accessible only in the elbow-down configuration (i.e. $\theta_2 < 0$), and the third zone was accessible in either configuration.

This fact presented a problem for the inverse kinematics component of the control algorithm. When given a reference trajectory for the robot to track, the proper elbow configuration had to be selected to ensure continuity from start to finish. Certain trajectories that traversed from the elbow-up-only zone to the elbow-down-only zone would require a configuration switch mid-trial. To solve this issue, the controller analyzed the entire reference trajectory prior to the trial and determined which zones it would need to enter. If the reference trajectory stayed in the zone that allowed both configurations, the elbow-up configuration was arbitrarily selected for the entire

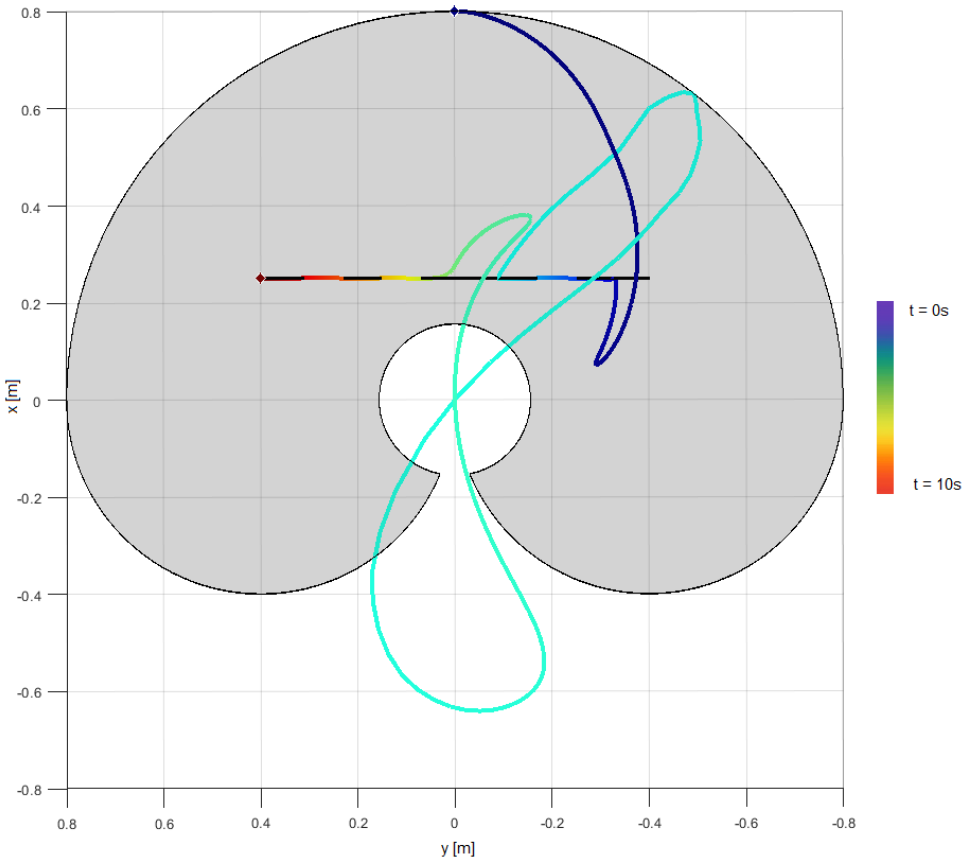

Figure 4.14: Configuration switch trial results

run. If the reference trajectory included travel into one of the areas that required a specific configuration, that configuration would be implemented for the entire run. If the reference trajectory required a configuration switch mid-run, the algorithm would immediately switch the configuration being used for the inverse kinematics and continue.

Figure 4.14 shows the results of a trial where the position controller was provided such a reference trajectory (where the configuration had to be switched mid-run). The black, horizontal line represents the reference trajectory, while the multi-coloured line represents the path of the end effector due to the position controller (starting at dark blue and ending in red). Note that the reference trajectory's starting point was displaced from the home position of the robot, so the trial began with a large,

fast correction move to get it on track. The trial proceeded normally, and then once the robot reached the border between the elbow-up only zone and the zone where both configurations would work (around where the path turned teal), the controller switched to using the elbow-down inverse kinematic equations. This meant that the robot was suddenly very far from its desired position, so it made a rapid, dramatic correction manoeuvre that involved it completely extending Joint 2 (and thus touching the edge of the workspace) and then actually leaving the workspace for a while (the joint limits were not programmed into the simulation). After a short break, the robot was able to successfully continue tracking the reference trajectory.

This trial demonstrated that the robot could successfully switch configurations mid-trial if necessary.

5 Force-Position Control

.

With the position controller fully developed, it was time to proceed to the development of the force-position controller. Before this could happen, however, the model had to be modified in order to make it so that the robot could encounter external reaction forces at its end effector.

5.1 Reaction Surface Model Development

This modification took the form a theoretical reaction surface in the robot's workspace that would exert a normal force against the end effector whenever it made contact with it. For simplicity's sake, the surface was made to be perfectly flat and aligned with the global z axis, and it was also assumed to be frictionless. The only reaction force that it would generate would be a normal force, which would be calculated with the following formula:

$$F_N = k_{rs}(z_{rs} - z_E) \tag{5.1}$$

... where F_N represents the normal reaction force that would be applied to the end effector, k_{rs} represents the stiffness of the reaction surface (set here to 10 kN/m) and z_{rs} represents the z coordinate of the reaction surface. Figure 5.1 shows the Simulink model updated to include the surface reaction forces.

The internal workings of the `reactionforces` function are shown below as well:

```
function reaction_forces = reactionforces(x,y,z,rx,ry,rz)

rsymax = 0.4;
```

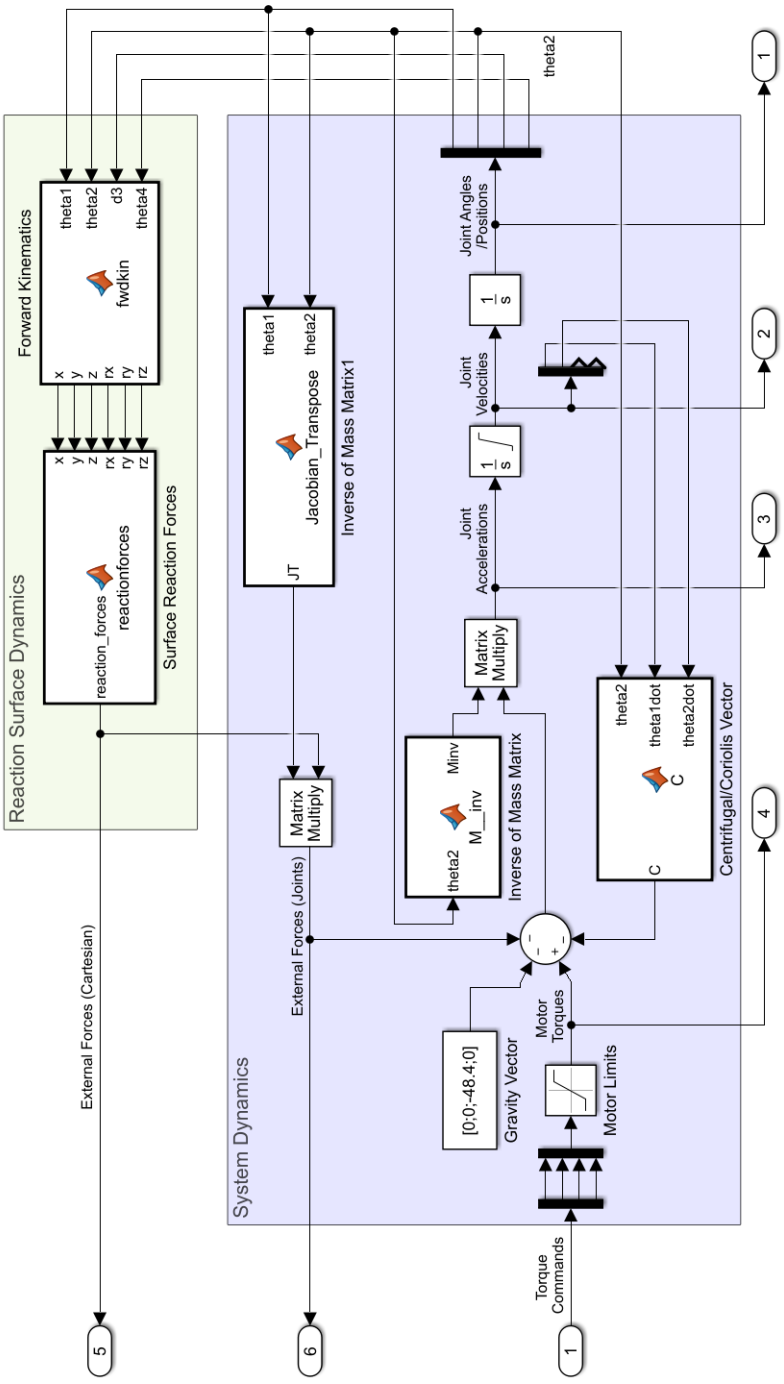

Figure 5.1: Updated Simulink model including reaction surface forces

```
rsymin = -0.4;
rsxmax = 0.6;
rsxmin = 0.2;
rszbase = 0.13;

rsstiffness = 10000;

if x < rsxmin | x > rsxmax | y < rsymin | y > rsymax | z >
    rszbase
     reaction_forces =  [                              0;
                                                       0;
                                                       0;
                                                       0;
                                                       0;
                                                      0];
else
     reaction_forces =  [                              0;
                                                       0;
                              -rsstiffness*(rszbase - z);
                                                       0;
                                                       0;
                                                      0];
end
```

The model first used the x, y and z position of the end effector, and the defined limits of the reaction surface plane (`rsxmax`, `rsxmin`, `rsymax`, `rsymin` and `rszbase`) to determine if the end effector was in contact with the reaction surface or whether it was still in free space. If it was found that it was in contact, the reaction force was calculated using Equation 5.1. As can be seen in Figure 5.1, this reaction force was then multiplied by the transpose of the Jacobian (as per Equation 3.66) and added to the $M(\theta)\ddot{\theta}$, $C(\theta, \dot{\theta})$, and $G(\theta)$ terms already present.

In order to test the functionality of the updated model, a new trajectory was established. This trajectory would start at the robot's home position (where are joint variables were equal to zero), travel in a straight line for 2 seconds to a point on the reaction surface, and then would transition to a sine wave that would repeatedly test the reaction surface. This trajectory is shown below:

$$r(t) = \begin{bmatrix} x_d \\ y_d \\ z_d \\ \psi_d \\ \nu_d \\ \phi_d \end{bmatrix} = \begin{bmatrix} 0.8 - 1.5\sin\frac{t}{T} \\ -1.5\sin\frac{t}{T} \\ 0.25 - (0.25 - z_{rs})\sin\frac{t}{0.2T} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.2}$$

... for $t < 2$ seconds, and to:

$$r(t) = \begin{bmatrix} x_d \\ y_d \\ z_d \\ \psi_d \\ \nu_d \\ \phi_d \end{bmatrix} = \begin{bmatrix} 0.5 \\ -0.3 + 0.75\sin(\frac{t}{T} - 0.2) \\ z_{rs} - 0.025\sin(10\pi(\frac{t}{T} - 0.2)) \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.3}$$

... for $t \geq 2$ seconds. Figure 5.2 below shows the trajectory in black, as well as the reaction surface in yellow.

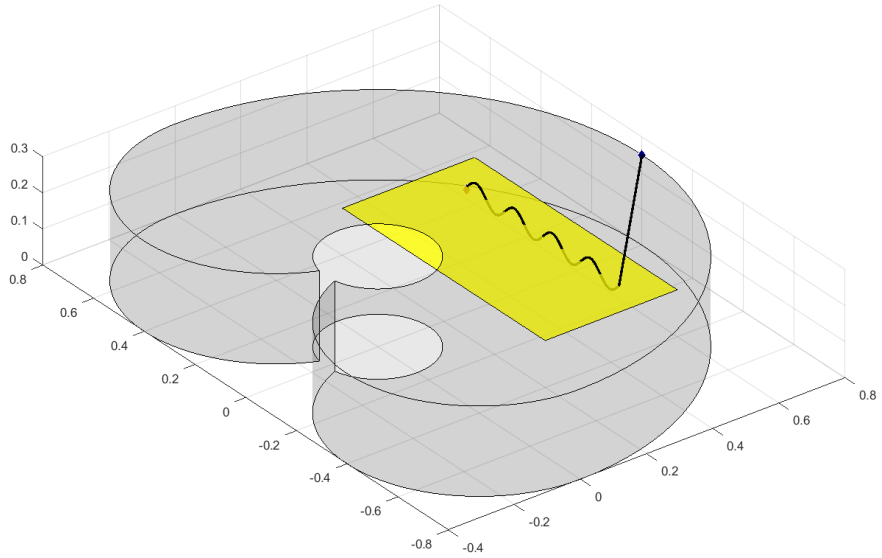

Figure 5.2: Reaction surface testing trajectory

When the position controller that was developed in the previous chapter was used on this updated model and desired trajectory, the results were predictably poor. They are shown below in Figures 5.3 and 5.4. The tracking of the desired trajectory is solid for joints 1, 2 and 4, comparable to the performance obtained in free space. This was because the reaction surface was aligned normal to the global z axis, and the motion of these three joints had no effect on the z position of the end effector (see the third row of Equation 3.37).

However, there was a very serious effect on the position-tracking performance of the third joint, which was fully aligned with the global z axis. The position tracking was good up until the point when it came into contact with the reaction surface, at which point it lost its ability to keep up with the desired trajectory as it descended below the surface. Even once the desired trajectory resurfaced into free space, the effect of the integral action in the position controller kept it from immediately regaining satisfactory tracking. In fact, the controller wasn't able to fully converge before the desired trajectory once again dipped below the reaction surface, causing it to relapse into unsatisfactory tracking. It is worth noting that due to the slight compliance of the reaction surface, the end effector was able to descend a little bit below the reaction surface, but not anywhere close to enough.

These results were exactly what was expected when implementing the position controller on the new model that included the reaction surface. This meant that the new model of the physical system could be considered as verified. With this verification complete, the force control strategy could be developed.

5.2 Hybrid Force-Position Controller

In order to develop and test force control strategies, the hybrid force-position control architecture had to be established first. This was challenging, because it involved building an outer control structure when one of the inner loops (the force controller) had not been developed. Due to the nature of hybrid force-position control, it was possible to build the entire outer loop while maintaining complete dominance of the position controller in the control of the system (using the selection matrix S), which allowed for troubleshooting and ensured functionality prior to the development of the force controller.

The algorithm for robot hybrid force-position control can be described using the following steps:

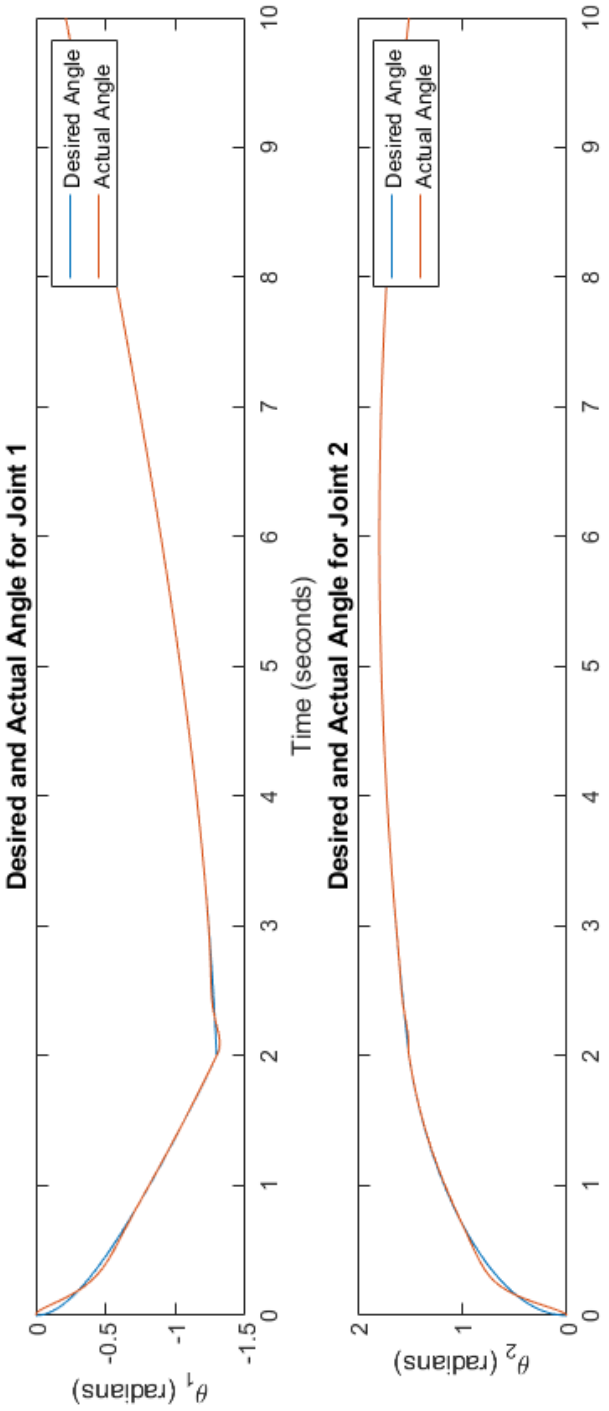

Figure 5.3: Joints 1 and 2 position controller results with reaction surface model

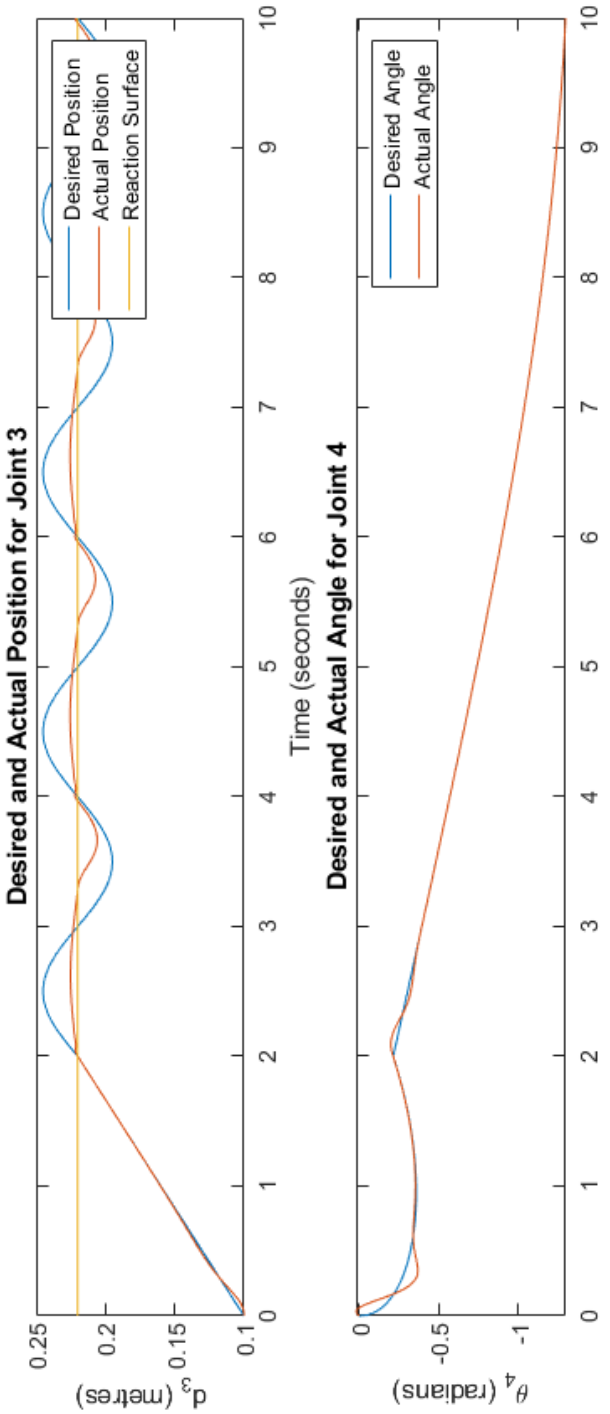

Figure 5.4: Joints 3 and 4 position controller results with reaction surface model

1. A set of desired position trajectories and desired force trajectories are provided as the system's inputs. Typically, both the desired force and position trajectories are expressed in Cartesian coordinates, because the task the robot is assigned to is usually not affected by the configuration of the robot. These trajectories may need to be converted to joint space if the controllers or sensors in the system so demand it.
    a) If the position trajectories need to be converted from Cartesian space to joint space, this can be done using the robot's inverse kinematics.
    b) If the force trajectories need to be converted from Cartesian space to joint space, this can be done by the transpose of the Jacobian matrix.
2. The actual position and force experienced at the end effector are measured using sensors.
    a) Typically, the positions are measured using optical encoders on each joint's motors, and thus are expressed in joint space. A numerical integration can be performed on these positions to obtain joint velocities and then another numerical integration will yield the joint accelerations.
    b) The force experienced at the end effector is usually measured with a force-torque sensor mounted near the end of the robot manipulator. This signal will usually require some extensive processing to remove measurement noise as well as the inertial, centrifugal/Coriolis effects that will come purely from the motion of the robot arm. With these effects removed, the signal representing the external force acting on the end effector may need to be multiplied by the transpose of the appropriate Jacobian matrix to convert it to joint space, if necessary.
3. The measured positions, velocities, accelerations, and external forces are then fed into the two independent position control and force control loops, along with the desired position and force trajectories. Each loop will produce a separate set of control forces/torques (either in joint space or Cartesian space) as their outputs.
    a) The goal of the position control loop is to produce a set of force/torques that will cause the actual position of the end effector to converge on the desired position. It usually will operate without using the force sensor data as an input.
    b) The goal of the force control loop is to produce a set of force/torques that will cause the external force experienced by the end effector to converge

on the desired force. Sometimes it will only use the force sensor data as input, but sometimes it will also use some of the position sensor data.

4. If they are expressed joint space, the output forces/torques from the two control loops must be converted to Cartesian space using the inverse of the transpose of the Jacobian matrix.
5. The force/torque output vector from the position control loop is multiplied by the selection matrix S, while the force/torque output vector from the force control loop is multiplied by the difference between the identity matrix and the selection matrix, $I - S$. These two products are added together to create the unified hybrid force-position control signal.
6. This unified control force/torque signal is then multiplied by the transpose of the Jacobian matrix to convert it back into joint space, and is then sent to the robot's motors to actuate the system.
7. New measurements are taken by the robot's sensors and Steps 2 through 7 repeat themselves.

In designing the hybrid force-position control architecture for this project, two main sets of decisions had to be made. The first surrounded the choice of the selection matrix S, and the second was when to express the signals in joint space vs. Cartesian space.

5.2.1 Establishing the Selection Matrix

To come up with an appropriate selection matrix, the task that would be assigned to the robot had to first be defined. Given that the reaction surface model established in the simulation could only exert a force in the global z direction, the desired force trajectories in the x and y directions, as well as the desired external torques about the x, y and z directions all were set to zero. Knowing that the reaction surface's stiffness was 10 kN/m and that the z height of the entire robot workspace was only 0.25m, it made sense for the desired force trajectory to be set around 200 N in the z direction. Thus, the desired force trajectory would be set to zero in all coordinates except the z direction, where it would be kept around 200 N.

To keep some alignment with the position trajectories established in Equations 5.2 and 5.3, the desired force would be set to zero until the robot had a chance to come into contact with the reaction surface at 2 seconds. However it is important to

note that for $t \geq 2$ seconds, the desired force would be in conflict with the desired position, as the desired position often would surface above the reaction surface, thus allowing no contact force to exist. This conflict between the desired position and the desired force was exactly the reason for which the selection matrix (and in fact the entire hybrid force-position control architecture) was required.

Table 5.1: Natural and artificial constraints for 4 DOF SCARA robot

Natural Constraints	**Artificial Constraints**
$f_x = 0$	$x = x_d$
$f_y = 0$	$y = y_d$
$\tau_z = 0$	$\phi = \phi_d$
$z = z_{rs} - \frac{f_{ext,z,d}}{k_{rs}}$	$f_z = f_{ext,z,d}$
$\tau_x = 0$	
$\tau_y = 0$	
$\psi = 0$	
$\nu = 0$	

To decide on how to make the selection matrix, the task was broken down into natural and artificial constraints, shown in Table 5.1. All twelve entries in this table should be discussed, as they all held some significance.

The simplest constraints to discuss are the two natural constraints of $\psi = 0$ and $\nu = 0$. These constraints were imposed on the robot simply due to its number of degrees of freedom. It was simply not possible for a 4 DOF SCARA robot configured as in this project to achieve any rotation of the end effector about the global x or y axes, thus ψ and ν by definition had to always be zero.

Next, we could establish the natural constraints of $f_x = 0$, $f_y = 0$, $\tau_x = 0$, $\tau_y = 0$ and $\tau_z = 0$. These external forces acting on the end effector had to always be zero simply because the reaction surface defined in this model had no ability to exert external forces or torques other than a reaction force in the global z direction. If the model included non-normal forces such as friction, or if the reaction surface was aligned on an oblique plane, or if the surface was curved, these constraints would not necessarily be true. However, with this model, they did hold true.

Next we can discuss the artificial constraints of $x = x_d$, $y = y_d$ and $\phi = \phi_d$. These constraints were called "artificial" because they were imposed via the position controller. Because there was no possibility of an external force being applied in the x or y directions, nor an external torque about the z axis, the position controller would

be free to constrain the motion of the end effector in these three degrees of freedom as if the motion were through free space.

Finally, we must discuss the final two constraints - the artificial constraint of $f_z = f_{ext,z,d}$ and its twin, the natural constraint of $z = z_{rs} - \frac{f_{ext,z,d}}{k_{rs}}$. These two constraints were the two that were acting in the all-important z direction, where the reaction surface would provide an impediment to position control as shown in Figure **??**. Because position control would not be effective in this direction, we could not use the position controller to establish an artificial constraint on the z position of the end effector. Additionally, we had a desired force trajectory that we were aiming to track in the z direction of $f_{ext,z,d}$, which meant that $f_z = f_{ext,z,d}$ should be defined as the artificial constraint. The natural constraint that would arise from Equation 5.1 would thus be $z = z_{rs} - \frac{f_{ext,z,d}}{k_{rs}}$. In defining the natural and artificial constraints this way, we were effectively telling the hybrid force-position controller to prioritize force control in the z direction and to accept that the z position of the end effector would be determined by the dynamics of the reaction surface, not by the desired position trajectory.

From these natural and artificial constraints, we could create the selection matrix S. This matrix was a 6×6 diagonal matrix where all of the diagonal elements s_i were between 0 and 1 (i.e. $S = \text{diag}(s_1, s_2, s_3, s_4, s_5, s_6)$ such that all $s_i \in (0, 1)$). The control law wherein the selection matrix is used is shown below:

$$\tau = J^T\big([S]\tau_p + [I - S]\tau_f\big) \tag{5.4}$$

... where I represents a 6×6 identity matrix, and τ_p and τ_f represent the actuating forces/torques generated by the position and force control loops, respectively. Note that τ_p and τ_f had to be expressed in the Cartesian base frame, while the unified actuating force/torque vector that was sent to the system as input, τ, was expressed in joint space.

The S matrix had to select the directions and axes of rotation which would be artificially constrained by the position controller, and had to leave unselected the directions and axes of rotations which would be artificially constrained by the force controller. In our example, since the x and y axes of rotation would not be artificially constrained by the position controller nor the force controller, it did not matter what the s_4 and s_5 entries were. Because we were asking the controller to completely prioritize position control for the first two seconds of the trial, the selection matrix for the first 2 seconds of the trial was set to:

$$S_{0-2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

... while the selection matrix from 2 seconds onward was set to:

$$S_{2-10} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

This simple deletion of one element of the selection matrix would ultimately lead to the prioritization of force control in the z direction. If we wanted the controller to slowly transition to force control from position control, we could have established a time variant S matrix where the s_3 element would gradually change from 0 to 1 over time. If we wanted to have the controller compromise between both force and position tracking error at the same time, we could have set s_3 to a value somewhere between 0 and 1 and held it there. Finally, if the task frame (and thus the natural and artificial constraints) were expressed in a frame of reference that was the same as the global frame of reference, or was time variant, the force/torque vectors τ_p and τ_f would have had to be multiplied by an appropriate rotation matrix to account for this. Additionally the entire right hand side of Equation 5.4 would have had to be multiplied by the inverse of this rotation matrix to convert it back to the global reference frame prior to multiplication by the transpose of the Jacobian to convert it into joint space. For this application, however, there was no need to make such modifications.

5.2.2 Joint Space/Cartesian Space Signal Assignments

With the selection matrix established, the remaining decisions in the development of the hybrid force-position control architecture involved deciding which signals would

be expressed in Cartesian coordinates, and which would be expressed in joint space. When reading through the seven-step description of the algorithm at the beginning of this section, there are many points where it was mentioned that a signal could be expressed in either Cartesian or joint space. The conversion between the two involves using the forward or inverse kinematics when discussing positions, multiplication by the Jacobian or by the inverse of the Jacobian when discussing velocities, and multiplying by the transpose of the Jacobian or the inverse of the transpose of the Jacobian when discussing forces. Each one of these conversions comes at a computational cost, and thus conversions were minimized as much as possible.

To make the decisions on where to make the conversions, it was important to first identify certain points in the overall algorithm where the signal **had to** be in one of the two forms (Cartesian space or joint space). As recently discussed, we knew that the actuating torque outputs from each of the position and force control loops **had to** be expressed in Cartesian coordinates in order for them to be used in the selection matrix control law (Equation 5.4). We also knew that after being processed by the selection matrix, the signal **had to** be converted back into joint space in order to be sent to each of the joint's motors. Thus, there had to be at least one multiplication by the transpose of the Jacobian.

Next we had to consider the position controller that has already been developed in the previous chapter. This position controller worked exclusively in joint space, using inverse kinematics to pre-compute the joint space equivalent of the desired position trajectory, which was provided in Cartesian coordinates (Equations 5.2 and 5.3). When the position controller was being used on its own to control the robot, it was convenient for the actuating force/torque vector to be expressed in joint space so that it could be directly sent to each of the robot's motors. Unfortunately, this meant that the position controller's output had to be multiplied by the inverse of the transpose of the Jacobian in order to be converted to Cartesian space prior to being sent to the selection matrix. This was a relatively significant issue, because the transpose of the Jacobian was not always invertible.

The first issue with the invertibility of the Jacobian matrix was a result of it being a 6×4 matrix. Any non-square matrix is, by definition, not invertible. Luckily, because two of the rows in the Jacobian were all zeroes (because of the inaccessibility of the rotational degrees of freedom about the x and y axes), the Jacobian could easily be made square by simply removing the two empty rows. The second issue was that even once the Jacobian was made square, it could still become uninvertible when its determinant was equal to zero. From Section 3.2.2, we recall that this condition arose

when the robot was in a singular configuration (when $\theta_2 = 0$ or π). This was more challenging to deal with, but could be avoided through some clever programming. The problem that would arise when the determinant was equal to zero was that in calculating the inverse of the matrix, there would be a requirement to divide by zero, thus breaking the control loop. There existed no issue if we were dividing by 0.0001 or by -0.0001, so we could monitor the value of θ_2 being used in the calculation, and if it ever went between -0.0001 and 0.0001, we could just manually adjust it to -0.0001 or 0.0001, whichever was closer. This would lead to a slight inaccuracy in the conversion from joint space to Cartesian space, but as long as the manipulator did not stay near the kinematic singularity for too long the inaccuracy would have a negligible effect. Once these two problems were taken care of in the manner outlined above, the inverse of the transpose of Jacobian could always be taken without risking a mathematical error. Once the inverse was taken, the two rows of zeros were added back in the appropriate places to yield the following matrix:

$$(J^T)^{-1} = \begin{bmatrix} 2.5\frac{\cos(\theta_1+\theta_2)}{\sin\theta_2} & -2.5\frac{\cos(\theta_1+\theta_2)+\cos\theta_1}{\sin\theta_2} & 0 & -2.5\frac{\cos\theta_1}{\sin\theta_2} \\ 2.5\frac{\sin(\theta_1+\theta_2)}{\sin\theta_2} & -2.5\frac{\sin(\theta_1+\theta_2)+\sin\theta_1}{\sin\theta_2} & 0 & -2.5\frac{\sin\theta_1}{\sin\theta_2} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{5.7}$$

With the inverse of the transpose of the Jacobian complete, the only remaining conversions that were necessary were for the force control loop. Luckily, because the force controller would be used only in the context of a hybrid force-position controller, it could be set up to require no conversions. The desired force trajectories were easiest to provide in Cartesian space - the force/torque measurements in our simulation could be easily accessed (without having to process force/torque sensor data) in Cartesian space so there was never a need to convert into joint space. The entire force control loop was thus be built in Cartesian space and then the loop output τ_f would require no conversion prior to being sent to the selection matrix.

A block diagram showing the different signals as they flowed through the hybrid force-position controller architecture is shown in Figure 5.5. Note that signals with a j subscript referred to signals in joint space, and those with C subscripts referred to signals in Cartesian space.

The Simulink implementation of the entire hybrid force-position control scheme

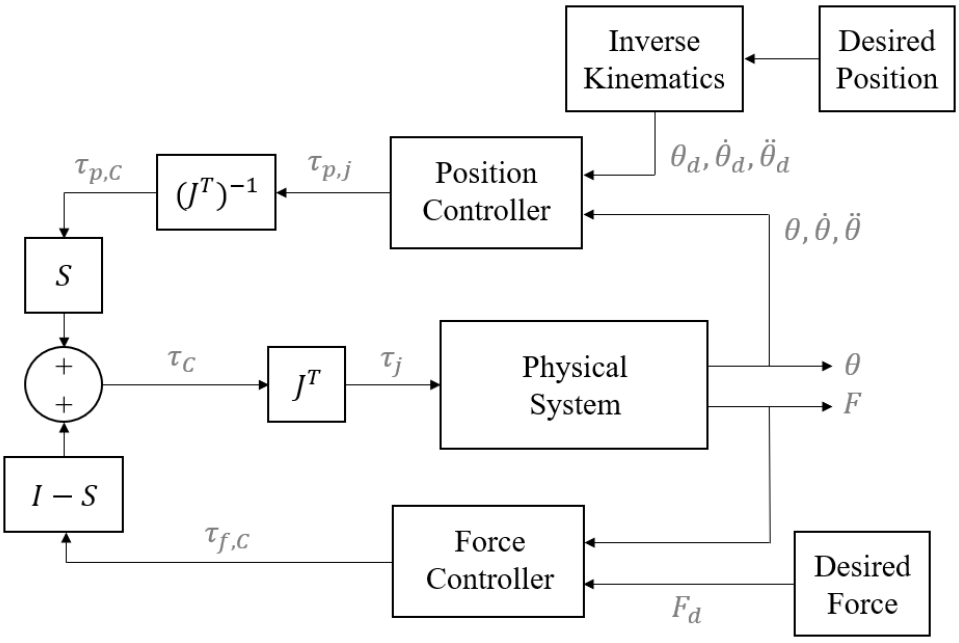

Figure 5.5: Block diagram of hybrid force-position controller architecture

is shown in Figure 5.6. The supervisory controller is blown up to show it in further detail. Note that the image also shows the force controller, despite their having been no discussion of its development as of yet. This discussion will follow in Section 5.3.

The MatLab Function block labelled `S` is the selection matrix. The function inside of it is as follows:

```
function [S,IS] = S(u)

if u>0
    S = [1 0 0 0 0 0;0 1 0 0 0 0;0 0 0 0 0 0;0 0 0 1 0 0;0 0
        0 0 1 0; 0 0 0 0 0 1];
else
    S = eye(6);
end

IS = eye(6) - S;
```

In Figure 5.6 it can be seen that the input "u" to the "S" function was a step

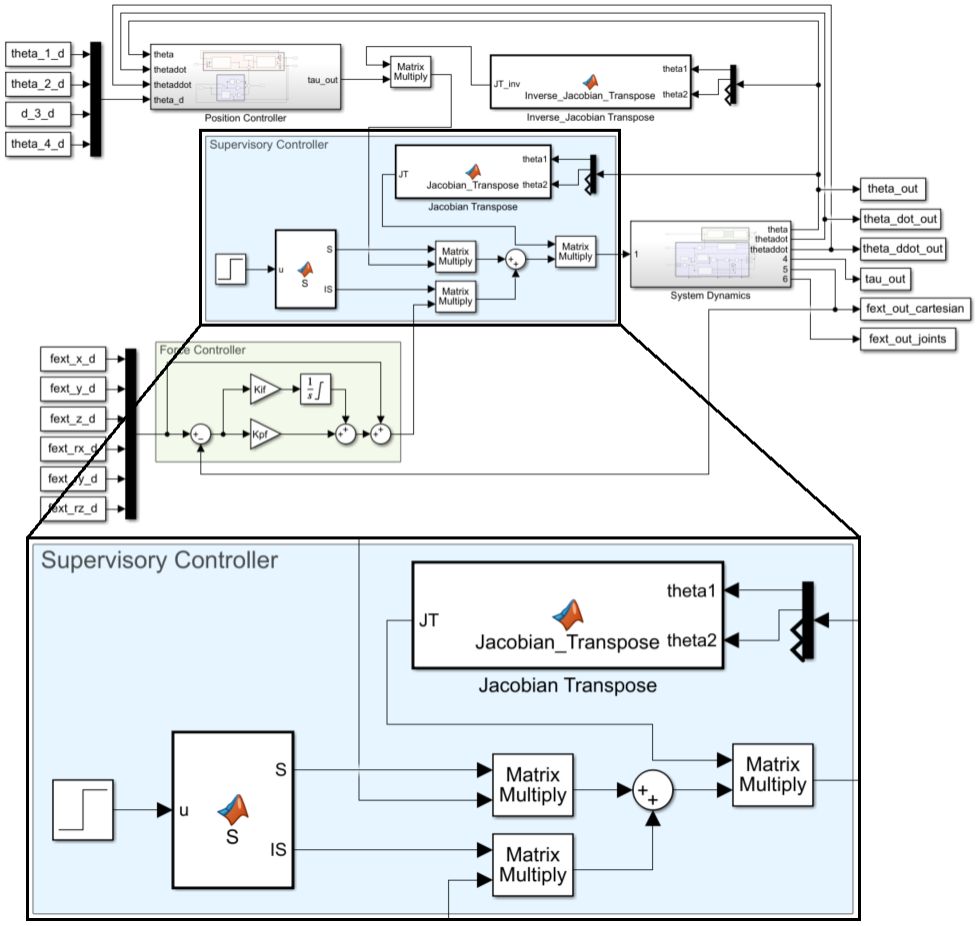

Figure 5.6: Simulink implementation of hybrid force-position controller

input. This step input was set to zero for $0 \leq t < 2$ and switched to 1 for $t \geq 2$. It functioned as a trigger to force the S matrix to change from pure position control dominance, to the hybrid force-position control structure detailed above.

Note also in Figure 5.6 the two other MatLab function blocks. The one that is blown up in the magnified area is the transpose of the Jacobian. Unfortunately, the top right of the two is not blown up for better visibility, but it is the inverse of the transpose of the Jacobian matrix. These two functions were the minimum requirement for Cartesian to joint space (and vice versa) signal conversion as discussed above, and they were placed in the scheme in the locations specified in Figure 5.5.

5.3 Force Control

The control of the external force applied by the end effector was a very different problem compared to control of the position of the end effector. Firstly, if a hybrid force-position control strategy is being used, it is likely because there is some degree of uncertainty about the dynamics of the environment that will be generating the contact forces (otherwise impedance control could easily be implemented). This means that the force controller can not be designed using control laws derived from the system's dynamic equations to the same degree that a position controller can. Secondly, force and position are very different concepts theoretically, in that they are separated by two orders of integration and have completely different relationships with other relevant variables such as velocity, acceleration and mass. Finally, the sensors that can be used to measure force produce much noisier signals compared to position sensors (although in a simulated environment this is less important).

The first issue is the most important to explain, as it was the cause of significant confusion during the force controller development, and ultimately determined the design process that was used. Originally it was assumed that to develop the force controller, a similar process to the position controller development should be used. The dynamic equations would be written out, and then manipulated until a control law could be arrived at. That control law would produce a control force/torque vector which would be mathematically guaranteed to achieve convergence between the desired contact force and the actual contact force.

Unfortunately, that process, when applied to the force control problem, would require precise knowledge of the dynamics of the reaction surface. While models for such dynamics were available, one of the main reasons hybrid force-position control is used is for situations when knowledge of the dynamics of the environment are

unknown, or too complex to mathematically model accurately. When the dynamics of the environment are known, it is simple enough to build an impedance controller that will convert the desired force into a desired "virtual position" that the position controller can track on its own. To build the force controller in a hybrid force-position control scheme using the known dynamic equations of the reaction surface would be to subvert the purpose of this project. Thus, the force controller had to be built without consideration of the dynamics of the environment in which it would be operating.

The implications of this reality were that there could not be a guarantee of convergence for the force controller in the same way that there was for the position controller. Additionally, any control gains that would be used could not be calculated based on theoretical knowledge of the dynamic equations, they would generally have to be found through a process of trial and error. It is possible that there exists a better way to come up with a control law and controller gains without information on the dynamics of the environment, but throughout the survey of the literature it was never discovered. This certainly could represent a future improvement on the work done in this thesis.

The second major difference between the development of the force controller in comparison to the position controller was the simply fact that forces and torques are mathematically much different than positions and angles. In the position controller, the robot could be thought of as a system that took a force/torque vector as input, and produced a linear/angular position vector as output. The nonlinear dynamics of the plant could be nullified with feedback linearization, leaving a clean second-order linear system (Newton's Second Law states that force/torque is proportional to the second derivative of linear/angular position) that could be controlled using established linear control methods such as PID control. For force control, the position-related terms ($M(\theta)$, $C(\theta, \dot{\theta})$ and $G(\theta)$) of the dynamic equation (Equation 3.66) could be cancelled using a process similar to feedback linearization, but what was left (shown below) was not a second-order differential equation:

$$\tau_{Cartesian} = f_{ext} \tag{5.8}$$

The simplicity of this dynamic equation suggested that the controller be built using **a feedforward control law** similar to the law shown below:

$$\tau_{Cartesian} = f_d \tag{5.9}$$

... where f_d represents the desired force trajectory expressed in Cartesian coordinates. This simple control law of course would not be enough to achieve satisfactory force tracking, but it would at least need to form the basis of the control law.

The final important distinction between force control and position control lay in the difference between the measurement hardware for the two feedback loops. In position control, the sensors used to generate the feedback for the controller are typically optical encoders on each of the motors. These encoders can very accurately measure the position of the joint, without significant measurement noise. In force control, the sensor providing the feedback is usually a force/torque sensor mounted on the end effector. This sensor works through the use of a strain gauge, which produces a very noisy signal. Filtering mechanisms do exist that can reduce the noise, but these come with a computational cost and also can muffle important feedback information.

The existence of this strain gauge measurement noise creates a significant challenge when taking a numerical derivative. There are many different ways to take a numerical derivative, but at their core they all must take the difference between previous measurements and divide it by the time interval between them. Because taking a larger time interval will create a larger lag in the measurements with respect to the current signal, an optimal numerical derivative is one that uses as small of a time interval as possible. When a numerical derivative is taken of a noisy signal, especially when the time interval is small, there is a very large chance that the difference between the two sampled measurements will not be a reflection of the overall trend in the rate of change of the signal. This means that any measurement noise will get amplified when a numerical derivative is taken. If a second numerical derivative must be taken, the measurement noise will get amplified even further, often to the point where the signal is no longer present amid all the noise.

The implication of this is that it is feasible to take the numerical derivative of a signal that comes from an optical encoder much moreso than it is to take one from a signal coming from a strain gauge. In the previous chapter, PID control with feedback linearization was used to control the position of the robot. This involved taking a numerical derivative of the joint angles/positions both in the feedback linearization and in the derivative part of the PID controller. Additionally, the signal had to undergo a second numerical derivative for use in the cancellation of the coupled terms of the M matrix of the feedback linearization. Unfortunately in the force control scheme, due to the expected noisiness of the sensor data, it was best to avoid the use of numerical derivatives.

It is important to note that because this controller was being developed using

a simulation where there was no measurement noise, it was functionally possible to take a numerical derivative of the feedback. This avoidance of numerical derivatives was a self-imposed constraint designed to make the developed force controller more applicable to reality.

Without the ability to take a numerical derivative, there was no way to implement the derivative component of a PID controller. This was unfortunate, due to the flexibility and simplicity of PID control. However, the benefits of PID control did not all have to be discarded, because the P (proportional) and I (integral) terms do not require numerical derivatives. Thus, a **PI controller** was thought of as an appropriate choice for the force control.

5.3.1 PI/Feedforward Controller

The two deductions from the above analysis on the differences between position and force control were that both feedforward control and PI control would be appropriate for the force controller. Thus, the first force control law that was proposed was:

$$\tau = f_d + K_{fp}(f_d - f) + K_{fi}\int_0^t (f_d - f)dt \tag{5.10}$$

... where τ represents the control torque (in Cartesian space), f_d and f represent the desired and actual external forces, respectively, and K_{fp} and K_{fi} represent the proportional and integral force control gains, respectively. The Simulink implementation of this control law is shown in Figure 5.7.

As mentioned above, no clear method was found to select the control gains K_{fp} and K_{fi}, so they were selected through trial and error. For this robot, this was relatively simple, in that the force control only had to operate through the one joint that was aligned with the z axis. This would not be so simple under many other circumstances, including:

- If the reaction surface generated forces that did not act only in the normal direction
- If the reaction surface was not aligned perfectly with the $x - y$ plane
- If the reaction surface was not perfectly flat
- If the robot manipulator had more than one joint that could generate forces in the z direction

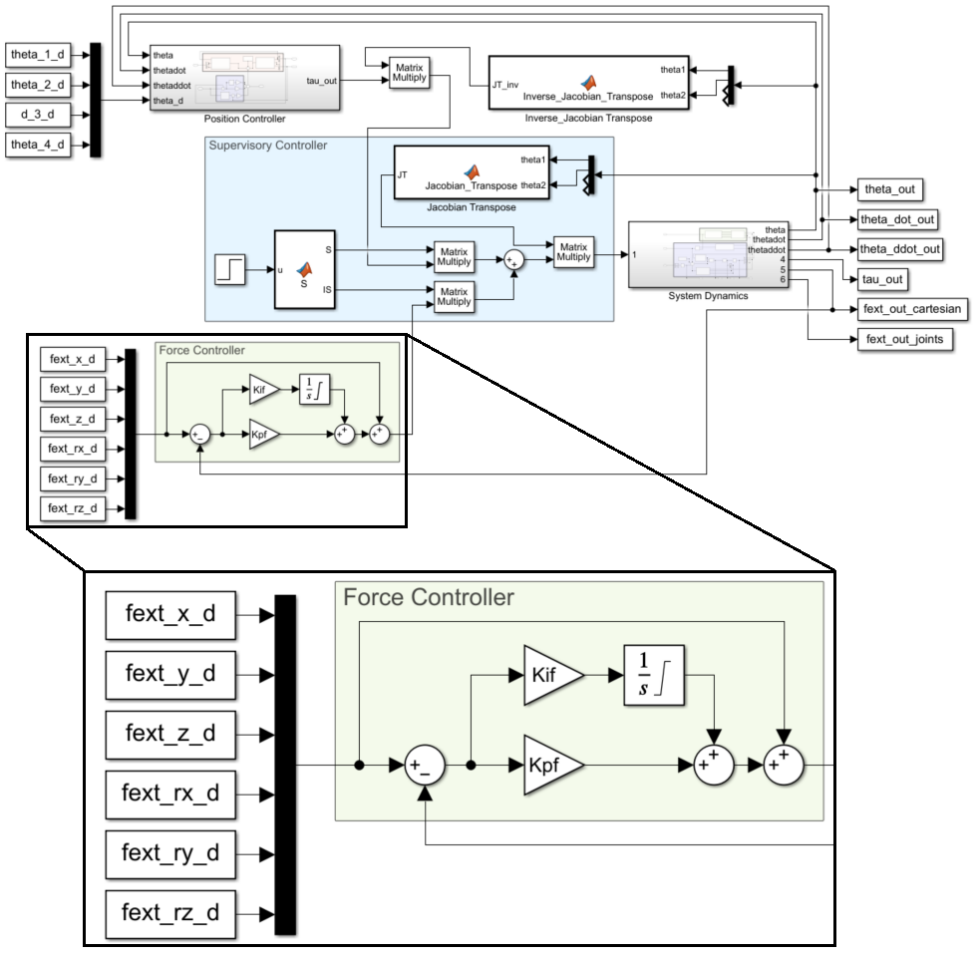

Figure 5.7: Simulink implementation of PI/Feedforward force controller

- If the robot manipulator did have only one joint that could generate forces in the z direction, but that joint was able to generate forces in the x and/or y directions as well
- If the selection matrix required force control in more directions than just z

In all of these cases, different control gains would be required for each of the 6 Cartesian degrees of freedom. This would be much more difficult to set up using trial and error alone, although not necessarily impossible.

To perform the trial and error search, the controller was first run with both gains set to zero. The feedforward controller alone created a tracking response, but only one that oscillated significantly about the desired value. Next, the value of K_{fp} was increased from zero in small increments until the point when the increases no longer corresponded to an improvement in the controller's performance. It was found that this value was somewhere around 20. At this point, the robot could track the desired force, although with a significant oscillatory response following the transition from position control to force control at 2 seconds. Additionally, there was a slight steady-state error in the response. The K_{fi} term was then increased until the steady-state error disappeared, at a value of around 5. These gains were accepted as somewhere close to optimal, as a better response could not be obtained with any different set of gains. The results of the PI/Feedforward controller implemented with these gains are shown in Figure 5.8.

These results were promising, in that they represented a functional hybrid force-position control scheme. The position controller was able to reliably track the desired trajectory for the first two seconds, and then when the selection matrix dictated that the force controller was to take over in the z direction, it did so. The position tracking in the x and y directions were not compromised, while the position control in the z direction was abandoned in favour of the force controller (as intended).

Where these results fell short was in the quality of the force controller's performance. Not only did it take a long time for the controller to converge on the desired value (roughly one second), it did so with a very large overshoot (almost 100%) and some serious oscillation in the response.

The next iteration of the force controller was designed to improve this performance.

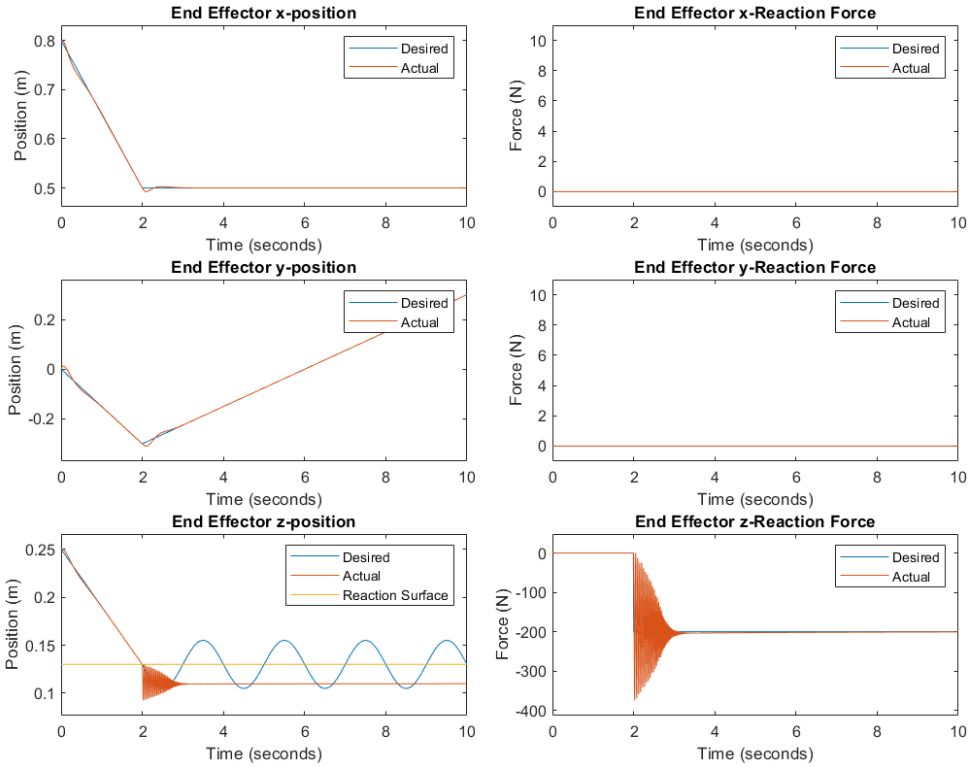

Figure 5.8: Results of the PI/Feedforward force controller

5.3.2 PI/Feedforward/Velocity Damping Controller

The issues with the response of the PI/Feedforward force controller were a significant overshoot, a slow settling time, and a significantly oscillatory response. In a normal PID scheme, these issues are taken care of with the derivative term, however in this project it was decided that derivative control would be avoided for the force controller due to issues with taking a numerical derivative of noisy force/torque sensor signals.

A typical derivative term in a PID controller comes in the form $K_D(\dot{x_d} - \dot{x})$, where K_D is the derivative gain, x_d is the desired system output, and x is the actual measured output. Thus, the command signal sent to the system's actuators has a negative-proportional relationship with the time derivative of the output of the system. Thus, the derivative term can be thought of as a sort of "artificial damping" of the system, removing kinetic energy when the system becomes over-excited. This removal of energy during peak excitation of the system has a strong stabilizing effect

on the controller's performance which can reduce/eliminate overshoot and oscillatory behaviour.

Thus, a clever substitute for a derivative control term was used. Instead of using the derivative of the noisy force/torque sensor data, a derivative of the position of the end effector was fed into the force controller. As previously stated, there was no problem with taking a numerical derivative of the position sensor data, because it contained very little measurement noise. As long as the control law ensured an negative-proportional relationship between the velocity of the end effector and the command signal, the "artificial damping" effect could be replicated. Thus, a "velocity damping" term was added to the end of the PI/Feedforward control law, as shown below in Equation 5.11:

$$\tau = f_d + K_{fp}(f_d - f) + K_{fi}\int_0^t (f_d - f)dt - K_{vd}\dot{X} \tag{5.11}$$

... where K_{vd} represents the velocity damping gain, and $\dot{X}$ represents the Cartesian expression of the end-effector's velocity:

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\psi} \\ \dot{\nu} \\ \dot{\phi} \end{bmatrix} = J(\theta)\dot{\theta} \tag{5.12}$$

The Simulink implementation of this control law is shown in Figure 5.9.

The same process was used to determine the appropriate value for K_{vd} as was used for the other two force control gains - trial and error. The value was raised in small increments from zero all the way until the increases no longer had a positive impact on the performance of the controller. The optimal value was found to be approximately 2000. The performance of the controller with K_{vd} set to this optimal value are shown in Figure 5.10.

These results were thought to be excellent. As with the PI/Feedforward controller, the position tracking was excellent up until $t = 2$ seconds in the x, y and z directions. For $t \geq 2$ seconds, the x and y position tracking continued to be excellent, while

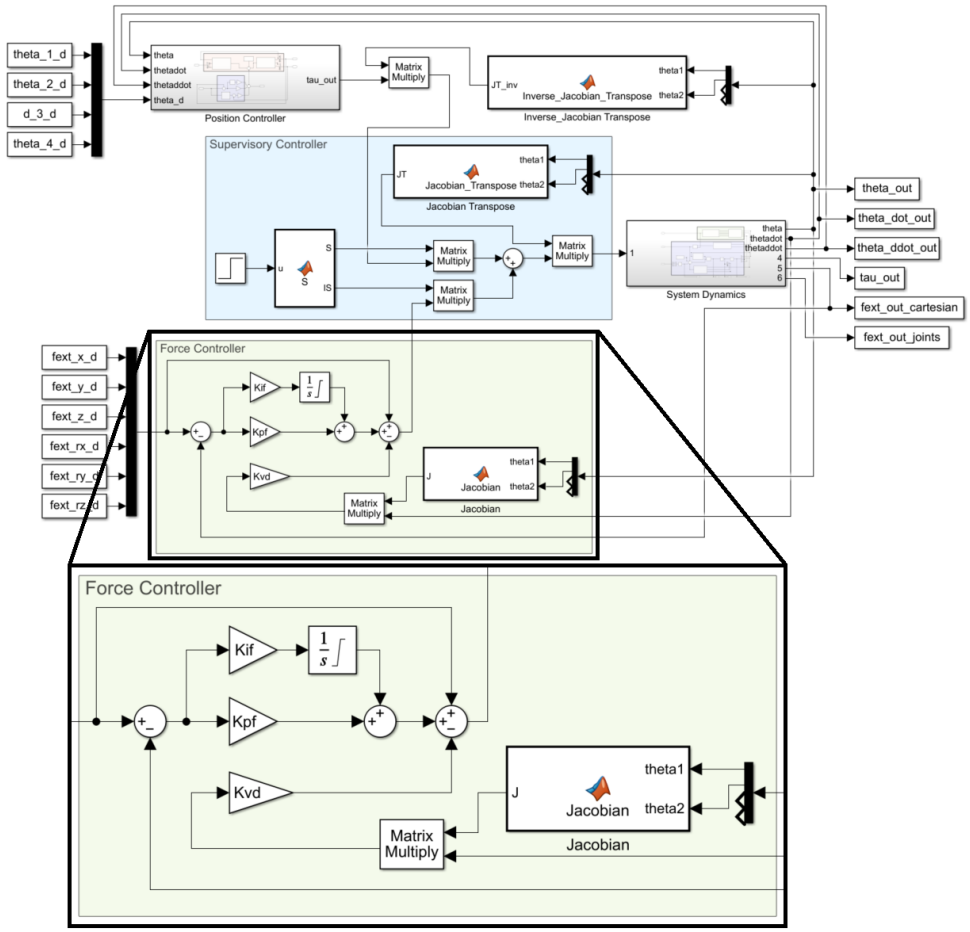

Figure 5.9: Simulink implementation of PI/Feedforward/Velocity Damping force controller

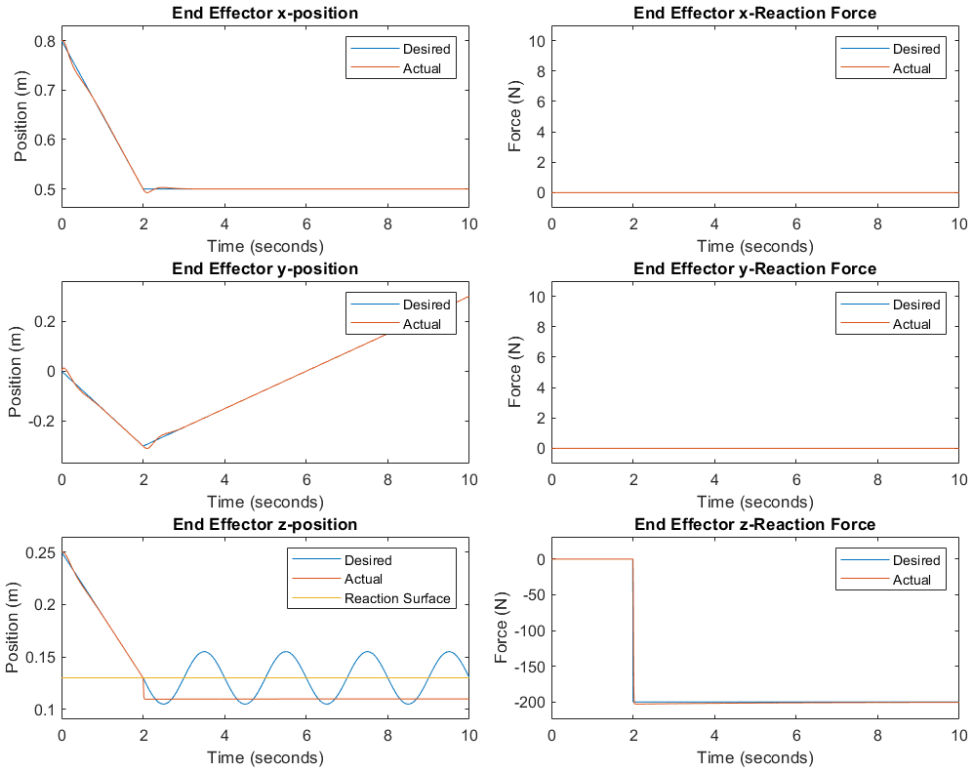

Figure 5.10: Results of the PI/Feedforward/Velocity Damping force controller

the position tracking in the z direction was ignored in favour of the force tracking. The performance of the force controller represented the major improvement against the previous controller, as the overshoot and settling time were dramatically reduced, while the oscillation was completely eliminated. This meant that the velocity damping term was working exactly as was intended.

This performance was considered to be fully satisfactory in terms of the aims of this project, in that the goal of hybrid force-position control had been achieved. To further test the capabilities of the controller, another trial was run, except that instead of asking the force controller to track a constant 200N desired force in the z direction, it was asked to track a z-force trajectory of:

$$f_z = 200\big(1 + \sin\big(\frac{10t}{T} - 0.2\big)\big) \tag{5.13}$$

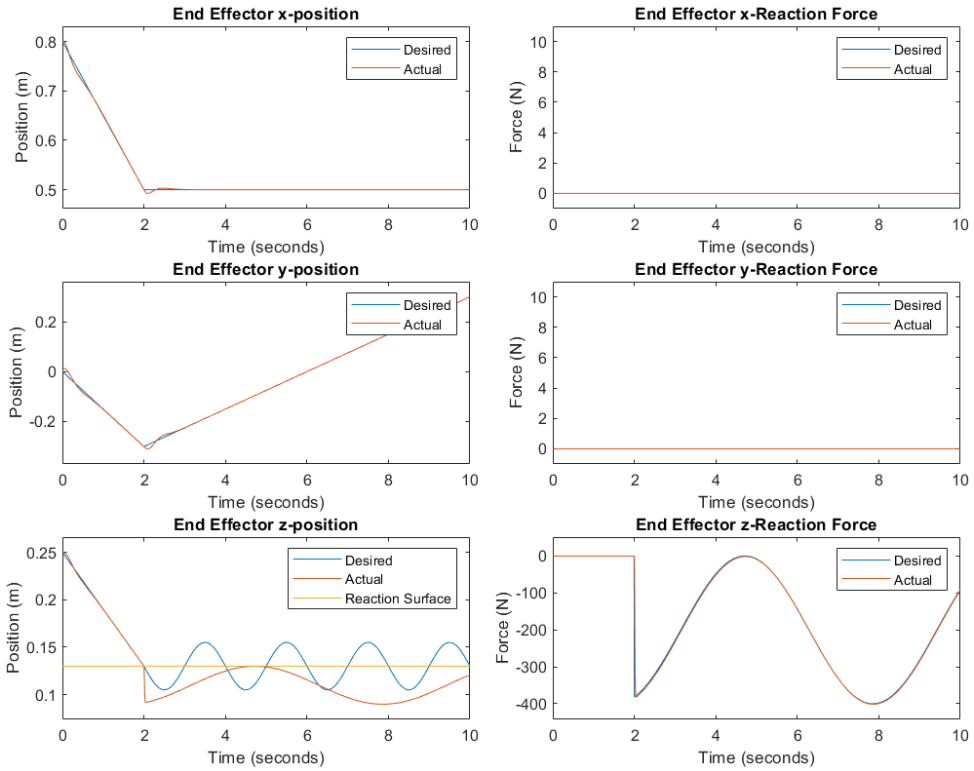

Figure 5.11: Results of variable force tracking

The results of the attempt to track this variable force trajectory are shown in Figure 5.11. Again, the performance was excellent. It was particularly notable that the controller was able to bring the force all the way down to 0N by bringing the end effector up to the edge of the reaction surface without encountering any disruptive behaviour at this boundary. The tracking was almost perfect, with just a slight lag and a small overshoot.

The success of this force controller within the overall hybrid force-position control scheme represented the conclusion of the simulation work that was done in this thesis. More discussion on future directions in which this work could be explored can be found in Chapter 6. The primary application for this controller was to be for use with a physical robot, the details of which will be discussed in the subsequent chapter.

6 Conclusion and Recommendations

The goal of this thesis was to develop a hybrid force-position controller for a 4-DOF SCARA manipulator, and this was successfully completed. However, throughout the course of the project, many different areas for further investigation were discovered and will be discussed in further detail here.

The first and most obvious area for future work would be in the completion of the construction of the physical robot. The decision to stop building it was made in this thesis not because it wasn't a worthwhile objective, but because of internal challenges specific to the project (software availability, time constraints). The construction of the robot would have allowed for testing of the controller on a real robot, which (if successful) would have represented a major success. Implementing a controller on a simulation is one thing, but dealing with a physical robot comes with an entirely separate suite of challenges. Some of these challenges can be resolved without having to change too much in the controller, but it is also possible to encounter challenges that might invalidate core assumptions within the design of the controller, necessitating major changes. Implementing the controller developed in this thesis on an actual physical robot would be the most valuable direction for future work, especially considering the design work that has already been done towards rebuilding the robot.

One important part of implementing the controller on a physical robot would be the redevelopment of the mathematical model of the robot, in order to properly calculate control gains and feedback linearization functions. This redevelopment would certainly require the determination of the physical characteristics of the robot, including the masses, moments of inertia, lengths, centre of mass locations and motor constants for all of the joints and links. In addition to this, a more accurate model of the robot's dynamics would have included a model for friction in each of the joints,

and also the motor dynamics. While many of these characteristics (mass, lengths, motor constants) can be found relatively easily and accurately, other characteristics (moments of inertia, centre of mass locations, friction constants) aren't easily determined, and are very specific to the setup of the robot.

In order to accurately determine these values, a process called *dynamic identification* can be used. In this process, the robot motors are given time-varying sequences of input voltages at each of the actuating motors, and the resulting joint positions and angles are measured over time. Then, a numerical optimization of all of the unknown parameters of the robot would have to be performed. In this optimization process, many different combinations of parameters would be proposed, and then used to map the known input voltages into corresponding output joint positions and angles. The combinations of parameters that generated output joint positions and angles that most closely matched the actual measured output joint positions and angles would form the best possible estimates of these parameters. The optimization process would continue until a set of parameters was found than generated output joint positions and angles that were deemed a close enough fit, and then these would be assumed to be actual representations of the real parameter values. This dynamic identification would ensure that the mathematical model upon which the robot controller was built would be an accurate representation of reality.

There are also many ways that the development of the controller itself could have been pushed further, with or without being implemented on a physical robot. Most of these areas for work involve removing one or more of the assumptions that simplified the work done in this thesis. These directions are listed below:

1. The reaction surface with which the robot had to interact could have been made an oblique plane (i.e. not perpendicular to the z axis), forcing the robot to prioritize between force and position control in more joints than just the prismatic joint.

2. The reaction surface could have been made to be a curved surface, forcing the supervisory controller to constantly re-evaluate the directions of the natural and artificial constraints.

3. The forces generated by the reaction surface could have been made to be more than just a normal force (i.e. the addition of a friction model), necessitating a new balance between the natural and artificial constraints in the plane tangent to the reaction surface.

4. The supervisory controller's selection matrix could have been made more intelligent, giving it the ability to determine on its own when to switch from position dominance to force dominance based on its position relative to the reaction surface.

5. The entire controller could have been improved to make it more robust, making it so that it would not produce erratic and crazy performance failures when presented with challenging boundary conditions or unexpected inputs.

6. The model could have been updated to include random measurement noise to test the robustness of the developed controllers.

Beyond all of these possible improvements to the quality of the controller, a major improvement to the controller could have been achieved if a more math-based approach was taken to determine the force controller's feedback gains than simply trial and error. As mentioned during the discussion of the development of the force controller, a more theoretically valid approach for this was never found throughout the literature, but this does not necessarily mean that such an approach does not exist.

Despite all of these areas for future work, this thesis was totally successful in its aim of developing a hybrid force-position controller for a 4-DOF SCARA manipulator. A kinematic model of the robot was developed with the DH convention, and then manipulated to find the robot's inverse kinematics and workspace. By differentiating the kinematic equations, the robot's Jacobian was found, and this was used to determine the locations of its kinematic singularities. The Lagrange method was used to develop the robot's dynamic equations and complete the mathematical model of the robot.

A simplified model of a 4-DOF SCARA robot was built in Solidworks, and the physical properties of this model were plugged into the mathematical model to generate a simulation of the model in Simulink, and also SimScape. Multiple position controllers were built for both simulations, and when they were applied, the system outputs were very similar - validating the mathematical model of the robot. Ultimately, the most successful of the position controllers was the Decentralized Feedback Linearization/PID controller.

The simulations were then rebuilt to include a reaction surface that would generate an external normal force acting on the end effector if it was interacted with. This allowed for the development of a force controller, the most successful of which

was a PI/Feedforward/Velocity Damping controller. Finally, the two controllers (position and force) were combined in a hybrid force-position control scheme that was successfully able to prioritize between position and force tracking goals.

Bibliography

[1] Goel, R., Gupta, P. (2020). Robotics and Industry 4.0. In: Nayyar, A., Kumar, A. (eds) A Roadmap to Industry 4.0: Smart Production, Sharp Business and Sustainable Development. Advances in Science, Technology & Innovation. Springer, Cham. https://doi.org/10.1007/978-3-030-14544-6_9

[2] Dupont, P. E., Nelson, B. J., Goldfarb, M., Hannaford, B., Menciassi, A., O'Malley, M. K., Simaan, N., Valdastri, P., & Yang, G.-Z. (2021). A decade retrospective of Medical Robotics Research from 2010 to 2020. Science Robotics, 6(60). https://doi.org/10.1126/scirobotics.abi8017

[3] Xiao, B., Chen, C., & Yin, X. (2022). Recent advancements of robotics in construction. Automation in Construction, 144. https://doi.org/10.1016/j.autcon.2022.104591

[4] Bogue, R. (2019). Strong prospects for robots in retail. Industrial Robot: the International Journal of Robotics Research and Application, 46(3), 326–331. https://doi.org/10.1108/ir-01-2019-0023

[5] Craig, J. J. (2018). Introduction to robotics: Mechanics and control. Pearson.

[6] Siciliano, B., Sciavicco, L., & Villani, L. (2009). Robotics modelling, planning and control. New York: Springer.

[7] Hogan, N. (1984). Impedance Control: An Approach to Manipulation. 1984 American Control Conference. doi:10.23919/acc.1984.4788393

[8] Mohammadzadegan, A. F., Ashrafzadeh, F., & Moodi, H. (2019). Hybrid force and position control of a 4DOF surgical robot with Disturbance Observer. 2019 27th Iranian Conference on Electrical Engineering (ICEE). https://doi.org/10.1109/iraniancee.2019.8786635

[9] Oh, S., Woo, H., & Kong, K. (2014). Frequency-shaped impedance control for safe human–robot interaction in reference tracking application. IEEE/ASME Transactions on Mechatronics, 19(6), 1907–1916. https://doi.org/10.1109/tmech.2014.2309118

[10] Park, H., Bae, J.-H., Park, J.-H., Baeg, M.-H., & Park, J. (2013). Intuitive Peg-in-hole assembly strategy with a compliant manipulator. IEEE ISR 2013, 1–5. https://doi.org/10.1109/isr.2013.6695699

[11] Song, H.-C., Kim, Y.-L., & Song, J.-B. (2016). Guidance algorithm for complex-shape peg-in-hole strategy based on geometrical information and force control. Advanced Robotics, 30(8), 552–563. https://doi.org/10.1080/01691864.2015.1130172

[12] Salisbury, J. K. (1980). Active stiffness control of a manipulator in Cartesian coordinates. 1980 19th IEEE Conference on Decision and Control Including the Symposium on Adaptive Processes, 95–100. https://doi.org/10.1109/cdc.1980.272026

[13] Salisbury, J. K., & Craig, J. J. (1982). Articulated Hands: Force Control and Kinematic Issues. The International Journal of Robotics Research, 1(1), 4–17. https://doi.org/10.1177/027836498200100102

[14] Hogan, N. (1985). Impedance Control: An approach to manipulation: Part I—theory. Journal of Dynamic Systems, Measurement, and Control, 107(1), 1–7. https://doi.org/10.1115/1.3140702

[15] Hogan, N. (1985). Impedance Control: An approach to manipulation: Part II—implementation. Journal of Dynamic Systems, Measurement, and Control, 107(1), 8–16. https://doi.org/10.1115/1.3140713

[16] Hogan, N. (1985). Impedance Control: An approach to manipulation: Part III—applications. Journal of Dynamic Systems, Measurement, and Control, 107(1), 17–24. https://doi.org/10.1115/1.3140701

[17] Mason, M. T. (1981). Compliance and Force Control for Computer Controlled Manipulators. IEEE Transactions on Systems, Man, and Cybernetics, 11(6), 418-432. doi:10.1109/tsmc.1981.4308708

[18] Craig, J. J., & Raibert, M. H. (1979). A systematic method of hybrid position/-force control of a manipulator. COMPSAC 79. Proceedings. Computer Software and The IEEE Computer Society's Third International Applications Conference, 1979., 446–451. https://doi.org/10.1109/cmpsac.1979.762539

[19] Raibert, M. H., & Craig, J. J. (1981). Hybrid Position/Force Control of Manipulators. Journal of Dynamic Systems, Measurement and Control, 102, 126-133.

[20] Zeng, G., & Hemami, A. (1997). An overview of Robot Force Control. Robotica, 15(5), 473–482. https://doi.org/10.1017/s026357479700057x

[21] Chiaverini, S., Siciliano, B., & Villani, L. (1999). A survey of Robot Interaction Control Schemes with experimental comparison. IEEE/ASME Transactions on Mechatronics, 4(3), 273–285. https://doi.org/10.1109/3516.789685

[22] Schumacher, M., Wojtusch, J., Beckerle, P., & von Stryk, O. (2019). An introductory review of active compliant control. Robotics and Autonomous Systems, 119, 185–200. https://doi.org/10.1016/j.robot.2019.06.009

[23] Yoshikawa, T. (1987). Dynamic hybrid position/force control of robot manipulators–description of hand constraints and calculation of Joint Driving Force. IEEE Journal on Robotics and Automation, 3(5), 386–392. https://doi.org/10.1109/jra.1987.1087120

[24] Chiaverini, S., & Sciavicco, L. (1993). The parallel approach to force/position control of robotic manipulators. IEEE Transactions on Robotics and Automation, 9(4), 361–373. https://doi.org/10.1109/70.246048

[25] Anderson, R. J., & Spong, M. W. (1988). Hybrid impedance control of robotic manipulators. IEEE Journal on Robotics and Automation, 4(5), 549–556. https://doi.org/10.1109/56.20440

[26] Liu, G. J., & Goldenberg, A. A. (1991). Robust hybrid impedance control of robot manipulators. Proceedings. 1991 IEEE International Conference on Robotics and Automation, 287–292. https://doi.org/10.1109/robot.1991.131589

[27] Lawrence, D. A. (1988). Impedance control stability properties in common implementations. Proceedings. 1988 IEEE International Conference on Robotics and Automation, 2, 1185–1190. https://doi.org/10.1109/robot.1988.12222

[28] An, C. H., & Hollerbach, J. M. (1989). The role of dynamic models in Cartesian force control of manipulators. The International Journal of Robotics Research, 8(4), 51–72. https://doi.org/10.1177/027836498900800403

[29] Yoshikawa, T. (2000). Force control of robot manipulators. Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), 1, 220–226. https://doi.org/10.1109/robot.2000.844062

[30] McClamroch, N. H., & Wang, D. (1988). Feedback stabilization and tracking of constrained robots. IEEE Transactions on Automatic Control, 33(5), 419–426. https://doi.org/10.1109/9.1220

[31] Zhang, H., & Paul, R. (1985). Hybrid control of robot manipulators. Proceedings. 1985 IEEE International Conference on Robotics and Automation, 602–607. https://doi.org/10.1109/robot.1985.1087304

[32] Zhang, H. (1989). Kinematic stability of robot manipulators under force control. Proceedings, 1989 International Conference on Robotics and Automation, 1, 80–85. https://doi.org/10.1109/robot.1989.99971

[33] Fisher, W. D., & Mujtaba, M. S. (1992). Hybrid position/force control: A correct formulation. The International Journal of Robotics Research, 11(4), 299–311. https://doi.org/10.1177/027836499201100403

[34] Craig, J., Hsu, P., & Sastry, S. (1986). Adaptive control of mechanical manipulators. Proceedings. 1986 IEEE International Conference on Robotics and Automation. https://doi.org/10.1109/robot.1986.1087661

[35] Hsia, T. (1986). Adaptive control of robot manipulators - A Review. Proceedings. 1986 IEEE International Conference on Robotics and Automation, 183–189. https://doi.org/10.1109/robot.1986.1087696

[36] Slotine, J.-J. E., & Li, W. (1987). On the adaptive control of robot manipulators. The International Journal of Robotics Research, 6(3), 49–59. https://doi.org/10.1177/027836498700600303

[37] Slotine, J.-J. E. (1984). Sliding controller design for Non-linear Systems. International Journal of Control, 40(2), 421–434. https://doi.org/10.1080/00207178408933284

[38] Utkin, V. I., & Drakunov, S. V. (1992). Sliding mode control in Dynamic Systems. International Journal of Control, 55(4), 1029–1037. https://doi.org/10.1109/cdc.1993.325637

[39] Slotine, J.-J. E. (1985). The robust control of robot manipulators. The International Journal of Robotics Research, 4(2), 49–64. https://doi.org/10.1177/027836498500400205

[40] Slotine, J.-J. E., & Spong, M. W. (1985). Robust robot control with bounded input torques. Journal of Robotic Systems, 2(4), 329–352. https://doi.org/10.1002/rob.4620020402

[41] Colbaugh, R., Seraji, H., & Glass, K. (1993). Direct adaptive impedance control of robot manipulators. Journal of Robotic Systems, 10(2), 217–248. https://doi.org/10.1002/rob.4620100205

[42] Zhen, R. R. Y., & Goldenberg, A. A. (1994). Robust position and force control of robots using sliding mode. Proceedings of the 1994 IEEE International Conference on Robotics and Automation, 1, 623–628. https://doi.org/10.1109/robot.1994.351416

[43] Su, C.-Y., Leung, T.-P., & Zhou, Q.-J. (1992). Force/motion control of constrained robots using sliding mode. IEEE Transactions on Automatic Control, 37(5), 668–672. https://doi.org/10.1109/9.135513

[44] Narendra, K. S., & Parthasarathy, K. (1990). Identification and control of dynamical systems using Neural Networks. IEEE Transactions on Neural Networks, 1(1), 4–27. https://doi.org/10.1109/72.80202

[45] Miyamoto, H., Kawato, M., Setoyama, T., & Suzuki, R. (1988). Feedback-error-learning neural network for trajectory control of a robotic manipulator. Neural Networks, 1(3), 251–265. https://doi.org/10.1016/0893-6080(88)90030-5

[46] Fukuda, T., & Shibata, T. (1992). Theory and applications of neural networks for industrial control systems. IEEE Transactions on Industrial Electronics, 39(6), 472–489. https://doi.org/10.1109/41.170966

[47] Lee, C. C. (1990). Fuzzy logic in control systems: Fuzzy Logic Controller. I. IEEE Transactions on Systems, Man, and Cybernetics, 20(2), 404–418. https://doi.org/10.1109/21.52551

[48] Li, Y. F., & Lau, C. C. (1989). Development of fuzzy algorithms for Servo Systems. IEEE Control Systems Magazine, 9(3), 65–72. https://doi.org/10.1109/37.24814

[49] Arimoto, S. (1990). Learning control theory for robotic motion. International Journal of Adaptive Control and Signal Processing, 4(6), 543–564. https://doi.org/10.1002/acs.4480040610

[50] Jung, S., & Hsia, T. C. (1995). Neural network techniques for robust force control of robot manipulators. Proceedings of Tenth International Symposium on Intelligent Control, 111–116. https://doi.org/10.1109/isic.1995.525046

[51] Kuo, R.-J. (1997). A robotic die polishing system through Fuzzy Neural Networks. Computers in Industry, 32(3), 273–280. https://doi.org/10.1016/s0166-3615(96)00078-4

[52] Bogdan, S., & Kovacic, Z. (1993). Fuzzy rule-based adaptive force control of a single DOF Mechanisms. Proceedings of 8th IEEE International Symposium on Intelligent Control, 469–474. https://doi.org/10.1109/isic.1993.397668

[53] Cheah, C. C., & Wang, D. (1998). Learning impedance control for robotic manipulators. Proceedings of 1995 IEEE International Conference on Robotics and Automation, 452–465. https://doi.org/10.1109/robot.1995.526026

[54] Jeon, D., & Tomizuka, M. (1993). Learning Hybrid Force and position control of robot manipulators. Proceedings 1992 IEEE International Conference on Robotics and Automation, 9(4), 423–431. https://doi.org/10.1109/robot.1992.220146

[55] Nguyen, Q. H., Ha, Q. P., Rye, D. C., & Durrant-Whyte, H. F. (2000). Force/-position tracking for electrohydraulic systems of a robotic excavator. Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187), 5, 5224–5229. https://doi.org/10.1109/cdc.2001.914787

[56] Kim, K.-H., & Kim, J.-Y. (2022). Effective landing strategy of robot leg using Hybrid Force/position control. Intelligent Service Robotics. https://doi.org/10.1007/s11370-022-00441-7

[57] Xu, X., Zhu, D., Zhang, H., Yan, S., & Ding, H. (2019). Application of novel force control strategies to enhance robotic abrasive belt grinding quality of aero-engine blades. Chinese Journal of Aeronautics, 32(10), 2368–2382. https://doi.org/10.1016/j.cja.2019.01.023

[58] Brahmi, B., Saad, M., Rahman, M. H., & Brahmi, A. (2020). Adaptive Force and position control based on quasi-time delay estimation of exoskeleton robot for rehabilitation. IEEE Transactions on Control Systems Technology, 28(6), 2152–2163. https://doi.org/10.1109/tcst.2019.2931522

[59] Chen, G., Zhou, H., & Yang, P. (2022). Force/Position Control Strategy of 3-PRS Ankle Rehabilitation Robot. International Journal of Innovative Computing, Information and Control, 16(2), 481–494.

[60] Jayender, J., Patel, R. V., & Nikumb, S. (2006). Robot-assisted catheter insertion using hybrid impedance control. Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., 607–612. https://doi.org/10.1109/robot.2006.1641777

[61] Xie, Y., Sun, D., Liu, C., Tse, H. Y., & Cheng, S. H. (2009). A force control approach to a robot-assisted cell microinjection system. The International Journal of Robotics Research, 29(9), 1222–1232. https://doi.org/10.1177/0278364909354325

[62] Navarro-Alarcon, D., Liu, Y., & Li, P. (2011). Stable force/position control of a robotic endoscope holder for constrained tasks in nasal surgery. 2011 9th World Congress on Intelligent Control and Automation, 1195–1200. https://doi.org/10.1109/wcica.2011.5970705

[63] Saeidi, H., Opfermann, J. D., Kam, M., Wei, S., Leonard, S., Hsieh, M. H., Kang, J. U., & Krieger, A. (2022). Autonomous robotic laparoscopic surgery for intestinal anastomosis. Science Robotics, 7(62). https://doi.org/10.1126/scirobotics.abj2908

[64] Zidane, I. F., Khattab, Y., Rezeka, S., & El-Habrouk, M. (2022). Robotics in laparoscopic surgery - A Review. Robotica, 1–48. https://doi.org/10.1017/s0263574722001175

[65] Duchemin, G., Dombre, E., Pierrot, F., & Poignet, P. (2003). Robotized skin harvesting. Springer Tracts in Advanced Robotics, 5, 404–413. https://doi.org/10.1007/3-540-36268-1_36

[66] Dombre, E., Duchemin, G., Poignet, P., & Pierrot, F. (2003). Dermarob: A safe robot for reconstructive surgery. IEEE Transactions on Robotics and Automation, 19(5), 876–884. https://doi.org/10.1109/tra.2003.817067

[67] Duchemin, G., Maillet, P., Poignet, P., Dombre, E., & Pierrot, F. (2005). A hybrid position/force control approach for identification of deformation models of skin and underlying tissues. IEEE Transactions on Biomedical Engineering, 52(2), 160–170. https://doi.org/10.1109/tbme.2004.840505

[68] [SCARA Robot]. (n.d.). Retrieved May 17, 2021, from https://www.allied-automation.com/new-low-cost-scara-robots/

[69] Spong, M. W., & Vidyasagar, M. (1989). Robot Modeling and Control.

[70] Khalil, H. K. (2002). Nonlinear Systems (3rd ed.). Upper Saddle River, NJ, New Jersey: Prentice Hall.

[71] Stefani, R. T., Shahian, B., Savant, C. J., Jr., & Hosteller, G. H. (2002). Design of Feedback Control Systems (4th ed.). New York: Oxford University Press.

[72] Khosla, P. K., & Kanade, T. (1986). Experimental Evaluation of the Feedforward Compensation and Computed-Torque Control Schemes. 1986 American Control Conference. doi:10.23919/acc.1986.4789043

[73] Siciliano, B., & Villani, L. (1999). Robot Force Control. Boston: Kluwer Academic.

[74] Hartenberg, R. S., & Denavit, J. (1964). Kinematic synthesis of linkages. McGraw-Hill.

Appendices

A Appendix A: Physical Robot Construction

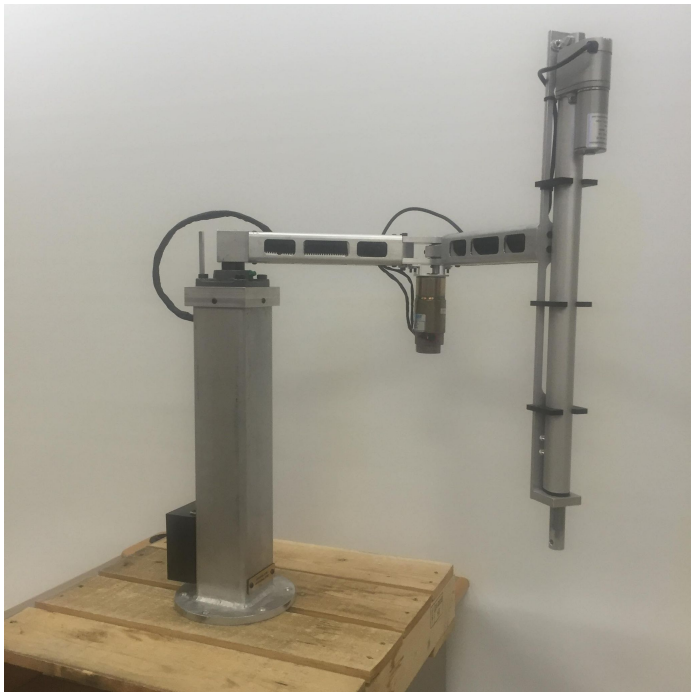

Figure A.1: Photo of existing 3-DOF SCARA manipulator

In addition to the simulation work done in this thesis, a parallel effort was made to design and build a 4-DOF SCARA robot on which the controller could be tested. Previous students had built a 3-DOF SCARA manipulator (shown in Figure A.1)

that originally was going to be used as a base, and augmented to add the fourth degree of freedom, as well as a force/torque sensor. This previous model also had some significant mechanical backlash that had to be rectified in order to make it useful for a high-precision task such as force-position control. Thus, the physical robot construction was broken down into three main lines of effort:

1. Add a fourth degree of freedom
2. Add a force/torque sensor
3. Eliminate mechanical backlash

Unfortunately, due mainly to concerns regarding the accessibility of the software required to interface between the robot motors and the controller, the physical construction of the robot was put on hold and eventually removed from the aims of this thesis. Thus, this chapter will discuss the work that was done, although ultimately a functional 4-DOF SCARA manipulator was never built.

A.1 Eliminating Mechanical Backlash

The first and most significant task in the rebuild of the robot was to get rid of the mechanical backlash in the existing robot. In machine design, mechanical backlash occurs when two moving parts within an assembly have too much clearance between them. A classic example of this is when the teeth of a gear become worn and no longer fit perfectly into the gears they are paired with, thus leaving some room for the gears to move around without any intentional driving force/torque. This example is depicted below in Figure A.2:

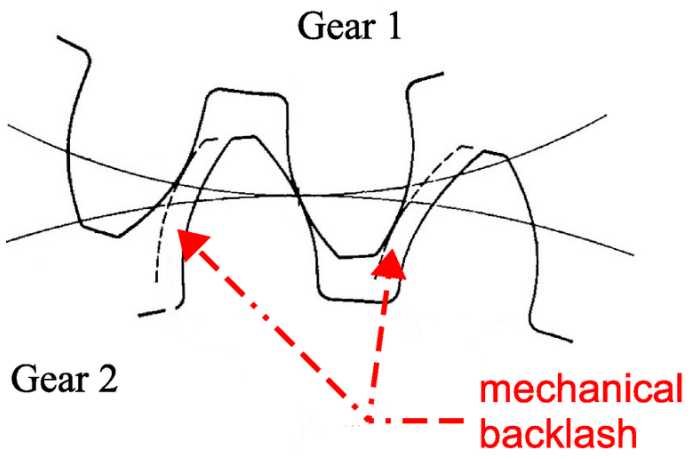

Figure A.2: Example of mechanical backlash

In a robot manipulator, even a small amount of mechanical backlash in any of the joints can lead to the end effector swinging around uncontrollably, which is a significant issue when precision control (such as in this project) is the goal. While it was difficult to determine where in the existing robot's mechanical chain the backlash was being generated, it was assumed to be in two different places.

The first was in the drive trains of the first and second (shoulder and elbow) motors. It was assumed to be here, because the end effector would swing left and right in a manner that could only have been caused by looseness in the two rotary joints, and the connections between the motor housings and shafts to the robot links all seemed to be quite tight. Thus, the internal drive train of each of the motors was thought to be the only other place where this horizontal movement of the end effector could have been coming from.

The second place that backlash was assumed to be coming from was in the two bearings of each of the first and second (shoulder and elbow) joints. It was assumed to be here, because the end effector would sag down a bit, and could also be lifted up - indicating some looseness in the vertical plane. This could have come from the prismatic joint, but this motor was driven by a worm gear which is a mechanism known to have very low backlash, and the prismatic joint itself seemed to have been assembled quite tightly. When the robot was disassembled, it could be clearly seen that the vertical backlash was coming from some bending at each of the two rotary joints, which indicated that the bearings were not serving their purpose properly.

Figure A.4: Existing elbow joint thrust (left) and radial (right) bearings

Figure A.3: Existing shoulder joint thrust bearing

These two sources of backlash had to be dealt with separately. The easier of the two sources to eliminate was the vertical backlash of the end effector coming from the loose vertical bearings in the shoulder and elbow joints. The thrust bearing for the shoulder joint is shown in Figure A.3. For the elbow joint, a combination of one thrust bearing and one radial bearing was used, both are shown in Figure A.4.

The thrust bearings were very strong to support the axial load of the robot arm, and the elbow joint's combination of the thrust and radial bearing was useful for radial loads as well. However, neither joint was well set up to deal with the bending

moment loads that were placed on them with the robot in an outstretched position. This bending load was where the most mechanical play was discovered, and so it was decided that both joints had to be re-designed with new bearings more capable of dealing with bending loads.

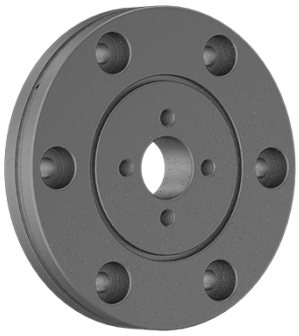

Figure A.5: Face-mounted crossed-roller bearing

The bearings that were selected with this criteria in mind were face-mounted crossed-roller bearings, shown in Figure A.5. These bearings consist of an inner ring that rotates independently from an outer ring, with both rings capable of being fixed to independently moving parts. These bearings are more expensive than typical bearings, at around $100 each, but they are able to deal with thrust, radial and bending loads. Thus, two of these bearings were purchased for the re-design, which was considered to be a good solution to the mechanical backlash in the vertical plane.

Dealing with the horizontal-plane backlash was more challenging. Since it was assumed that this backlash was coming from looseness in the drive trains of each of the rotary motors for the first two joints, the simplest solution would have simply been to purchase new low-backlash motors. These, however were too large and outside of the budget for this build. If the same motors were to be used, a new, low-backlash power transmission element would need to be installed between the output shaft of the motor and the subsequent driven link in the robot arm.

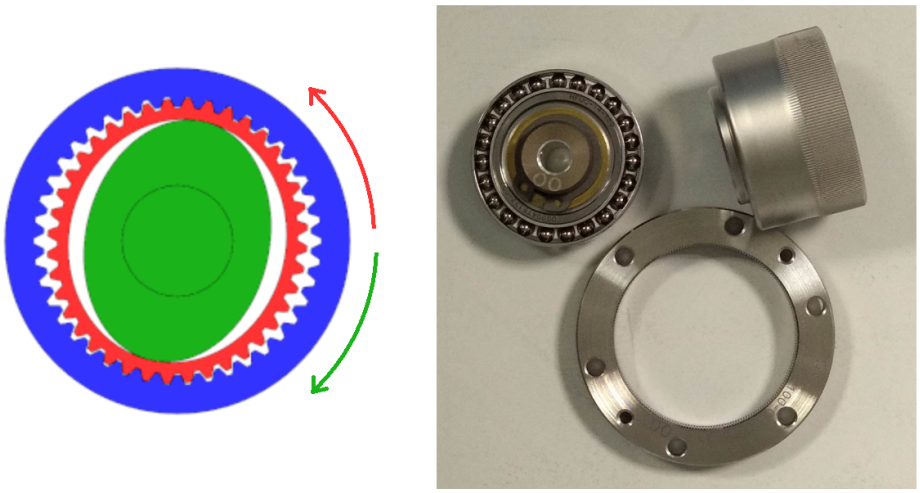

Figure A.6: Graphic demonstrating the strain wave principle (left) and photo of a disassembled Harmonic Drive (right)

Such a low-backlash power transmission device was hard to find. In a stroke of good luck, a gear system known as a Harmonic Drive was available at RMC which was perfectly suited to the task. Harmonic Drives operate on a complex mechanical principle called "strain wave gearing" which is difficult to explain. Figure A.6 shows two images, on the right it shows a photo of a disassembled Harmonic Drive, and on the left it shows a graphic demonstrating the principle. Without going into further detail, the Harmonic Drive can be understood to simply act as a gear reducer, but without any backlash.

The Harmonic Drive was perfect for this application, and would thus be used for the shoulder joint - because the shoulder joint was the place where mechanical backlash would lead to the most movement at the end effector. Thus, the entire first joint was be redesigned to include the Harmonic Drive, as well as the face-mounted crossed-roller bearing. After a few iterations and changes to components to allow them to be more easily built at the RMC Machine Shop, the final design for the first joint was completed, and can be seen in Figure A.7. Unfortunately during the construction of this first joint, the software concerns were realized, which led to a halt in the development of the physical robot - so it was never fully built.

While the Harmonic Drive solved the issue of backlash in the shoulder motor, a solution was still needed for the elbow motor. Unfortunately, only one Harmonic

ITEM NO.	PART NUMBER	QTY.
1	Base Plate	1
2	Riser Boss	16
3	Riser	4
4	Bottom Plate	1
5	Middle Plate	1
6	Top Plate	1
7	Pancake Motor	1
8	Motor Shaft Adapter	1
9	Harmonic Drive	1
10	Flexspline Adapter	1
11	Bearing Adapter	1
12	Face-Mount Bearing	1

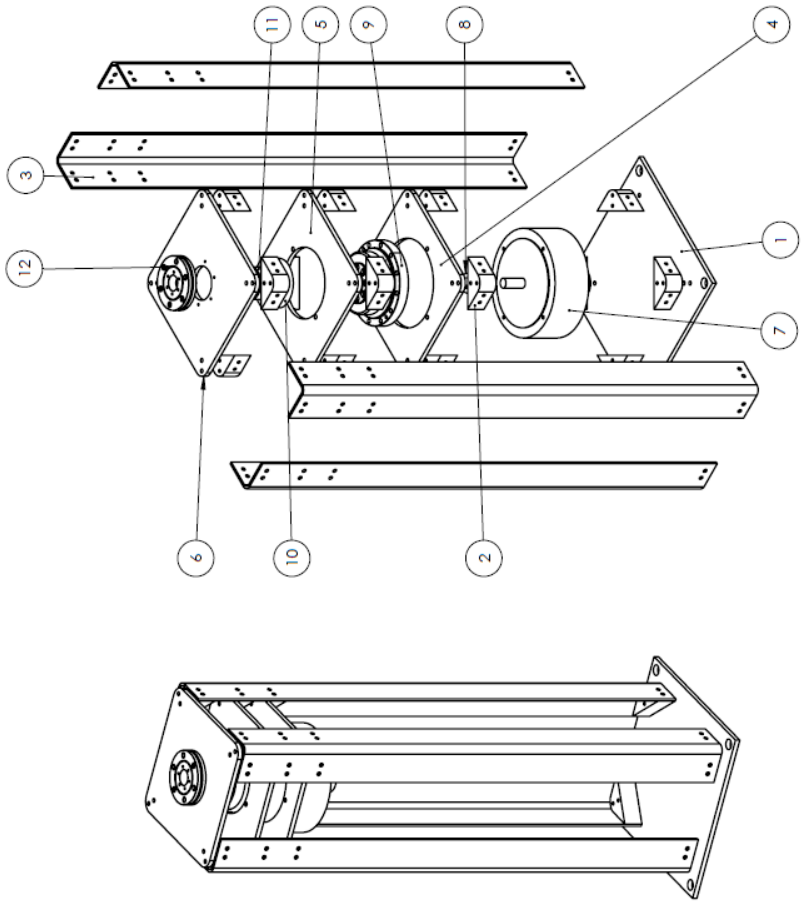

Figure A.7: Exploded view of complete design for the new first joint

Drive was available, and buying another one was outside of the budget of this build. Thus, a cheaper solution had to be developed. After some research, the concept of a "cycloidal drive" was discovered. A cycloidal drive is another gear reduction system that works with a similar principle to the Harmonic Drive, only just not manufactured and sold under a patent. Off-the-shelf cycloidal drives can be purchased, but are even more expensive than Harmonic Drives, so the idea to manufacture one at the RMC Machine Shop was considered.

A functional prototype (shown in Figure A.8) of a custom cycloidal drive was designed on Solidworks and 3D printed at the RMC Machine Shop. This initial prototype was built with the purpose of determining whether the concept could work, and the physical model showed that it was a potentially valid solution to the elbow joint's backlash. A second prototype was planned to be built with CNC-machined rotors and proper bearings and bushings, but again, the physical robot construction was halted before it could be built.

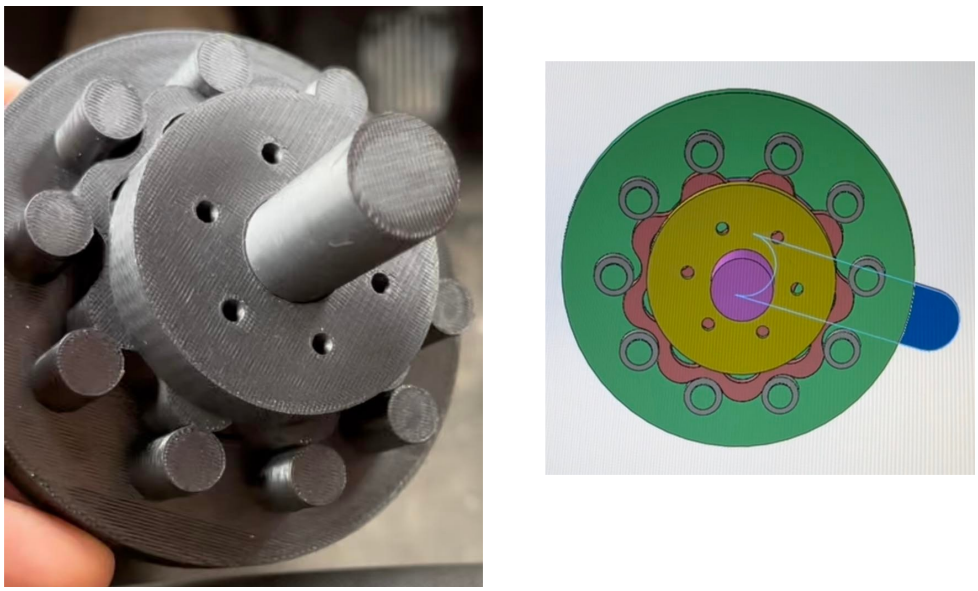

Figure A.8: Images of the cycloidal drive prototype

This represented the full extent of the work that was done to eliminate the backlash in the physical robot. While plenty of work still remains in order to get a functional robot built, assuming that the cycloidal drive is effective in reducing the elbow joint's backlash, all that must to be done is to design and put together all of

Figure A.9: Robotiq 6-axis force-torque sensor

the pieces.

A.2 Force/Torque Sensor and Fourth DOF

The approach to rebuilding the robot was to start by designing and building the first joint, and then to move on and design/build each subsequent link and joint in the order they appeared in the robot's kinematic chain. Thus, the addition of a force/torque sensor and the fourth degree of freedom had not even began by the time that the physical construction had been halted due to the concerns surrounding software availability.

The only exception to this, was the actual purchasing of the force-torque sensor, as it had to be done early on in the project due to its price. After some searching on the market, a 6-axis force-torque sensor made by a company called Robotiq was deemed to be the most suitable option and was purchased, and can be seen in Figure A.9. This force-torque sensor was to be installed directly after the fourth rotary joint in the kinematic chain.

This represents all of the work that was done towards the goal of building the physical robot for testing. While it was certainly unfortunate that it was not completed, significant progress was made, and hopefully future work will see it finished.

B Appendix B: Source Code

B.1 Robot Model Development

```
tic

syms theta_1(t) theta_2(t) d_3(t) theta_4(t)
syms theta_1_dot theta_2_dot d_3_dot theta_4_dot
syms theta_1_ddot theta_2_ddot d_3_ddot theta_4_ddot
syms theta1 theta2 d3 theta4
syms m0 m1 m2 m3 m4
syms g_
syms l0 l1 l2 l3 l4
syms lc0x lc0y lc0z lc1x lc1y lc1z lc2x lc2y lc2z lc3x lc3y
    lc3z lc4x lc4y lc4z
syms I0xx I0yy I0zz I1xx I1yy I1zz I1xy I1yz I1xz I2xx I2yy
    I2zz I2xy I2yz I2xz I3xx I3yy I3zz I3xy I3yz I3xz I4xx
    I4yy I4zz I4xy I4yz I4xz


%% Generate the robot's kinematic model using the following
    DH Table:

%  Link     a       alpha       d       theta
%   1       l1        0         l0      theta1
%   2       l2        pi        0       theta2
%   3       0         0         d3      0
%   4       0         0         l4      theta4
```

```
% First generate the homogeneous transformation matrices:

A_0_1 = [cos(theta_1(t)), -sin(theta_1(t)) , 0   , l1*cos(
    theta_1(t));
         sin(theta_1(t)), cos(theta_1(t))  , 0   , l1*sin(
             theta_1(t));
         0              , 0                , 1   , l0
                          ;
         0              , 0                , 0   , 1
                           ;];

A_1_2 = [cos(theta_2(t)), sin(theta_2(t))  , 0   , l2*cos(
    theta_2(t));
         sin(theta_2(t)), -cos(theta_2(t)) , 0   , l2*sin(
             theta_2(t));
         0              , 0                , -1  , 0
                          ;
         0              , 0                , 0   , 1
                           ;];

A_2_3 = [1              , 0                , 0   , 0
                 ;
         0              , 1                , 0   , 0
                          ;
         0              , 0                , 1   , d_3(t)
                        ;
         0              , 0                , 0   , 1
                           ;];

A_3_4 = [cos(theta_4(t)), -sin(theta_4(t)) , 0   , 0
                 ;
         sin(theta_4(t)), cos(theta_4(t))  , 0   , 0
                          ;
         0              , 0                , 1   , l4
                        ;
         0              , 0                , 0   , 1
```

```
                                        ;];

A_0_2 = simplify(A_0_1*A_1_2);
A_0_3 = simplify(A_0_2*A_2_3);
A_0_4 = simplify(A_0_3*A_3_4);

%Extract the rotation matrices and other important info from
    transformation
%matrices:

R_0_1 = A_0_1(1:3, 1:3);
R_0_2 = A_0_2(1:3, 1:3);
R_0_3 = A_0_3(1:3, 1:3);
R_0_4 = A_0_4(1:3, 1:3);

r_0_1 = A_0_1(1:3,4);
r_1_2 = R_0_1*A_1_2(1:3,4);
r_2_3 = R_0_2*A_2_3(1:3,4);
r_3_4 = R_0_3*A_3_4(1:3,4);

z_0_1 = R_0_1(1:3,3);
z_0_2 = R_0_2(1:3,3);
z_0_3 = R_0_3(1:3,3);
z_0_4 = R_0_4(1:3,3);

%Compute the translational and rotational velocity vectors of
     each frame
%using the iterative method:

w_0 = [0;0;0];
w_1 = w_0 + theta_1_dot*[0;0;1];
w_2 = w_1 + theta_2_dot*z_0_1;
w_3 = w_2;
w_4 = w_3 + theta_4_dot*z_0_3;

p_dot_0 = zeros(3,1);
p_dot_1 = p_dot_0 + cross(w_1,r_0_1);
```

```
p_dot_2 = p_dot_1 + cross(w_2,r_1_2);
p_dot_3 = p_dot_2 + cross(w_3,r_2_3) + d_3_dot*z_0_2;

= p_dot_3 + cross(w_4,r_3_4);

%Set the velocities equal to zero and solve to fine the
    translational and
%rotational Jacobian matrices:

eqns1 = [w_1(1)==0, w_1(2) ==0, w_1(3)==0];
vars1 = [theta_1_dot];
[J_01, ~] = equationsToMatrix(eqns1, vars1);
eqns2 = [w_2(1)==0, w_2(2) ==0, w_2(3)==0];
vars2 = [theta_1_dot, theta_2_dot];
[J_02, ~] = equationsToMatrix(eqns2, vars2);
eqns3 = [w_3(1)==0, w_3(2) ==0, w_3(3)==0];
vars3 = [theta_1_dot, theta_2_dot, d_3_dot];
[J_03, ~] = equationsToMatrix(eqns3, vars3);
eqns4 = [w_4(1)==0, w_4(2) ==0, w_4(3)==0];
vars4 = [theta_1_dot, theta_2_dot, d_3_dot, theta_4_dot];
[J_04, ~] = equationsToMatrix(eqns4, vars4)

eqns1 = [p_dot_1(1)==0, p_dot_1(2) ==0, p_dot_1(3)==0];
vars1 = [theta_1_dot];
[J_P1, ~] = equationsToMatrix(eqns1, vars1);
eqns2 = [p_dot_2(1)==0, p_dot_2(2) ==0, p_dot_2(3)==0];
vars2 = [theta_1_dot, theta_2_dot];
[J_P2, ~] = equationsToMatrix(eqns2, vars2);
eqns3 = [p_dot_3(1)==0, p_dot_3(2) ==0, p_dot_3(3)==0];
vars3 = [theta_1_dot, theta_2_dot, d_3_dot];
[J_P3, ~] = equationsToMatrix(eqns3, vars3);
eqns4 = [p_dot_4(1)==0, p_dot_4(2) ==0, p_dot_4(3)==0];
vars4 = [theta_1_dot, theta_2_dot, d_3_dot, theta_4_dot];
[J_P4, ~] = equationsToMatrix(eqns4, vars4)

J_4 = [J_P4 ; J_04];
```

```
disp('Jacobian Computed')
toc

%% Generate a dynamic model for the robot using Lagrange's
    Method:

%First establish the dynamic parameters:

Ic1 = [I1xx 0 0;0 I1yy 0; 0 0 I1zz];
Ic2 = [I2xx 0 0;0 I2yy 0; 0 0 I2zz];
Ic3 = [I3xx 0 0;0 I3yy 0; 0 0 I3zz];
Ic4 = [I4xx 0 0;0 I4yy 0; 0 0 I4zz];

lc1_wrt0 = R_0_1*[lc1x; lc1y; lc1z];
lc2_wrt0 = R_0_2*[lc2x; lc2y; lc2z];
lc3_wrt0 = R_0_3*[lc3x; lc3y; lc3z];
lc4_wrt0 = R_0_4*[lc4x; lc4y; lc4z];

%Next generate expressions for the translational and
    rotational velocities,
%and the positions of the centres of masses of each link:

q1_dot = [theta_1_dot];
q2_dot = [theta_1_dot;theta_2_dot];
q3_dot = [theta_1_dot;theta_2_dot;d_3_dot];
q4_dot = [theta_1_dot;theta_2_dot;d_3_dot;theta_4_dot];

vc1 = 0 + cross(J_01*q1_dot,lc1_wrt0);
vc2 = J_P1*q1_dot + cross(J_02*q2_dot,lc2_wrt0);
vc3 = J_P2*q2_dot + cross(J_03*q3_dot,lc3_wrt0);
vc4 = J_P3*q3_dot + cross(J_04*q4_dot,lc4_wrt0);

omega1 = J_01*q1_dot;
omega2 = inv(R_0_2)*J_02*q2_dot;
omega3 = inv(R_0_3)*J_03*q3_dot;
omega4 = inv(R_0_4)*J_04*q4_dot;
```

```
Pc1 = lc1_wrt0;
Pc2 = A_0_1(1:3,4) + lc2_wrt0;
Pc3 = A_0_2(1:3,4) + lc3_wrt0;
Pc4 = A_0_3(1:3,4) + lc4_wrt0;

%Use these to find expressions for the kinetic and potential
    energies of
%each link as a function of each of the joint variables (
    which each are
%represented as a function of time):

k1 = 0.5*m1*transpose(vc1)*vc1 + 0.5*transpose(omega1)*Ic1*
    omega1;
k2 = 0.5*m2*transpose(vc2)*vc2 + 0.5*transpose(omega2)*Ic2*
    omega2;
k3 = 0.5*m3*transpose(vc3)*vc3 + 0.5*transpose(omega3)*Ic3*
    omega3;
k4 = 0.5*m4*transpose(vc4)*vc4 + 0.5*transpose(omega4)*Ic4*
    omega4;

k = k1+k2+k3+k4;

g1 = -m1*[0 0 g_]*Pc1;
g2 = -m2*[0 0 g_]*Pc2;
g3 = -m3*[0 0 g_]*Pc3;
g4 = -m4*[0 0 g_]*Pc4;

g = g1+g2+g3+g4;
g = simplify(g);

%Change variables and differentiate to continue Lagrange's
    Method:

k = subs(subs(subs(subs(k,theta_1,theta1),theta_2,theta2),d_3
    ,d3),theta_4,theta4)
g = subs(subs(subs(subs(g,theta_1,theta1),theta_2,theta2),d_3
    ,d3),theta_4,theta4)
```

```
dk_dtheta_dot = [diff(k,theta_1_dot);diff(k,theta_2_dot);diff
    (k,d_3_dot);diff(k,theta_4_dot)];
dk_dtheta_dot = subs(subs(subs(subs(dk_dtheta_dot,theta_1_dot
    ,diff(theta_1,t)),theta_2_dot,diff(theta_2,t)),d_3_dot,
    diff(d_3,t)),theta_4_dot,diff(theta_4,t));
time_derivative_dk_dtheta_dot = diff(dk_dtheta_dot,t);
dk_dtheta = [diff(k,theta1);diff(k,theta2);diff(k,d3);diff(k,
    theta4)];
dk_dtheta = subs(subs(subs(subs(dk_dtheta,theta1,theta_1),
    theta2,theta_2),d3,d_3),theta4,theta_4);
dg_dtheta = [diff(g,theta1);diff(g,theta2);diff(g,d3);diff(g,
    theta4)];
dg_dtheta = subs(subs(subs(subs(dg_dtheta,theta1,theta_1),
    theta2,theta_2),d3,d_3),theta4,theta_4);

tau = time_derivative_dk_dtheta_dot - dk_dtheta + dg_dtheta;
tau = subs(tau,[diff(theta_1(t),t,t),diff(theta_2(t),t,t),
    diff(d_3(t),t,t),diff(theta_4(t),t,t)],[theta_1_ddot,
    theta_2_ddot,d_3_ddot,theta_4_ddot]);
tau = simplify(tau);

tau = subs(tau,[theta_1(t),theta_2(t),d_3(t),theta_4(t)],[
    theta1,theta2,d3,theta4]);
M = tau - subs(tau,[theta_1_ddot,theta_2_ddot,d_3_ddot,
    theta_4_ddot],[0,0,0,0])
G = tau - subs(tau,[g_],[0])
G = simplify(G);
C = tau - M - G
C = simplify(C)

M11 = simplify((M(1) - subs(M(1),theta_1_ddot,0))/
    theta_1_ddot);
M12 = simplify((M(1) - subs(M(1),theta_2_ddot,0))/
    theta_2_ddot);
M13 = simplify((M(1) - subs(M(1),d_3_ddot,0))/d_3_ddot);
M14 = simplify((M(1) - subs(M(1),theta_4_ddot,0))/
```

```
    theta_4_ddot);
M21 = simplify((M(2) - subs(M(2),theta_1_ddot,0))/
    theta_1_ddot);
M22 = simplify((M(2) - subs(M(2),theta_2_ddot,0))/
    theta_2_ddot);
M23 = simplify((M(2) - subs(M(2),d_3_ddot,0))/d_3_ddot);
M24 = simplify((M(2) - subs(M(2),theta_4_ddot,0))/
    theta_4_ddot);
M31 = simplify((M(3) - subs(M(3),theta_1_ddot,0))/
    theta_1_ddot);
M32 = simplify((M(3) - subs(M(3),theta_2_ddot,0))/
    theta_2_ddot);
M33 = simplify((M(3) - subs(M(3),d_3_ddot,0))/d_3_ddot);
M34 = simplify((M(3) - subs(M(3),theta_4_ddot,0))/
    theta_4_ddot);
M41 = simplify((M(4) - subs(M(4),theta_1_ddot,0))/
    theta_1_ddot);
M42 = simplify((M(4) - subs(M(4),theta_2_ddot,0))/
    theta_2_ddot);
M43 = simplify((M(4) - subs(M(4),d_3_ddot,0))/d_3_ddot);
M44 = simplify((M(4) - subs(M(4),theta_4_ddot,0))/
    theta_4_ddot);

M_matrix = [M11,M12,M13,M14;M21,M22,M23,M24;M31,M32,M33,M34;
    M41,M42,M43,M44]
M_inv = inv(M_matrix)

C11 = simplify((C(1) - subs(C(1),theta_1_dot,0))/theta_1_dot)
    ;
C12 = simplify((C(1) - subs(C(1),theta_2_dot,0))/theta_2_dot)
    ;
C13 = simplify((C(1) - subs(C(1),d_3_dot,0))/d_3_dot);
C14 = simplify((C(1) - subs(C(1),theta_4_dot,0))/theta_4_dot)
    ;
C21 = simplify((C(2) - subs(C(2),theta_1_dot,0))/theta_1_dot)
    ;
C22 = simplify((C(2) - subs(C(2),theta_2_dot,0))/theta_2_dot)
```

```
    ;
C23 = simplify((C(2) - subs(C(2),d_3_dot,0))/d_3_dot);
C24 = simplify((C(2) - subs(C(2),theta_4_dot,0))/theta_4_dot)
    ;
C31 = simplify((C(3) - subs(C(3),theta_1_dot,0))/theta_1_dot)
    ;
C32 = simplify((C(3) - subs(C(3),theta_2_dot,0))/theta_2_dot)
    ;
C33 = simplify((C(3) - subs(C(3),d_3_dot,0))/d_3_dot);
C34 = simplify((C(3) - subs(C(3),theta_4_dot,0))/theta_4_dot)
    ;
C41 = simplify((C(4) - subs(C(4),theta_1_dot,0))/theta_1_dot)
    ;
C42 = simplify((C(4) - subs(C(4),theta_2_dot,0))/theta_2_dot)
    ;
C43 = simplify((C(4) - subs(C(4),d_3_dot,0))/d_3_dot);
C44 = simplify((C(4) - subs(C(4),theta_4_dot,0))/theta_4_dot)
    ;

C_matrix = [C11,C12,C13,C14;C21,C22,C23,C24;C31,C32,C33,C34;
    C41,C42,C43,C44]
```

B.2 Position Control Simulation

```
tic

close all

%% Establish simulation parameters using the following values
    :

g__ = -9.8;
l0_ = 0.5;
l1_ = 0.4;
l2_ = 0.4;
l4_ = 0.15;
```

```
m1_ = 6.01;
m2_ = 5.37;
m3_ = 4.03;
m4_ = 0.91;
I1zz_ = 0.1807;
I2zz_ = 0.1558;
I3zz_ = 0.0064;
I4zz_ = 0.0025;
lc1x_ = -0.185;
lc2x_ = -0.224;
lc3x_ = 0;
lc4x_ = 0;
lc1y_ = 0;
lc2y_ = 0;
lc3y_ = 0;
lc4y_ = 0;
lc1z_ = 0;
lc2z_ = 0;
lc3z_ = -0.201;
lc4z_ = -0.122;

theta1upperlimit = pi/2;
theta1lowerlimit = -theta1upperlimit;
theta2upperlimit = 7*pi/8;
theta2lowerlimit = -theta2upperlimit;
d3upperlimit = 0.35;
d3lowerlimit = 0.1;
theta4upperlimit = pi;
theta4lowerlimit = -theta4upperlimit;

joint1motorlimit = 1000;
joint2motorlimit = 1000;
joint3motorlimit = 1000;
joint4motorlimit = 1000;

M_inv = subs(M_inv,{g, l0, l1, l2, l4, m1, m2, m3, m4, I1zz,
    I2zz, I3zz, I4zz, lc1x, lc2x, lc3x, lc4x, lc1y, lc2y, lc3y
```

```
    , lc4y},{g__, l0_, l1_, l2_, l4_, m1_, m2_, m3_, m4_,
    I1zz_, I2zz_, I3zz_, I4zz_, lc1x_, lc2x_, lc3x_, lc4x_,
    lc1y_, lc2y_, lc3y_, lc4y_});
M = subs(M_matrix,{g, l0, l1, l2, l4, m1, m2, m3, m4, I1zz,
    I2zz, I3zz, I4zz, lc1x, lc2x, lc3x, lc4x, lc1y, lc2y, lc3y
    , lc4y},{g__, l0_, l1_, l2_, l4_, m1_, m2_, m3_, m4_,
    I1zz_, I2zz_, I3zz_, I4zz_, lc1x_, lc2x_, lc3x_, lc4x_,
    lc1y_, lc2y_, lc3y_, lc4y_});
C = subs(C_matrix,{g, l0, l1, l2, l4, m1, m2, m3, m4, I1zz,
    I2zz, I3zz, I4zz, lc1x, lc2x, lc3x, lc4x, lc1y, lc2y, lc3y
    , lc4y},{g__, l0_, l1_, l2_, l4_, m1_, m2_, m3_, m4_,
    I1zz_, I2zz_, I3zz_, I4zz_, lc1x_, lc2x_, lc3x_, lc4x_,
    lc1y_, lc2y_, lc3y_, lc4y_});
G = subs(G,{g_, l0, l1, l2, l4, m1, m2, m3, m4, I1zz, I2zz,
    I3zz, I4zz, lc1x, lc2x, lc3x, lc4x, lc1y, lc2y, lc3y, lc4y
    },{g__, l0_, l1_, l2_, l4_, m1_, m2_, m3_, m4_, I1zz_,
    I2zz_, I3zz_, I4zz_, lc1x_, lc2x_, lc3x_, lc4x_, lc1y_,
    lc2y_, lc3y_, lc4y_});

%% First we plot the workspace of the robot

zz1 = [];
zz2 = [];

n = 50;

cornerr = sqrt((l1_*cos(theta1upperlimit) + l2_*cos(
    theta1upperlimit+theta2upperlimit))^2 + (l1_*sin(
    theta1upperlimit) + l2_*sin(theta1upperlimit+
    theta2upperlimit))^2 );
cornerangle = acos(l1_*cos(theta1upperlimit) + l2_*cos(
    theta1upperlimit+theta2upperlimit)/cornerr);

surface1theta = theta1lowerlimit:(theta1upperlimit-
    theta1lowerlimit)/n:theta1upperlimit;
surface1r = cornerr:((l1_ + l2_)-cornerr)/n:l1_ + l2_;
```

```
xx2 = zeros(n+1,n+1);
yy2 = zeros(n+1,n+1);

for i = 1:n+1
    angle1 = theta1upperlimit + (cornerangle -
        theta1upperlimit)*(i-1)/n;
    angle2 = (cornerangle - theta1upperlimit)*(i-1)/n;
    radius = sqrt((l1_*cos(angle2))^2 - l1_^2 +l2_^2) + l1_*
        cos(angle2);
    for j = 1:n+1
        thickness = (n+1-j)/n;
        xx2(i,j) = (cornerr+(radius-cornerr)*thickness)*cos(
            angle1);
        yy2(i,j) = (cornerr+(radius-cornerr)*thickness)*sin(
            angle1);
    end
end

xx3 = zeros(n+1,n+1);
yy3 = zeros(n+1,n+1);

for i = 1:n+1
    angle1 = theta1lowerlimit + (-cornerangle -
        theta1lowerlimit)*(i-1)/n;
    angle2 = (-cornerangle - theta1lowerlimit)*(i-1)/n;
    radius = sqrt((l1_*cos(angle2))^2 - l1_^2 +l2_^2) + l1_*
        cos(angle2);
    for j = 1:n+1
        thickness = (n+1-j)/n;
        xx3(i,j) = (cornerr+(radius-cornerr)*thickness)*cos(
            angle1);
        yy3(i,j) = (cornerr+(radius-cornerr)*thickness)*sin(
            angle1);
    end
end

[thetatheta,rr] = meshgrid([surface1theta],[surface1r]);
```

```
xx1 = rr.*cos(thetatheta);
yy1 = rr.*sin(thetatheta);

z4 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
    n:l0_ - l4_ - d3lowerlimit;
x4 = (l1_+l2_)*cos(surface1theta);
y4 = (l1_+l2_)*sin(surface1theta);

yy4 = [];
for i = 1:n+1
    yy4 = [yy4;y4];
end

[xx4, zz4] = meshgrid([x4],[z4]);

z5 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
    n:l0_ - l4_ - d3lowerlimit;
x5 = l1_*cos(theta1upperlimit) + l2_*cos(theta1upperlimit:
    theta2upperlimit/n:theta1upperlimit+theta2upperlimit);
y5 = l1_*sin(theta1upperlimit) + l2_*sin(theta1upperlimit:
    theta2upperlimit/n:theta1upperlimit+theta2upperlimit);

yy5 = [];
for i = 1:n+1
    yy5 = [yy5;y5];
end

[xx5, zz5] = meshgrid([x5],[z5]);

z6 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
    n:l0_ - l4_ - d3lowerlimit;
x6 = l1_*cos(theta1lowerlimit) + l2_*cos(theta1lowerlimit:
    theta2lowerlimit/n:theta1lowerlimit+theta2lowerlimit);
y6 = l1_*sin(theta1lowerlimit) + l2_*sin(theta1lowerlimit:
    theta2lowerlimit/n:theta1lowerlimit+theta2lowerlimit);
```

```
yy6 = [];
for i = 1:n+1
    yy6 = [yy6;y6];
end

[xx6, zz6] = meshgrid([x6],[z6]);

z7 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
   n:l0_ - l4_ - d3lowerlimit;
x7 = (cornerr)*cos(linspace(-cornerangle,cornerangle,n+1));
y7 = (cornerr)*sin(linspace(-cornerangle,cornerangle,n+1));

yy7 = [];
for i = 1:n+1
    yy7 = [yy7;y7];
end

[xx7, zz7] = meshgrid([x7],[z7]);

grey = [0.1,0.1,0.1];

for i = 1:n+1
    zzz = [];
    for j = 1:n+1
        zzz = [zzz, l0_ - l4_ - d3lowerlimit];
    end
    zz1 = [zz1;zzz];
end

for i = 1:n+1
    zzz = [];
    for j = 1:n+1
        zzz = [zzz, l0_ - l4_ - d3upperlimit];
    end
    zz2 = [zz2;zzz];
end
```

```
figure(1)
surf(xx1,yy1,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx1,yy1,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx3,yy3,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx3,yy3,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
```

```
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x6,y6,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x6,y6,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x7,y7,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x7,y7,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3([cornerr*cos(cornerangle),cornerr*cos(cornerangle)],[
```

```
    cornerr*sin(cornerangle),cornerr*sin(cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on
plot3([cornerr*cos(-cornerangle),cornerr*cos(-cornerangle)],[
    cornerr*sin(-cornerangle),cornerr*sin(-cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on

figure(5)
surf(xx1,yy1,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx1,yy1,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx3,yy3,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx3,yy3,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
```

```
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x6,y6,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
```

```
plot3(x6,y6,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x7,y7,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x7,y7,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3([cornerr*cos(cornerangle),cornerr*cos(cornerangle)],[
    cornerr*sin(cornerangle),cornerr*sin(cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on
plot3([cornerr*cos(-cornerangle),cornerr*cos(-cornerangle)],[
    cornerr*sin(-cornerangle),cornerr*sin(-cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on

%% Generate and plot the non-rigid surface (nrs) that the
    robot will interact with

nrs_ymax = 0.4;
nrs_ymin = -0.4;
nrs_xmax = 0.6;
nrs_xmin = 0.2;
nrs_zbase = 0.13;

nrs_stiffness = 10000;
nrs_frictioncoefficient = 0.2;

nrs_x = linspace(nrs_xmin,nrs_xmax,n+1);
nrs_y = linspace(nrs_ymin,nrs_ymax,n+1);

nrs_zz = [];
```

```
for i = 1:n+1
    nrs_z = [];
    for j = 1:n+1
        nrs_z = [nrs_z, nrs_zbase];
    end
    nrs_zz = [nrs_zz;nrs_z];
end

[nrs_xx, nrs_yy] = meshgrid([nrs_x],[nrs_y]);

%% Plot the reaction surface

% figure(1)
% surf(nrs_xx,nrs_yy,nrs_zz,'FaceColor',[1, 1, 0],'FaceAlpha
    ',.8,'EdgeColor','none');
% get(gcf, 'Renderer');
% hold on
% plot3(nrs_x,linspace(nrs_ymax,nrs_ymax,n+1),nrs_zz(n+1,:),'
    black')
% hold on
% plot3(nrs_x,linspace(nrs_ymin,nrs_ymin,n+1),nrs_zz(1,:),'
    black')
% hold on
% plot3(linspace(nrs_xmax,nrs_xmax,n+1),nrs_y,nrs_zz(:,n+1),'
    black')
% hold on
% plot3(linspace(nrs_xmin,nrs_xmin,n+1),nrs_y,nrs_zz(:,1),'
    black')
% hold on

%% Generate the desired trajectory and the corresponding
    joint angle timeseries using the inverse kinematics, as
    well as the desired normal and friction forces:

trial_time = 10;
stepsize = 0.005;
```

```
x_d = [];
y_d = [];
z_d = [];
zo_d = [];
theta_1_d = [];
theta_2_d = [];
theta_1_d1 = [];
theta_2_d1 = [];
theta_4_d1 = [];
theta_1_d2 = [];
theta_2_d2 = [];
theta_4_d2 = [];
d_3_d = [];
z_base = [];
theta_4_d = [];
f_normal_d = [];
f_friction_d = [];
t = [];
DOFs = [];
spline = [];

for i = 0:stepsize:trial_time - stepsize
    t = [t; i];
    j = i/trial_time;
%     Line from starting position to surface
%     x__d = l1_ + l2_ - j*(l1_ + l2_ - (nrs_xmin + nrs_xmax)
    /2);
%     y__d = j*(nrs_ymin + 0.1*(nrs_ymax - nrs_ymin));
%     z__d = l0_ - l4_ - d3lowerlimit - j*(l0_ - l4_ -
    d3lowerlimit - nrs_zz(n/10,n/2));
%     Squiggle within single area
%     x__d = 0.5 + j*0.2;
%     y__d = -0.2 + j*0.4;
%     z__d = 0.05 + j*0.15 + 0.025*sin(j*10*pi);
%     Squiggle within two areas
%     x__d = 0.1 + j*0.4;
%     y__d = -0.4 + j*0.6;
```

```
%     z__d = 0.05 + j*0.15 + 0.025*sin(j*10*pi);
%    Squiggle within three areas
%     x__d = 0.25;
%     y__d = -0.4 + j*0.8;
%     z__d = 0.05 + j*0.15 + 0.025*sin(j*10*pi);
%    NRS test line
%     x__d = 0.5;
%     y__d = -0.2 + j*0.4;
%     z__d = 0.08 + 0.025*sin(j*10*pi);
%     Position Only Controller Test Line;
    x__d = 0.8 - 0.4*j - 0.05*sin(j*4*pi);
    y__d = -0.2*j + 0.15*sin(j*1.5*pi);
    z__d = 0.25 - 0.1*j - 0.005*sin(j*15*pi);
    spline = [spline; x__d y__d z__d];
    zo__d = 0;
    x_d = [x_d; x__d];
    y_d = [y_d; y__d];
    z_d = [z_d; z__d];
    zo_d = [zo_d; zo__d];
    theta_2__d1 = acos((x__d^2 + y__d^2 - l1_^2 - l2_^2)/(2*
        l1_*l2_));
    theta_2__d2 = -acos((x__d^2 + y__d^2 - l1_^2 - l2_^2)/(2*
        l1_*l2_));
    theta_1__d1 = atan2(y__d,x__d) - asin(l2_*sin(theta_2__d1
        )/(sqrt(x__d^2 + y__d^2)));
    theta_1__d2 = atan2(y__d,x__d) - asin(l2_*sin(theta_2__d2
        )/(sqrt(x__d^2 + y__d^2)));
    d_3__d = -z__d + l0_ - l4_;
    theta_4__d1 = zo__d - theta_1__d1 - theta_2__d1;
    theta_4__d2 = zo__d - theta_1__d2 - theta_2__d2;
    theta_2__d1check = or(theta_2__d1 < theta2lowerlimit,
        theta_2__d1 > theta2upperlimit);
    theta_2__d2check = or(theta_2__d2 < theta2lowerlimit,
        theta_2__d2 > theta2upperlimit);
    theta_1__d1check = or(theta_1__d1 < theta1lowerlimit,
        theta_1__d1 > theta1upperlimit);
    theta_1__d2check = or(theta_1__d2 < theta1lowerlimit,
```

```
    theta_1__d2 > theta1upperlimit);
if or(theta_2__d1check,theta_1__d1check) == 0
    if or(theta_2__d2check,theta_1__d2check) == 0
        DOFs = [DOFs; 2];
        theta_1__d = theta_1__d1;
        theta_2__d = theta_2__d1;
        theta_4__d = theta_4__d1;
    else
        DOFs = [DOFs; 1.1];
        theta_1__d = theta_1__d1;
        theta_2__d = theta_2__d1;
        theta_4__d = theta_4__d1;
    end
else
    if or(theta_2__d2check,theta_1__d2check) == 0
        DOFs = [DOFs; 1.2];
        theta_1__d = theta_1__d2;
        theta_2__d = theta_2__d2;
        theta_4__d = theta_4__d2;
    else
        DOFs = [DOFs; 0];
        theta_1__d = theta_1__d1;
        theta_2__d = theta_2__d1;
        theta_4__d = theta_4__d1;
    end
end
theta_1_d = [theta_1_d; theta_1__d];
theta_2_d = [theta_2_d; theta_2__d];
theta_4_d = [theta_4_d; theta_4__d];
theta_1_d1 = [theta_1_d1; theta_1__d1];
theta_2_d1 = [theta_2_d1; theta_2__d1];
theta_4_d1 = [theta_4_d1; theta_4__d1];
theta_1_d2 = [theta_1_d2; theta_1__d2];
theta_2_d2 = [theta_2_d2; theta_2__d2];
theta_4_d2 = [theta_4_d2; theta_4__d2];
d_3_d = [d_3_d; d_3__d];
z_base = [z_base; l0_ - l4_ - nrs_zbase];
```

```
    f_normal_d = [f_normal_d; 100];
    f_friction_d = [f_friction_d; 20];
end

if sum(DOFs == 0) > 0
    disp('The desired trajectory is not fully enclosed in the
        reachable workspace')
else
    if and(sum(DOFs == 1.1) > 0,sum(DOFs == 1.2) > 0 )
        disp('The desired trajectory has at least one
            inflection point')
    else
        if sum(DOFs == 1.1) > 0
            theta_1_d = theta_1_d1;
            theta_2_d = theta_2_d1;
        end
        if sum(DOFs == 1.2) > 0
            theta_1_d = theta_1_d2;
            theta_2_d = theta_2_d2;
        end
    end
end

figure(2)
subplot(4,2,1)
plot(t,theta_1_d)
xlabel('Time (seconds)')
ylabel('\theta_1 (radians)')
title('Desired and Actual Angle for Joint 1')
legend('Desired Angle')
hold on

subplot(4,2,3)
plot(t,theta_2_d)
xlabel('Time (seconds)')
ylabel('\theta_2 (radians)')
title('Desired and Actual Angle for Joint 2')
```

```
legend('Desired Angle')
hold on

subplot(4,2,5)
plot(t,d_3_d)
xlabel('Time (seconds)')
ylabel('d_3 (metres)')
title('Desired and Actual Position for Joint 3')
legend('Desired Position')
hold on

subplot(4,2,7)
plot(t,theta_4_d)
xlabel('Time (seconds)')
ylabel('\theta_4 (radians)')
title('Desired and Actual Angle for Joint 4')
legend('Desired Angle')
hold on

theta_1_d = [t,theta_1_d];
theta_2_d = [t,theta_2_d];
d_3_d = [t,d_3_d];
theta_4_d = [t,theta_4_d];

figure(1)
plot3(x_d,y_d,z_d,'color','black','linewidth',2);
hold on

%% Establish the timeseries of external forces/torques
    applied to the end effector %%

fext = [];

for i = 0:stepsize:trial_time - stepsize
    j = i/trial_time;
    if i > trial_time/2
        fext = [ fext; i, 0, 0, 0, 0, 0, 0 ];
```

```
    else
        fext = [ fext; i, 0, 0, 0, 0, 0, 0 ];
    end
end

%% Run the simulation and plot the results

Kp = [1347.94; 271.22; 1192.9; 0.605];
Kd = [150.39; 30.26; 133.1; 0.0675];
Ki = [4010.4; 806.9; 3549.03; 1.8];

sim('RobotSimDecentralizedPID')
sim('RobotSimscapeDecentralizedPID')

t = [];

color = jet(1+ round(trial_time/stepsize));
x = [];
y = [];
z = [];
error_x = [];
error_y = [];
error_z = [];


for i = 0:stepsize:trial_time - stepsize
    t = [t; i];
    j = round(i/stepsize);
    x = [x; l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2))];
    y = [y; l1_*sin(theta_out(j+1,1)) + l2_*sin(theta_out(j
        +1,1) + theta_out(j+1,2))];
    z = [z; l0_ - l4_ - theta_out(j+1,3)];
    error_x = [error_x; x_d(j+1) - x(length(x))];
    error_y = [error_y; y_d(j+1) - y(length(y))];
    error_z = [error_z; z_d(j+1) - z(length(z))];
    plot3([l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
```

```
        +1,1) + theta_out(j+1,2)), l1_*cos(theta_out(j+2,1)) +
         l2_*cos(theta_out(j+2,1) + theta_out(j+2,2))],[l1_*
        sin(theta_out(j+1,1)) + l2_*sin(theta_out(j+1,1) +
        theta_out(j+1,2)),l1_*sin(theta_out(j+2,1)) + l2_*sin(
        theta_out(j+2,1) + theta_out(j+2,2))],[l0_ - l4_ -
        theta_out(j+1,3),l0_ - l4_ - theta_out(j+2,3)],'color'
        ,color(j+1,:),'linewidth',3)
    %scatter3(l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2)),l1_*sin(theta_out(j+1,1)) +
        l2_*sin(theta_out(j+1,1) + theta_out(j+1,2)),l0_ - l4_
         - theta_out(j+1,3),10,color(j+1,:),'filled');
    hold on
end

error_1 = [];
error_2 = [];
error_3 = [];
error_4 = [];
error_1_simscape = [];
error_2_simscape = [];
error_3_simscape = [];
error_4_simscape = [];

for i = 1:1:length(theta_out(:,1))-1
    error_1 = [error_1; theta_out(i+1,1) - theta_1_d(i,2)];
    error_2 = [error_2; theta_out(i+1,2) - theta_2_d(i,2)];
    error_3 = [error_3; theta_out(i+1,3) - d_3_d(i,2)];
    error_4 = [error_4; theta_out(i+1,4) - theta_4_d(i,2)];
end

for i = 1:1:length(theta_out_simscape(:,1))-1
    error_1_simscape = [error_1_simscape; theta_out_simscape(
        i+1,1) - theta_1_d(i,2)];
    error_2_simscape = [error_2_simscape; theta_out_simscape(
        i+1,2) - theta_2_d(i,2)];
    error_3_simscape = [error_3_simscape; theta_out_simscape(
        i+1,3) - d_3_d(i,2)];
```

```
    error_4_simscape = [error_4_simscape; theta_out_simscape(
        i+1,4) - theta_4_d(i,2)];
end

scatter3(x(1),y(1),z(1),50,color(1,:),'d','filled','
    markeredgecolor','w')
scatter3(x(round(trial_time/stepsize)),y(round(trial_time/
    stepsize)),z(round(trial_time/stepsize)),50,color(round(
    trial_time/stepsize),:),'d','filled','markeredgecolor','w'
    )
set(gca,'DataAspectRatio',[1 1 1])
set(gcf,'Position',[150 50 1250 900])
hold off

t2 = [t ; trial_time];

figure(2)
subplot(4,2,1)
plot(t2, theta_out(:,1))
legend('Desired Angle','Actual Angle')

subplot(4,2,3)
plot(t2, theta_out(:,2))
legend('Desired Angle','Actual Angle')

subplot(4,2,5)
plot(t2, theta_out(:,3))
% plot(t,z_base)
legend('Desired Position','Actual Position')%,'Non-Rigid
    Surface Height')%

subplot(4,2,7)
plot(t2, theta_out(:,4))
legend('Desired Angle','Actual Angle')

tau_maxes = [max(tau_out(:,1)) , max(tau_out(:,2)) , max(
    tau_out(:,3)) , max(tau_out(:,4))];
```

```
tau_mins = [min(tau_out(:,1)) , min(tau_out(:,2)) , min(
    tau_out(:,3)) , min(tau_out(:,4))];
tau_diffs = tau_maxes- tau_mins;

error_maxes = [max(error_1) , max(error_2) , max(error_3) ,
    max(error_4)];
error_mins = [min(error_1) , min(error_2) , min(error_3) ,
    min(error_4)];
error_diffs = error_maxes - error_mins;

upperylims = [ max([tau_maxes(1)/tau_diffs(1),error_maxes(1)/
    error_diffs(1),0]) + 0.1 , max([tau_maxes(2)/tau_diffs(2),
    error_maxes(2)/error_diffs(2),0]) + 0.1, max([tau_maxes(3)
    /tau_diffs(3),error_maxes(3)/error_diffs(3),0]) + 0.1, max
    ([tau_maxes(4)/tau_diffs(4),error_maxes(4)/error_diffs(4)
    ,0]) + 0.1];
lowerylims = [ min([tau_mins(1)/tau_diffs(1),error_mins(1)/
    error_diffs(1),0]) - 0.1, min([tau_mins(2)/tau_diffs(2),
    error_mins(2)/error_diffs(2),0]) - 0.1, min([tau_mins(3)/
    tau_diffs(3),error_mins(3)/error_diffs(3),0]) - 0.1, min([
    tau_mins(4)/tau_diffs(4),error_mins(4)/error_diffs(4),0])
    - 0.1];

subplot(4,2,2)
yyaxis left
plot(t, error_1,'r')
xlabel('Time (seconds)')
ylabel('Error (rad)')
set(gca,'ycolor','r')
ylim([lowerylims(1)*error_diffs(1),upperylims(1)*error_diffs
    (1)])
yyaxis right
plot(t2, tau_out(:,1),'g')
ylabel('\tau_1 (N m)')
set(gca,'ycolor','g')
ylim([lowerylims(1)*tau_diffs(1),upperylims(1)*tau_diffs(1)])
title('Joint 1 Error and Torque Command')
```

```
legend('Joint Error','Command Torque')
grid on
hold on

subplot(4,2,4)
yyaxis left
plot(t, error_2,'r')
xlabel('Time (seconds)')
ylabel('Error (rad)')
set(gca,'ycolor','r')
ylim([lowerylims(2)*error_diffs(2),upperylims(2)*error_diffs
    (2)])
yyaxis right
plot(t2, tau_out(:,2),'g')
ylabel('\tau_2 (N m)')
set(gca,'ycolor','g')
ylim([lowerylims(2)*tau_diffs(2),upperylims(2)*tau_diffs(2)])
title('Joint 2 Error and Torque Command')
legend('Joint Error','Command Torque')
grid on
hold on

subplot(4,2,6)
yyaxis left
plot(t, error_3,'r')
xlabel('Time (seconds)')
ylabel('Error (m)')
set(gca,'ycolor','r')
ylim([lowerylims(3)*error_diffs(3),upperylims(3)*error_diffs
    (3)])
yyaxis right
plot(t2, tau_out(:,3),'g')
ylabel('\tau_3 (N)')
set(gca,'ycolor','g')
ylim([lowerylims(3)*tau_diffs(3),upperylims(3)*tau_diffs(3)
    +0.000000001])
title('Joint 3 Error and Torque Command')
```

```
legend('Joint Error','Command Torque')
grid on
hold on

subplot(4,2,8)
yyaxis left
plot(t, error_4,'r')
xlabel('Time (seconds)')
ylabel('Error (rad)')
set(gca,'ycolor','r')
ylim([lowerylims(4)*error_diffs(4),upperylims(4)*error_diffs
    (4)])
yyaxis right
plot(t2, tau_out(:,4),'g')
ylabel('\tau_4 (N m)')
set(gca,'ycolor','g')
ylim([lowerylims(4)*tau_diffs(4),upperylims(4)*tau_diffs(4)])
title('Joint 4 Error and Torque Command')
legend('Joint Error','Command Torque')
grid on
hold on

set(gcf,'Position',[100 50 1250 900])

figure(3)
subplot(4,1,1)
plot(t,theta_1_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_1 (radians)')
title('Desired and Actual Angle for Joint 1')
plot(t2, theta_out(:,1))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,2)
plot(t,theta_2_d(:,2))
```

```
hold on
xlabel('Time (seconds)')
ylabel('\theta_2 (radians)')
title('Desired and Actual Angle for Joint 2')
plot(t2, theta_out(:,2))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,3)
plot(t,d_3_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('d_3 (metres)')
title('Desired and Actual Position for Joint 3')
plot(t2, theta_out(:,3))
legend('Desired Position','Actual Position')
hold off

subplot(4,1,4)
plot(t,theta_4_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_4 (radians)')
title('Desired and Actual Angle for Joint 4')
plot(t2, theta_out(:,4))
legend('Desired Angle','Actual Angle')
hold off

set(gcf,'Position',[50 50 800 900])

combined_error_maxes = [max(error_1), max(error_2), max(
    error_3), max(error_4), max(error_1_simscape), max(
    error_2_simscape), max(error_3_simscape), max(
    error_4_simscape)];
combined_error_mins = [min(error_1), min(error_2), min(
    error_3), min(error_4), min(error_1_simscape), min(
    error_2_simscape), min(error_3_simscape), min(
```

```
    error_4_simscape)];
combined_error_diffs = max(combined_error_maxes) - min(
    combined_error_mins);

figure(4)

plot(t, error_1,'color',[1 0.563 0])
hold on
plot(t, error_2,'b')
plot(t, error_3,'color',[0 0.5 0])
plot(t, error_4,'color',[0.781 0 1])
plot(t, error_1_simscape,'--','color',[1 0.563 0])
plot(t, error_2_simscape,'b--')
plot(t, error_3_simscape,'--','color',[0 0.5 0])
plot(t, error_4_simscape,'--','color',[0.781 0 1])
xlabel('Time (seconds)')
ylabel('Error (rad or m)')
xlim([0,2])
title('Joint Errors')
legend('Joint 1 Error','Joint 2 Error','Joint 3 Error','Joint
    4 Error','Joint 1 Error (SimScape)','Joint 2 Error (
    SimScape)','Joint 3 Error (SimScape)','Joint 4 Error (
    SimScape)')
grid on
hold off

%% Run the Decentralized PID with Feedback Linearization
    Controller and plot the results

Kp2 = [781.66; 271.22; 1192.9; 0.605];
Kd2 = [87.21; 30.26; 133.1; 0.0675];
Ki2 = [2325.6; 806.9; 3549.03; 1.8];

sim('RobotSimFLDecentralizedPID')
sim('RobotSimscapeFLDecentralizedPID')

figure(5)
```

```
plot3(x_d,y_d,z_d,'color','black','linewidth',2);
hold on

t = [];

color = jet(1+ round(trial_time/stepsize));
x = [];
y = [];
z = [];
error_x = [];
error_y = [];
error_z = [];


for i = 0:stepsize:trial_time - stepsize
    t = [t; i];
    j = round(i/stepsize);
    x = [x; l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2))];
    y = [y; l1_*sin(theta_out(j+1,1)) + l2_*sin(theta_out(j
        +1,1) + theta_out(j+1,2))];
    z = [z; l0_ - l4_ - theta_out(j+1,3)];
    error_x = [error_x; x_d(j+1) - x(length(x))];
    error_y = [error_y; y_d(j+1) - y(length(y))];
    error_z = [error_z; z_d(j+1) - z(length(z))];
    plot3([l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2)), l1_*cos(theta_out(j+2,1)) +
         l2_*cos(theta_out(j+2,1) + theta_out(j+2,2))],[l1_*
        sin(theta_out(j+1,1)) + l2_*sin(theta_out(j+1,1) +
        theta_out(j+1,2)),l1_*sin(theta_out(j+2,1)) + l2_*sin(
        theta_out(j+2,1) + theta_out(j+2,2))],[l0_ - l4_ -
        theta_out(j+1,3),l0_ - l4_ - theta_out(j+2,3)],'color'
        ,color(j+1,:),'linewidth',3)
    %scatter3(l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2)),l1_*sin(theta_out(j+1,1)) +
        l2_*sin(theta_out(j+1,1) + theta_out(j+1,2)),l0_ - l4_
```

```
        - theta_out(j+1,3),10,color(j+1,:),'filled');
    hold on
end

error_1 = [];
error_2 = [];
error_3 = [];
error_4 = [];
error_1_simscape = [];
error_2_simscape = [];
error_3_simscape = [];
error_4_simscape = [];

for i = 1:1:length(theta_out(:,1))-1
    error_1 = [error_1; theta_out(i+1,1) - theta_1_d(i,2)];
    error_2 = [error_2; theta_out(i+1,2) - theta_2_d(i,2)];
    error_3 = [error_3; theta_out(i+1,3) - d_3_d(i,2)];
    error_4 = [error_4; theta_out(i+1,4) - theta_4_d(i,2)];
end

for i = 1:1:length(theta_out_simscape(:,1))-1
    error_1_simscape = [error_1_simscape; theta_out_simscape(
        i+1,1) - theta_1_d(i,2)];
    error_2_simscape = [error_2_simscape; theta_out_simscape(
        i+1,2) - theta_2_d(i,2)];
    error_3_simscape = [error_3_simscape; theta_out_simscape(
        i+1,3) - d_3_d(i,2)];
    error_4_simscape = [error_4_simscape; theta_out_simscape(
        i+1,4) - theta_4_d(i,2)];
end

scatter3(x(1),y(1),z(1),50,color(1,:),'d','filled','
    markeredgecolor','w')
scatter3(x(round(trial_time/stepsize)),y(round(trial_time/
    stepsize)),z(round(trial_time/stepsize)),50,color(round(
    trial_time/stepsize),:),'d','filled','markeredgecolor','w'
    )
```

```
set(gca,'DataAspectRatio',[1 1 1])
set(gcf,'Position',[200 50 1250 900])
hold off

figure(6)
subplot(4,1,1)
plot(t,theta_1_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_1 (radians)')
title('Desired and Actual Angle for Joint 1')
plot(t2, theta_out(:,1))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,2)
plot(t,theta_2_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_2 (radians)')
title('Desired and Actual Angle for Joint 2')
plot(t2, theta_out(:,2))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,3)
plot(t,d_3_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('d_3 (metres)')
title('Desired and Actual Position for Joint 3')
plot(t2, theta_out(:,3))
legend('Desired Position','Actual Position')
hold off

subplot(4,1,4)
plot(t,theta_4_d(:,2))
```

```
hold on
xlabel('Time (seconds)')
ylabel('\theta_4 (radians)')
title('Desired and Actual Angle for Joint 4')
plot(t2, theta_out(:,4))
legend('Desired Angle','Actual Angle')
hold off

set(gcf,'Position',[250 50 800 900])

combined_error_maxes = [max(error_1), max(error_2), max(
    error_3), max(error_4), max(error_1_simscape), max(
    error_2_simscape), max(error_3_simscape), max(
    error_4_simscape)];
combined_error_mins = [min(error_1), min(error_2), min(
    error_3), min(error_4), min(error_1_simscape), min(
    error_2_simscape), min(error_3_simscape), min(
    error_4_simscape)];
combined_error_diffs = max(combined_error_maxes) - min(
    combined_error_mins);

figure(7)

plot(t, error_1,'color',[1 0.563 0])
hold on
plot(t, error_2,'b')
plot(t, error_3,'color',[0 0.5 0])
plot(t, error_4,'color',[0.781 0 1])
plot(t, error_1_simscape,'--','color',[1 0.563 0])
plot(t, error_2_simscape,'b--')
plot(t, error_3_simscape,'--','color',[0 0.5 0])
plot(t, error_4_simscape,'--','color',[0.781 0 1])
xlabel('Time (seconds)')
ylabel('Error (rad or m)')
xlim([0,2])
title('Joint Errors')
legend('Joint 1 Error','Joint 2 Error','Joint 3 Error','Joint
```

```
    _4_Error','Joint_1_Error_(SimScape)','Joint_2_Error_(
    SimScape)','Joint_3_Error_(SimScape)','Joint_4_Error_(
    SimScape)')
grid on
hold off



%% Implement and plot results for full state feedback
    linearization controller

M_linear_inv = [0.4739 , -0.4739 , 0 , 0 ; -0.4739 , 1.3688 ,
     0 , 0.8949 ; 0 , 0 , 0.2024 , 0 ; 0 , 0.8949 , 0 ,
    400.8949];
A = [zeros(4) , eye(4) ; zeros(4) , zeros(4)];
B = [zeros(4) ; M_linear_inv];
K = place(A,B,[-8.0, -8.1, -8.2, -8.3, -8.4, -8.5, -8.6,
    -8.7])
Q = [ eye(4) , zeros(4); zeros(4) , eye(4)];
R = eye(4);
K = lqr(A,B,Q,R)

sim('RobotSimFLCentralizedPID')
sim('RobotSimscapeFLCentralizedPID')

figure(8)

plot3(x_d,y_d,z_d,'color','black','linewidth',2);
hold on

t = [];

color = jet(1+ round(trial_time/stepsize));
x = [];
y = [];
z = [];
error_x = [];
```

```
error_y = [];
error_z = [];


for i = 0:stepsize:trial_time - stepsize
    t = [t; i];
    j = round(i/stepsize);
    x = [x; l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2))];
    y = [y; l1_*sin(theta_out(j+1,1)) + l2_*sin(theta_out(j
        +1,1) + theta_out(j+1,2))];
    z = [z; l0_ - l4_ - theta_out(j+1,3)];
    error_x = [error_x; x_d(j+1) - x(length(x))];
    error_y = [error_y; y_d(j+1) - y(length(y))];
    error_z = [error_z; z_d(j+1) - z(length(z))];
    plot3([l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2)), l1_*cos(theta_out(j+2,1)) +
         l2_*cos(theta_out(j+2,1) + theta_out(j+2,2))],[l1_*
        sin(theta_out(j+1,1)) + l2_*sin(theta_out(j+1,1) +
        theta_out(j+1,2)),l1_*sin(theta_out(j+2,1)) + l2_*sin(
        theta_out(j+2,1) + theta_out(j+2,2))],[l0_ - l4_ -
        theta_out(j+1,3),l0_ - l4_ - theta_out(j+2,3)],'color'
        ,color(j+1,:),'linewidth',3)
    %scatter3(l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2)),l1_*sin(theta_out(j+1,1)) +
        l2_*sin(theta_out(j+1,1) + theta_out(j+1,2)),l0_ - l4_
         - theta_out(j+1,3),10,color(j+1,:),'filled');
    hold on
end

error_1 = [];
error_2 = [];
error_3 = [];
error_4 = [];
error_1_simscape = [];
error_2_simscape = [];
error_3_simscape = [];
```

```
error_4_simscape = [];

for i = 1:1:length(theta_out(:,1))-1
    error_1 = [error_1; theta_out(i+1,1) - theta_1_d(i,2)];
    error_2 = [error_2; theta_out(i+1,2) - theta_2_d(i,2)];
    error_3 = [error_3; theta_out(i+1,3) - d_3_d(i,2)];
    error_4 = [error_4; theta_out(i+1,4) - theta_4_d(i,2)];
end

for i = 1:1:length(theta_out_simscape(:,1))-1
    error_1_simscape = [error_1_simscape; theta_out_simscape(
        i+1,1) - theta_1_d(i,2)];
    error_2_simscape = [error_2_simscape; theta_out_simscape(
        i+1,2) - theta_2_d(i,2)];
    error_3_simscape = [error_3_simscape; theta_out_simscape(
        i+1,3) - d_3_d(i,2)];
    error_4_simscape = [error_4_simscape; theta_out_simscape(
        i+1,4) - theta_4_d(i,2)];
end

scatter3(x(1),y(1),z(1),50,color(1,:),'d','filled','
    markeredgecolor','w')
scatter3(x(round(trial_time/stepsize)),y(round(trial_time/
    stepsize)),z(round(trial_time/stepsize)),50,color(round(
    trial_time/stepsize),:),'d','filled','markeredgecolor','w'
    )
set(gca,'DataAspectRatio',[1 1 1])
set(gcf,'Position',[300 50 1250 900])
hold off

figure(9)
subplot(4,1,1)
plot(t,theta_1_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_1 (radians)')
title('Desired and Actual Angle for Joint 1')
```

```
plot(t2, theta_out(:,1))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,2)
plot(t,theta_2_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_2 (radians)')
title('Desired and Actual Angle for Joint 2')
plot(t2, theta_out(:,2))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,3)
plot(t,d_3_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('d_3 (metres)')
title('Desired and Actual Position for Joint 3')
plot(t2, theta_out(:,3))
legend('Desired Position','Actual Position')
hold off

subplot(4,1,4)
plot(t,theta_4_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_4 (radians)')
title('Desired and Actual Angle for Joint 4')
plot(t2, theta_out(:,4))
legend('Desired Angle','Actual Angle')
hold off

set(gcf,'Position',[350 50 800 900])

combined_error_maxes = [max(error_1) , max(error_2) , max(
```

```
    error_3) , max(error_4) , max(error_1_simscape) , max(
    error_2_simscape) , max(error_3_simscape) , max(
    error_4_simscape)];
combined_error_mins = [min(error_1) , min(error_2)  , min(
    error_3) , min(error_4) , min(error_1_simscape) , min(
    error_2_simscape) , min(error_3_simscape) , min(
    error_4_simscape)];
combined_error_diffs = max(combined_error_maxes) - min(
    combined_error_mins);

figure(10)

plot(t, error_1,'color',[1 0.563 0])
hold on
plot(t, error_2,'b')
plot(t, error_3,'color',[0 0.5 0])
plot(t, error_4,'color',[0.781 0 1])
plot(t, error_1_simscape,'--','color',[1 0.563 0])
plot(t, error_2_simscape,'b--')
plot(t, error_3_simscape,'--','color',[0 0.5 0])
plot(t, error_4_simscape,'--','color',[0.781 0 1])
xlabel('Time (seconds)')
ylabel('Error (rad or m)')
xlim([0,2])
title('Joint Errors')
legend('Joint 1 Error','Joint 2 Error','Joint 3 Error','Joint
    4 Error','Joint 1 Error (SimScape)','Joint 2 Error (
    SimScape)','Joint 3 Error (SimScape)','Joint 4 Error (
    SimScape)')
grid on
hold off

toc
```

B.3 Hybrid Force-Position Control Simulation

```
tic

close all

%% Establish simulation parameters using the following values
    :

g__ = -9.8;
l0_ = 0.5;
l1_ = 0.4;
l2_ = 0.4;
l4_ = 0.15;
m1_ = 6.01;
m2_ = 5.37;
m3_ = 4.03;
m4_ = 0.91;
I1zz_ = 0.1807;
I2zz_ = 0.1558;
I3zz_ = 0.0064;
I4zz_ = 0.0025;
lc1x_ = -0.185;
lc2x_ = -0.224;
lc3x_ = 0;
lc4x_ = 0;
lc1y_ = 0;
lc2y_ = 0;
lc3y_ = 0;
lc4y_ = 0;
lc1z_ = 0;
lc2z_ = 0;
lc3z_ = -0.201;
lc4z_ = -0.122;

theta1upperlimit = pi/2;
theta1lowerlimit = -theta1upperlimit;
theta2upperlimit = 7*pi/8;
theta2lowerlimit = -theta2upperlimit;
```

```
d3upperlimit = 0.35;
d3lowerlimit = 0.1;
theta4upperlimit = pi;
theta4lowerlimit = -theta4upperlimit;

joint1motorlimit = 1000;
joint2motorlimit = 1000;
joint3motorlimit = 1000;
joint4motorlimit = 1000;

M_inv = subs(M_inv,{g, l0, l1, l2, l4, m1, m2, m3, m4, I1zz,
    I2zz, I3zz, I4zz, lc1x, lc2x, lc3x, lc4x, lc1y, lc2y, lc3y
    , lc4y},{g__, l0_, l1_, l2_, l4_, m1_, m2_, m3_, m4_,
    I1zz_, I2zz_, I3zz_, I4zz_, lc1x_, lc2x_, lc3x_, lc4x_,
    lc1y_, lc2y_, lc3y_, lc4y_});
C = subs(C_matrix,{g, l0, l1, l2, l4, m1, m2, m3, m4, I1zz,
    I2zz, I3zz, I4zz, lc1x, lc2x, lc3x, lc4x, lc1y, lc2y, lc3y
    , lc4y},{g__, l0_, l1_, l2_, l4_, m1_, m2_, m3_, m4_,
    I1zz_, I2zz_, I3zz_, I4zz_, lc1x_, lc2x_, lc3x_, lc4x_,
    lc1y_, lc2y_, lc3y_, lc4y_});
G = subs(G,{g_, l0, l1, l2, l4, m1, m2, m3, m4, I1zz, I2zz,
    I3zz, I4zz, lc1x, lc2x, lc3x, lc4x, lc1y, lc2y, lc3y, lc4y
    },{g__, l0_, l1_, l2_, l4_, m1_, m2_, m3_, m4_, I1zz_,
    I2zz_, I3zz_, I4zz_, lc1x_, lc2x_, lc3x_, lc4x_, lc1y_,
    lc2y_, lc3y_, lc4y_});

%% First we plot the workspace of the robot

zz1 = [];
zz2 = [];

n = 50;

cornerr = sqrt((l1_*cos(theta1upperlimit) + l2_*cos(
    theta1upperlimit+theta2upperlimit))^2 + (l1_*sin(
    theta1upperlimit) + l2_*sin(theta1upperlimit+
    theta2upperlimit))^2 );
```

```
cornerangle = acos(l1_*cos(theta1upperlimit) + l2_*cos(
    theta1upperlimit+theta2upperlimit)/cornerr);

surface1theta = theta1lowerlimit:(theta1upperlimit-
    theta1lowerlimit)/n:theta1upperlimit;
surface1r = cornerr:((l1_ + l2_)-cornerr)/n:l1_ + l2_;

xx2 = zeros(n+1,n+1);
yy2 = zeros(n+1,n+1);

for i = 1:n+1
    angle1 = theta1upperlimit + (cornerangle -
        theta1upperlimit)*(i-1)/n;
    angle2 = (cornerangle - theta1upperlimit)*(i-1)/n;
    radius = sqrt((l1_*cos(angle2))^2 - l1_^2 +l2_^2) + l1_*
        cos(angle2);
    for j = 1:n+1
        thickness = (n+1-j)/n;
        xx2(i,j) = (cornerr+(radius-cornerr)*thickness)*cos(
            angle1);
        yy2(i,j) = (cornerr+(radius-cornerr)*thickness)*sin(
            angle1);
    end
end

xx3 = zeros(n+1,n+1);
yy3 = zeros(n+1,n+1);

for i = 1:n+1
    angle1 = theta1lowerlimit + (-cornerangle -
        theta1lowerlimit)*(i-1)/n;
    angle2 = (-cornerangle - theta1lowerlimit)*(i-1)/n;
    radius = sqrt((l1_*cos(angle2))^2 - l1_^2 +l2_^2) + l1_*
        cos(angle2);
    for j = 1:n+1
        thickness = (n+1-j)/n;
        xx3(i,j) = (cornerr+(radius-cornerr)*thickness)*cos(
```

```
            angle1);
        yy3(i,j) = (cornerr+(radius-cornerr)*thickness)*sin(
            angle1);
    end
end

[thetatheta,rr] = meshgrid([surface1theta],[surface1r]);

xx1 = rr.*cos(thetatheta);
yy1 = rr.*sin(thetatheta);

z4 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
   n:l0_ - l4_ - d3lowerlimit;
x4 = (l1_+l2_)*cos(surface1theta);
y4 = (l1_+l2_)*sin(surface1theta);

yy4 = [];
for i = 1:n+1
    yy4 = [yy4;y4];
end

[xx4, zz4] = meshgrid([x4],[z4]);

z5 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
   n:l0_ - l4_ - d3lowerlimit;
x5 = l1_*cos(theta1upperlimit) + l2_*cos(theta1upperlimit:
   theta2upperlimit/n:theta1upperlimit+theta2upperlimit);
y5 = l1_*sin(theta1upperlimit) + l2_*sin(theta1upperlimit:
   theta2upperlimit/n:theta1upperlimit+theta2upperlimit);

yy5 = [];
for i = 1:n+1
    yy5 = [yy5;y5];
end

[xx5, zz5] = meshgrid([x5],[z5]);
```

```
z6 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
    n:l0_ - l4_ - d3lowerlimit;
x6 = l1_*cos(theta1lowerlimit) + l2_*cos(theta1lowerlimit:
    theta2lowerlimit/n:theta1lowerlimit+theta2lowerlimit);
y6 = l1_*sin(theta1lowerlimit) + l2_*sin(theta1lowerlimit:
    theta2lowerlimit/n:theta1lowerlimit+theta2lowerlimit);

yy6 = [];
for i = 1:n+1
    yy6 = [yy6;y6];
end

[xx6, zz6] = meshgrid([x6],[z6]);

z7 =  l0_ - l4_ - d3upperlimit:(d3upperlimit - d3lowerlimit)/
    n:l0_ - l4_ - d3lowerlimit;
x7 = (cornerr)*cos(linspace(-cornerangle,cornerangle,n+1));
y7 = (cornerr)*sin(linspace(-cornerangle,cornerangle,n+1));

yy7 = [];
for i = 1:n+1
    yy7 = [yy7;y7];
end

[xx7, zz7] = meshgrid([x7],[z7]);

grey = [0.1,0.1,0.1];

for i = 1:n+1
    zzz = [];
    for j = 1:n+1
        zzz = [zzz, l0_ - l4_ - d3lowerlimit];
    end
    zz1 = [zz1;zzz];
end

for i = 1:n+1
```

```
    zzz = [];
    for j = 1:n+1
        zzz = [zzz, l0_ - l4_ - d3upperlimit];
    end
    zz2 = [zz2;zzz];
end

figure(1)
surf(xx1,yy1,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx1,yy1,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx3,yy3,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx3,yy3,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
```

```
    'none');
get(gcf, 'Renderer');
hold on
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x6,y6,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x6,y6,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
```

```
plot3(x7,y7,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x7,y7,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3([cornerr*cos(cornerangle),cornerr*cos(cornerangle)],[
    cornerr*sin(cornerangle),cornerr*sin(cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on
plot3([cornerr*cos(-cornerangle),cornerr*cos(-cornerangle)],[
    cornerr*sin(-cornerangle),cornerr*sin(-cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on

figure(5)
surf(xx1,yy1,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx1,yy1,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx2,yy2,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx3,yy3,zz1,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
```

```
hold on
surf(xx3,yy3,zz2,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx4,yy4,zz4,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx5,yy5,zz5,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
surf(xx6,yy6,zz6,'FaceColor',grey,'FaceAlpha',.1,'EdgeColor',
    'none');
get(gcf, 'Renderer');
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x4,y4,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
```

```
hold on
plot3(x5,y5,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x6,y6,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x6,y6,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3(x7,y7,linspace(l0_ - l4_ - d3upperlimit,l0_ - l4_ -
    d3upperlimit,n+1),'black')
hold on
plot3(x7,y7,linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ -
    d3lowerlimit,n+1),'black')
hold on
plot3([cornerr*cos(cornerangle),cornerr*cos(cornerangle)],[
    cornerr*sin(cornerangle),cornerr*sin(cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on
plot3([cornerr*cos(-cornerangle),cornerr*cos(-cornerangle)],[
    cornerr*sin(-cornerangle),cornerr*sin(-cornerangle)],
    linspace(l0_ - l4_ - d3lowerlimit,l0_ - l4_ - d3upperlimit
    ,2),'black')
hold on

%% Generate the reaction surface (rs) that the robot will
    interact with

rs_ymax = 0.4;
rs_ymin = -0.4;
rs_xmax = 0.6;
rs_xmin = 0.2;
rs_zbase = 0.13;

rs_x = linspace(rs_xmin,rs_xmax,n+1);
```

```
rs_y = linspace(rs_ymin,rs_ymax,n+1);

rs_zz = [];

for i = 1:n+1
    rs_z = [];
    for j = 1:n+1
        rs_z = [rs_z, rs_zbase];
    end
    rs_zz = [rs_zz;rs_z];
end

[rs_xx, rs_yy] = meshgrid([rs_x],[rs_y]);

%% Plot the reaction surface

figure(1)
surf(rs_xx,rs_yy,rs_zz,'FaceColor',[1, 1, 0],'FaceAlpha',.8,'
    EdgeColor','none');
get(gcf, 'Renderer');
hold on
plot3(rs_x,linspace(rs_ymax,rs_ymax,n+1),rs_zz(n+1,:),'black'
    )
hold on
plot3(rs_x,linspace(rs_ymin,rs_ymin,n+1),rs_zz(1,:),'black')
hold on
plot3(linspace(rs_xmax,rs_xmax,n+1),rs_y,rs_zz(:,n+1),'black'
    )
hold on
plot3(linspace(rs_xmin,rs_xmin,n+1),rs_y,rs_zz(:,1),'black')
hold on

%% Generate the desired trajectory and the corresponding
    joint angle timeseries using the inverse kinematics, as
    well as the desired normal and friction forces:

trial_time = 10;
```

```
stepsize = 0.005;

x_d = [];
y_d = [];
z_d = [];
zo_d = [];
theta_1_d = [];
theta_2_d = [];
theta_1_d1 = [];
theta_2_d1 = [];
theta_4_d1 = [];
theta_1_d2 = [];
theta_2_d2 = [];
theta_4_d2 = [];
d_3_d = [];
z_base = [];
theta_4_d = [];
fext_x_d = [];
fext_y_d = [];
fext_z_d = [];
fext_rx_d = [];
fext_ry_d = [];
fext_rz_d = [];
t = [];
DOFs = [];
spline = [];

for i = 0:stepsize:trial_time - stepsize
    t = [t; i];
    j = i/trial_time;
%     Line from starting position to surface
%     x__d = l1_ + l2_ - j*(l1_ + l2_ - (nrs_xmin + nrs_xmax)
    /2);
%     y__d = j*(nrs_ymin + 0.1*(nrs_ymax - nrs_ymin));
%     z__d = l0_ - l4_ - d3lowerlimit - j*(l0_ - l4_ -
    d3lowerlimit - nrs_zz(n/10,n/2));
%     Squiggle within single area
```

```
%      x__d = 0.5 + j*0.2;
%      y__d = -0.2 + j*0.4;
%      z__d = 0.05 + j*0.15 + 0.025*sin(j*10*pi);
%      Squiggle within two areas
%      x__d = 0.1 + j*0.4;
%      y__d = -0.4 + j*0.6;
%      z__d = 0.05 + j*0.15 + 0.025*sin(j*10*pi);
%     Squiggle within three areas
%      x__d = 0.25;
%      y__d = -0.4 + j*0.8;
%      z__d = 0.05 + j*0.15 + 0.025*sin(j*10*pi);
%      RS test line
    if j < 0.2
        x__d = 0.8 - j*1.5;
        y__d = -j*1.5;
        z__d = 0.25 - j*(0.25 - rs_zbase)/0.2;
    else
        x__d = 0.5;
        y__d = -0.3 + (j-0.2)*0.75;
        z__d = rs_zbase - 0.025*sin((j-0.2)*10*pi);
    end
%      Position Only Controller Test Line;
%      x__d = 0.8 - 0.4*j - 0.05*sin(j*4*pi);
%      y__d = -0.2*j + 0.15*sin(j*1.5*pi);
%      z__d = 0.25 - 0.1*j - 0.005*sin(j*15*pi);
    fext_x__d = 0;
    fext_y__d = 0;
    fext_rx__d = 0;
    fext_ry__d = 0;
    fext_rz__d = 0;
    if j < 0.2
        fext_z__d = 0;
    else
        fext_z__d = -200 -200*sin(j*10);
%         fext_z__d = -200;
    end
    spline = [spline; x__d y__d z__d];
```

```
zo__d = 0;
x_d = [x_d; x__d];
y_d = [y_d; y__d];
z_d = [z_d; z__d];
zo_d = [zo_d; zo__d];
theta_2__d1 = acos((x__d^2 + y__d^2 - l1_^2 - l2_^2)/(2*
    l1_*l2_));
theta_2__d2 = -acos((x__d^2 + y__d^2 - l1_^2 - l2_^2)/(2*
    l1_*l2_));
theta_1__d1 = atan2(y__d,x__d) - asin(l2_*sin(theta_2__d1
    )/(sqrt(x__d^2 + y__d^2)));
theta_1__d2 = atan2(y__d,x__d) - asin(l2_*sin(theta_2__d2
    )/(sqrt(x__d^2 + y__d^2)));
d_3__d = -z__d + l0_ - l4_;
theta_4__d1 = zo__d - theta_1__d1 - theta_2__d1;
theta_4__d2 = zo__d - theta_1__d2 - theta_2__d2;
theta_2__d1check = or(theta_2__d1 < theta2lowerlimit,
    theta_2__d1 > theta2upperlimit);
theta_2__d2check = or(theta_2__d2 < theta2lowerlimit,
    theta_2__d2 > theta2upperlimit);
theta_1__d1check = or(theta_1__d1 < theta1lowerlimit,
    theta_1__d1 > theta1upperlimit);
theta_1__d2check = or(theta_1__d2 < theta1lowerlimit,
    theta_1__d2 > theta1upperlimit);
if or(theta_2__d1check,theta_1__d1check) == 0
    if or(theta_2__d2check,theta_1__d2check) == 0
        DOFs = [DOFs; 2];
        theta_1__d = theta_1__d1;
        theta_2__d = theta_2__d1;
        theta_4__d = theta_4__d1;
    else
        DOFs = [DOFs; 1.1];
        theta_1__d = theta_1__d1;
        theta_2__d = theta_2__d1;
        theta_4__d = theta_4__d1;
    end
else
```

```
        if or(theta_2__d2check,theta_1__d2check) == 0
            DOFs = [DOFs; 1.2];
            theta_1__d = theta_1__d2;
            theta_2__d = theta_2__d2;
            theta_4__d = theta_4__d2;
        else
            DOFs = [DOFs; 0];
            theta_1__d = theta_1__d1;
            theta_2__d = theta_2__d1;
            theta_4__d = theta_4__d1;
        end
    end
    theta_1_d = [theta_1_d; theta_1__d];
    theta_2_d = [theta_2_d; theta_2__d];
    theta_4_d = [theta_4_d; theta_4__d];
    theta_1_d1 = [theta_1_d1; theta_1__d1];
    theta_2_d1 = [theta_2_d1; theta_2__d1];
    theta_4_d1 = [theta_4_d1; theta_4__d1];
    theta_1_d2 = [theta_1_d2; theta_1__d2];
    theta_2_d2 = [theta_2_d2; theta_2__d2];
    theta_4_d2 = [theta_4_d2; theta_4__d2];
    d_3_d = [d_3_d; d_3__d];
    z_base = [z_base; l0_ - l4_ - rs_zbase];
    fext_x_d = [fext_x_d; fext_x__d];
    fext_y_d = [fext_y_d; fext_y__d];
    fext_z_d = [fext_z_d; fext_z__d];
    fext_rx_d = [fext_rx_d; fext_rx__d];
    fext_ry_d = [fext_ry_d; fext_ry__d];
    fext_rz_d = [fext_rz_d; fext_rz__d];
end

if sum(DOFs == 0) > 0
    disp('The desired trajectory is not fully enclosed in the
        reachable workspace')
else
    if and(sum(DOFs == 1.1) > 0,sum(DOFs == 1.2) > 0 )
        disp('The desired trajectory has at least one
```

```
            inflection_point')
    else
        if sum(DOFs == 1.1) > 0
            theta_1_d = theta_1_d1;
            theta_2_d = theta_2_d1;
        end
        if sum(DOFs == 1.2) > 0
            theta_1_d = theta_1_d2;
            theta_2_d = theta_2_d2;
        end
    end
end

theta_1_d = [t,theta_1_d];
theta_2_d = [t,theta_2_d];
d_3_d = [t,d_3_d];
theta_4_d = [t,theta_4_d];
fext_x_d = [t,fext_x_d];
fext_y_d = [t,fext_y_d];
fext_z_d = [t,fext_z_d];
fext_rx_d = [t,fext_rx_d];
fext_ry_d = [t,fext_ry_d];
fext_rz_d = [t,fext_rz_d];

figure(1)
plot3(x_d,y_d,z_d,'color','black','linewidth',2);
hold on


%% Run the Decentralized PID with Feedback Linearization
    Controller and plot the results

Kp2 = [781.66; 271.22; 1192.9; 0.605];
Kd2 = [87.21; 30.26; 133.1; 0.0675];
Ki2 = [2325.6; 806.9; 3549.03; 1.8];

Kpf = 20;
```

```
Kif = 5;
Kvd = 2000;

sim('RobotSimFORCEShybridforcepositionYESVELOCITYDAMPING')
% sim('RobotSimscapeFORCEShybridforceposition')

figure(1)

plot3(x_d,y_d,z_d,'color','black','linewidth',2);
hold on

t = [];

color = jet(1+ round(trial_time/stepsize));
x = [];
y = [];
z = [];
error_x = [];
error_y = [];
error_z = [];


for i = 0:stepsize:trial_time - stepsize
    t = [t; i];
    j = round(i/stepsize);
    x = [x; l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2))];
    y = [y; l1_*sin(theta_out(j+1,1)) + l2_*sin(theta_out(j
        +1,1) + theta_out(j+1,2))];
    z = [z; l0_ - l4_ - theta_out(j+1,3)];
    error_x = [error_x; x_d(j+1) - x(length(x))];
    error_y = [error_y; y_d(j+1) - y(length(y))];
    error_z = [error_z; z_d(j+1) - z(length(z))];
    plot3([l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2)), l1_*cos(theta_out(j+2,1)) +
         l2_*cos(theta_out(j+2,1) + theta_out(j+2,2))],[l1_*
        sin(theta_out(j+1,1)) + l2_*sin(theta_out(j+1,1) +
```

```
        theta_out(j+1,2)),l1_*sin(theta_out(j+2,1)) + l2_*sin(
        theta_out(j+2,1) + theta_out(j+2,2))],[l0_ - l4_ -
        theta_out(j+1,3),l0_ - l4_ - theta_out(j+2,3)],'color'
        ,color(j+1,:),'linewidth',3)
    %scatter3(l1_*cos(theta_out(j+1,1)) + l2_*cos(theta_out(j
        +1,1) + theta_out(j+1,2)),l1_*sin(theta_out(j+1,1)) +
        l2_*sin(theta_out(j+1,1) + theta_out(j+1,2)),l0_ - l4_
         - theta_out(j+1,3),10,color(j+1,:),'filled');
    hold on
end

error_1 = [];
error_2 = [];
error_3 = [];
error_4 = [];
error_1_simscape = [];
error_2_simscape = [];
error_3_simscape = [];
error_4_simscape = [];

for i = 1:1:length(theta_out(:,1))-1
    error_1 = [error_1; theta_out(i+1,1) - theta_1_d(i,2)];
    error_2 = [error_2; theta_out(i+1,2) - theta_2_d(i,2)];
    error_3 = [error_3; theta_out(i+1,3) - d_3_d(i,2)];
    error_4 = [error_4; theta_out(i+1,4) - theta_4_d(i,2)];
end

% for i = 1:1:length(theta_out_simscape(:,1))-1
%     error_1_simscape = [error_1_simscape;
    theta_out_simscape(i+1,1) - theta_1_d(i,2)];
%     error_2_simscape = [error_2_simscape;
    theta_out_simscape(i+1,2) - theta_2_d(i,2)];
%     error_3_simscape = [error_3_simscape;
    theta_out_simscape(i+1,3) - d_3_d(i,2)];
%     error_4_simscape = [error_4_simscape;
    theta_out_simscape(i+1,4) - theta_4_d(i,2)];
% end
```

```
scatter3(x(1),y(1),z(1),50,color(1,:),'d','filled','
    markeredgecolor','w')
scatter3(x(round(trial_time/stepsize)),y(round(trial_time/
    stepsize)),z(round(trial_time/stepsize)),50,color(round(
    trial_time/stepsize),:),'d','filled','markeredgecolor','w'
    )
set(gca,'DataAspectRatio',[1 1 1])
set(gcf,'Position',[200 50 1250 700])
hold off

t2 = [t ; trial_time];

figure(2)
subplot(4,1,1)
plot(t,theta_1_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_1 (radians)')
title('Desired and Actual Angle for Joint 1')
plot(t2, theta_out(:,1))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,2)
plot(t,theta_2_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_2 (radians)')
title('Desired and Actual Angle for Joint 2')
plot(t2, theta_out(:,2))
legend('Desired Angle','Actual Angle')
hold off

subplot(4,1,3)
plot(t,d_3_d(:,2))
hold on
```

```
xlabel('Time (seconds)')
ylabel('d_3 (metres)')
title('Desired and Actual Position for Joint 3')
plot(t2, theta_out(:,3))
plot([0,trial_time],[-rs_zbase + l0_ - l4_,-rs_zbase + l0_ -
    l4_])
legend('Desired Position','Actual Position','Reaction Surface
    ')
hold off

subplot(4,1,4)
plot(t,theta_4_d(:,2))
hold on
xlabel('Time (seconds)')
ylabel('\theta_4 (radians)')
title('Desired and Actual Angle for Joint 4')
plot(t2, theta_out(:,4))
legend('Desired Angle','Actual Angle')
hold off

set(gcf,'Position',[250 50 800 700])

combined_error_maxes = [max(error_1) , max(error_2) , max(
    error_3) , max(error_4) , max(error_1_simscape) , max(
    error_2_simscape) , max(error_3_simscape) , max(
    error_4_simscape)];
combined_error_mins = [min(error_1) , min(error_2) , min(
    error_3) , min(error_4) , min(error_1_simscape) , min(
    error_2_simscape) , min(error_3_simscape) , min(
    error_4_simscape)];
combined_error_diffs = max(combined_error_maxes) - min(
    combined_error_mins);

figure(3)

plot(t, error_1,'color',[1 0.563 0])
hold on
```

```
plot(t, error_2,'b')
plot(t, error_3,'color',[0 0.5 0])
plot(t, error_4,'color',[0.781 0 1])
% plot(t, error_1_simscape,'--','color',[1 0.563 0])
% plot(t, error_2_simscape,'b--')
% plot(t, error_3_simscape,'--','color',[0 0.5 0])
% plot(t, error_4_simscape,'--','color',[0.781 0 1])
xlabel('Time (seconds)')
ylabel('Error (rad or m)')
title('Joint Errors')
%legend('Joint 1 Error','Joint 2 Error','Joint 3 Error','
    Joint 4 Error','Joint 1 Error (SimScape)','Joint 2 Error (
    SimScape)','Joint 3 Error (SimScape)','Joint 4 Error (
    SimScape)')
legend('Joint 1 Error','Joint 2 Error','Joint 3 Error','Joint
    4 Error')
grid on
hold off

figure(4)

subplot(3,2,1)
plot(t, x_d)
hold on
plot(t, x)
hold off
ylim([min([x;x_d]) - 0.1*(max([x;x_d])-min([x;x_d])),max([x;
    x_d])+ 0.1*(max([x;x_d])-min([x;x_d]))])
xlabel('Time (seconds)')
ylabel('Position (m)')
legend('Desired','Actual')
title('End Effector x-position')

subplot(3,2,3)
plot(t, y_d)
hold on
plot(t, y)
```

```
hold off
ylim([min([y;y_d]) - 0.1*(max([y;y_d])-min([y;y_d])),max([y;
    y_d])+ 0.1*(max([y;y_d])-min([y;y_d]))])
xlabel('Time (seconds)')
ylabel('Position (m)')
legend('Desired','Actual')
title('End Effector y-position')

subplot(3,2,5)
plot(t, z_d)
hold on
plot(t, z)
plot([0,trial_time],[rs_zbase,rs_zbase])
hold off
ylim([min([z;z_d]) - 0.1*(max([z;z_d])-min([z;z_d])),max([z;
    z_d])+ 0.1*(max([z;z_d])-min([z;z_d]))])
xlabel('Time (seconds)')
ylabel('Position (m)')
legend('Desired','Actual','Reaction Surface')
title('End Effector z-position')

fext_x = fext_out_cartesian(:,1);
fext_y = fext_out_cartesian(:,2);
fext_z = fext_out_cartesian(:,3);

subplot(3,2,2)
plot(t, fext_x_d(:,2))
hold on
plot(t2, fext_x)
hold off
ylim([min([min(fext_x),min(fext_x_d)]) - 0.1*(max([max(fext_x
    ),max(fext_x_d)])-min([min(fext_x),min(fext_x_d)])),max([
    max(fext_x),max(fext_x_d)])+ 0.1*(max([max(fext_x),max(
    fext_x_d)])-min([min(fext_x),min(fext_x_d)]))])
xlabel('Time (seconds)')
ylabel('Force (N)')
legend('Desired','Actual')
```

```
title('End Effector x-Reaction Force')

subplot(3,2,4)
plot(t, fext_y_d(:,2))
hold on
plot(t2, fext_y)
hold off
ylim([min([min(fext_y),min(fext_y_d)]) - 0.1*(max([max(fext_y
    ),max(fext_y_d)])-min([min(fext_y),min(fext_y_d)])),max([
    max(fext_y),max(fext_y_d)])+ 0.1*(max([max(fext_y),max(
    fext_y_d)])-min([min(fext_y),min(fext_y_d)]))])
xlabel('Time (seconds)')
ylabel('Force (N)')
legend('Desired','Actual')
title('End Effector y-Reaction Force')

subplot(3,2,6)
plot(t, fext_z_d(:,2))
hold on
plot(t2, fext_z)
hold off
ylim([min([min(fext_z),min(fext_z_d)]) - 0.1*(max([max(fext_z
    ),max(fext_z_d)])-min([min(fext_z),min(fext_z_d)])),max([
    max(fext_z),max(fext_z_d)])+ 0.1*(max([max(fext_z),max(
    fext_z_d)])-min([min(fext_z),min(fext_z_d)]))])
xlabel('Time (seconds)')
ylabel('Force (N)')
legend('Desired','Actual')
title('End Effector z-Reaction Force')

set(gcf,'Position',[300 50 1000 700])

toc
```