Optimal Trajectory Planning and Compliant Spacecraft Capture Using a Space Robot

by
Alexander Crain

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

**Master of Applied Science
in
Aerospace Engineering**

Carleton University
Ottawa, Ontario

© 2018
Alexander Crain

This work is dedicated to my entire family; you've all been there to support me in your own ways, and I love you all.

Abstract

Due to the ever-increasing volume of on-orbit debris, determining viable means of capturing and removing said debris has become vital. This thesis proposes solutions to the first two of three phases during debris removal: namely, deployment of a robotic manipulator, and the capture of a target spacecraft.

For the deployment phase, to solve the nonlinear trajectory planning problem for free-floating robots, this thesis proposes the use of pseudospectral methods. Using three different pseudospectral optimal control tools, simulations are performed and it is shown that each tool is capable of finding a unique local minima, which ensures zero attitude change by the end of the deployment. Each solution is then validated using Pontryagin's minimum principle, Bellman's principle of optimality, and by propagating the control torques using a numerical integrator and the dynamics model. Experimental validation is performed at Carleton University's Spacecraft Robotics and Control Laboratory to further investigate the solutions obtained from each tool. Ultimately, it is determined that all solutions resulted in a reduced attitude disturbance at the end of the deployment manoeuvre.

After the deployment phase, a nonlinear disturbance observer-based impedance controller is proposed for the compliant capture of free-flying spacecraft using a free-flying robot. An existing nonlinear disturbance observer that can be used to determine the end-effector disturbance for a fixed-base manipulator without knowledge of the second state derivatives is extended for use with a free-flying robot. The observer is shown to be asymptotically stable. The observer is then paired with an impedance controller in a simulated contact scenario, and the response is shown to be stable and compliant even in the presence of noise. Furthermore, the estimated contact force from the observer is compared to the actual disturbance, and the error converges asymptotically to zero. Finally, the proposed technique is successfully experimentally validated at Carleton University's Spacecraft Robotics and Control Laboratory.

Acknowledgements

First and foremost, I want to express my gratitude to Steve Ulrich, my supervisor. Your patience and guidance helped make this thesis possible, hopefully you also learned something during our time working together.

I also want to thank my friends and colleagues; in particular, thanks to Kirk Hovell (who, for all intents and purposes, built the lab from the ground up) and Justin Kernot (whose excellence in design work was fundamental to the success of the experiments in this thesis).

Thanks as well to my family, to whom this thesis is dedicated to - family is the most important part of my life, and I'm glad that I am able to spend so much time with everyone. Thanks especially to my grandparents, I would not be who I am today without their influence, and their impromptu vacation invitation may have saved my sanity and ensured the completion of this thesis.

This research was financially supported in part by the Natural Sciences and Engineering Research Council of Canada under the Canada Graduate Scholarship CSGM-302098-2017. In addition, the Ontario Graduate Scholarship, entrance scholarships to the Mechanical and Aerospace Engineering department, and Teaching Assistantships helped fund my research.

Table of Contents

Abstract	iii
Acknowledgements	iv
List of Tables	ix
List of Figures	x
List of Symbols	xiii
List of Acronyms and Initialisms	xviii
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	1
1.3 Previous Work	2
1.3.1 Optimal Trajectory Planning	3
1.3.2 Compliant Spacecraft Capture	6
1.4 Thesis Objectives	8
1.5 Contributions	9
1.6 Organization	10
Chapter 2 Development of Experimental Facility	12
2.1 Introduction	12
2.2 Spacecraft Properties	14
2.3 Robotic Manipulator Properties	18
2.4 Development of SPOT 2.0 Software	26
Chapter 3 Kinematics and Dynamics of Space Robots	28
3.1 Primary Assumptions	28

3.2	Kinematics Model	29
3.3	Dynamics Model	29
3.4	Generalized Jacobian	31
3.5	Kinematics and Dynamics of the SPOT Space Robot	32
3.6	A Note on Model Validation	39
Chapter 4	Optimal Trajectory Planning for Space Robots	40
4.1	Approach Overview	40
4.1.1	Gauss Pseudospectral Collocation	41
4.1.2	Sparse Nonlinear Optimizer	44
4.2	Simulations	45
4.2.1	Optimal Guidance - Problem Setup	45
4.2.2	Results	48
4.3	Verification of Optimal Solutions	52
4.3.1	Pontryagin's Necessary Conditions	53
4.3.2	Bellman's Principle of Optimality	58
4.3.3	Propagation of Optimal Solutions	61
4.4	Experiments	63
4.4.1	Results	65
4.4.2	Conclusions and Context	67
Chapter 5	Compliant Spacecraft Capture	70
5.1	Primary Assumptions	70
5.2	Baseline Disturbance Observer	71
5.3	Nonlinear Disturbance Observer	72
5.3.1	Basic Observer	72
5.3.2	Modified Observer	73
5.3.3	Observer Stability	74
5.4	Impedance Controller	77
5.5	Simulations	78
5.5.1	Contact Model	78

5.5.2	Simulation Setup	79
5.5.3	Results - Baseline Observer	80
5.5.4	Results - Nonlinear Observer	83
5.5.5	Verifying Observer Stability	85
5.6	Experiments	86
5.6.1	Case #1 - Contact with a Static Wall	87
5.6.2	Case #2 - Capture of a Controlled Target	90
5.6.3	Conclusions and Context	94
Chapter 6	Conclusion	96
6.1	Thesis Motivation	96
6.2	Significance of Work	97
6.2.1	Published Works	97
6.2.2	Under Review	97
6.3	A Note on Practicality	98
6.4	Recommendations for Future Work	99
6.5	Primary Contributions	100
Bibliography		102
Appendix A	Supplementary Documentation - Chapter #2	109
A.1	Getting Started	109
A.2	Quick Start	111
A.2.1	Creating a New Experiment	111
A.2.2	Initialization	112
A.2.3	Preparing the Diagram	114
A.2.4	Performing the Experiment	116
A.3	Device Driver Blocks	118
A.3.1	Dynamixel Driver Blocks	119
A.3.2	PhaseSpace Driver Blocks	163

Appendix B	Supplementary Documentation - Chapter #3	196
B.1	Derivation of Dynamics Model - MATLAB Script	196
B.2	Elements of the Space Robot Inertia Matrix	200
B.3	Elements of the Space Robot Centrifugal/Coriolis Matrix	203
Appendix C	Supplementary Documentation - Chapter #4	208
C.1	MATLAB Code - DIDO Optimization	208
C.1.1	Initialization Script	208
C.1.2	Dynamics Function	214
C.1.3	Event Function	216
C.1.4	Cost Function	217
C.2	MATLAB Code - GPOPS-I Optimization	218
C.2.1	Initialization Script	218
C.2.2	Dynamics Function	225
C.2.3	Cost Function	227
C.3	MATLAB Code - TOMLAB Optimization	227
C.4	MATLAB Code - Baseline Controller	233
C.4.1	Initialization Script	233
C.4.2	Top-Level Simulink Diagram	237
C.5	Pontryagin's Principle - Symbolic Derivation	238
Appendix D	Supplementary Documentation - Chapter #5	247
D.1	Baseline - Initializer Script	247
D.2	Baseline - Disturbance Observer Function	249
D.3	Baseline - Top-Level Simulink Diagram	251
D.4	NDO - Initializer Script	252
D.5	NDO - Disturbance Observer Function	254
D.6	NDO - Top-Level Simulink Diagram	256
D.7	Generalized Jacobian - Force Estimation	257

List of Tables

2.1	Mass properties and COM for each SPOT platform.	15
2.2	Inertia properties for each SPOT platform.	17
2.3	Components required to build the robotic manipulator.	18
2.4	Dynamixel MX-64 hardware specifications.	20
2.5	Mass properties for each link.	23
2.6	Inertia properties for each link.	26
4.1	Convergence time and final attitude for each solution.	52

List of Figures

2.1	SPOT platforms and gravity offset table.	13
2.2	RED platform with labels for each LED.	14
2.3	Configuration for COM measurement.	14
2.4	Free-body diagram for COM measurement.	15
2.5	COM for RED (left) and BLACK (right).	16
2.6	RED platform - mass moment of inertia estimation.	17
2.7	Assembled (top) and exploded view (bottom) of a link.	18
2.8	Exploded view of the magnetic gripper/dock.	19
2.9	Photo of the fully assembled robotic manipulator.	19
2.10	Dynamixel MX-64 PID control logic (https://goo.gl/1wAM6R).	21
2.11	Calibration curve to convert current into torque.	22
2.12	Configuration for COM measurement (link).	22
2.13	Free-body diagram of a link for COM measurement.	23
2.14	Experimental setup for inertia measurement using BP technique.	24
2.15	RED platform - mass moment of inertia with link #1.	25
3.1	Representation of the reference frame and generalized states.	33
4.1	Joint trajectories for joint #1.	48
4.2	Joint trajectories for joint #2.	48
4.3	Joint trajectories for joint #3.	49
4.4	Attitude during the optimized manipulator motion.	49
4.5	EE trajectory - transpose Jacobian.	50
4.6	EE trajectory - GPOPS-I.	50
4.7	EE trajectory - DIDO.	51
4.8	EE trajectory - TOMLAB.	51
4.9	Hamiltonian evolution equation at each time-step.	58
4.10	Applying Bellman's principle to the solution from GPOPS-I.	59
4.11	Applying Bellman's principle to the solution from DIDO.	59

4.12	Applying Bellman's principle to the solution from TOMLAB. .	60
4.13	Total sum of squares for each solution at various starting nodes.	61
4.14	Propagation of the solution from GPOPS-I.	62
4.15	Propagation of the solution from DIDO.	62
4.16	Propagation of the solution from TOMLAB.	63
4.17	Control logic for experiment.	64
4.18	Experimental attitude disturbance - transpose Jacobian. . . .	65
4.19	Experimental attitude disturbance - GPOPS-I.	65
4.20	Experimental attitude disturbance - DIDO.	66
4.21	Experimental attitude disturbance - TOMLAB.	66
4.22	Final platform attitudes following deployment manoeuvre. . .	68
5.1	Reference frame and capture point on BLACK platform. . . .	79
5.2	Control logic for DO-based impedance controller.	81
5.3	Change in reference point due to impedance controller - baseline.	82
5.4	Estimated vs. actual EE force - baseline.	82
5.5	Error between estimated and actual EE force - baseline.	83
5.6	Change in reference point due to impedance controller - NDO.	84
5.7	Estimated vs. actual EE force - NDO.	84
5.8	Error between estimated and actual EE force - NDO.	85
5.9	Stability criterion evaluated for the duration of the manoeuvre.	86
5.10	Experiment setup for case #1.	87
5.11	Experiment setup for case #1 - EE position.	87
5.12	Contact force in the x-direction - baseline.	88
5.13	Contact force in the x-direction - NDO.	89
5.14	Change in reference point due to impedance controller - baseline.	89
5.15	Change in reference point due to impedance controller - NDO.	90
5.16	Experiment setup for case #2.	91
5.17	Contact force in the x-direction - baseline.	92
5.18	Contact force in the x-direction - NDO.	92

5.19	Change in reference point due to impedance controller - baseline.	93
5.20	Change in reference point due to impedance controller - NDO.	93
5.21	Platform behaviour without impedance control - case #1. . . .	95
5.22	Platform behaviour without impedance control - case #2. . . .	95
A.1	Graphical user interface used in the SPOT 2.0 software.	116
C.1	Simulink diagram for transpose Jacobian simulations.	237
D.1	Simulink diagram for baseline DO simulations.	251
D.2	Simulink diagram for NDO simulations.	256

List of Symbols

In this thesis, any **bolded** variable is a matrix or a vector, while any *italicized* variable is a scalar.

a_i	Distance between joint i and COM of link i , [m]
A	Sparse matrix
A, B, C	Force readings at each corner of the rigid body, [N]
b_0	Distance between COM and joint #1, [m]
b ₀	Position vector from inertial frame to joint #1, [m]
b_i	Distance between COM of link i and joint $i+1$, [m]
c	Constant positive gain value for disturbance observer
c_c	Contact surface damping, [Ns/m]
C_{ij}	Components of the Centrifugal/Coriolis matrix
c	Vector of m -smooth nonlinear constraint functions $c_i(\mathbf{x}_v)$
C	Centrifugal/Coriolis matrix
C_b	Current, [bits]
c_{ijk}	Christoffel symbols
c _b	Velocity dependent nonlinear matrix for the base
c _m	Velocity dependent nonlinear matrix for the manipulator
C _t	Impedance controller damping [Ns/m]
d	Vector of actual disturbance torques, [Nm]
$\hat{d}$	Vector of estimated disturbance torques, [Nm]
D	Distance between force measurement B and C , [m]
D_{ik}	Differentiation matrix
D, $\bar{D}$	Differential operators
d	Depth of the platform, [m]
d_p	Distance between target contact point and end-effector, [m]
d_1	Distance between COM and scale #1, [m]
d_2	Distance between COM and scale #2, [m]

d_c	Distance between cables during inertia measurement, [m]
d_T	Distance between both scales, [m]
Δd_1	Distance between base COM and base/link COM, [m]
Δd_2	Distance between link COM and base/link COM, [m]
E	Distance between force measurement A and $B - C$, [m]
E_c	Endpoint constraint
$\bar{E}$	Endpoint Lagrangian
$\mathbf{e}$	Path constraints
$\mathbf{e}_{ob}$	Disturbance observer error, [Nm]
F	Running cost
$\mathbf{f}_b$	Force and moment vector on the spacecraft COM, [N, Nm]
$\mathbf{f}_e$	Force and moment vector on the end-effector, [N, Nm]
$\hat{\mathbf{f}}_e$	Estimated force and moment vector on the end-effector, [N, Nm]
F_x, F_y	Forces acting on the base of the spacecraft, [N]
g	Gravity constant for Earth, [m/s ²]
g_L	Lagrange component of the cost function
H	Hamiltonian
$\mathcal{H}$	Lower Hamiltonian
$\bar{H}$	Lagrangian of the Hamiltonian
I	Mass moment of inertia, [kgm ²]
I_{link}	Mass moment of inertia of linkage, [kgm ²]
$I_{base/link}$	Mass moment of inertia of base/link combination, [kgm ²]
I_0, I_1, I_2, I_3	Mass moment of inertia for each body, [kgm ²]
$\mathbf{I}$	Identity matrix
J	Cost function
$\mathbf{J}_b$	Jacobian matrix for the base
$\mathbf{J}_m$	Jacobian matrix for the manipulator
$\mathbf{J}^*$	Generalized Jacobian matrix for the manipulator
$\mathbf{J}_v$	Jacobian relating the linear velocity to the joint velocities
$\mathbf{J}_\omega$	Jacobian relating the angular velocity to the joint velocities

k_c	Contact surface stiffness, [N/m ^{3/2}]
$\mathbf{K}_t$	Impedance controller stiffness
$\mathbf{K}_{FF1}, \mathbf{K}_{FF2}$	Feedforward gains
$\mathbf{K}_P, \mathbf{K}_I, \mathbf{K}_D$	Proportional, integral, and derivative gains
$\mathbf{K}_a$	Anti-windup gain
$\mathbf{L}$	Observer gain matrix
L_b	Constant lower bound for the NLP
L_c	Length of cable during inertia measurement, [m]
L_k	Lagrange interpolating polynomial
L	Lagrangian
$\mathbf{L}_i$	Position vector from joint i to joint $i+1$, [m]
m	Mass, [kg]
m_{base}	Mass of the base, [kg]
m_{link}	Mass of a linkage, [kg]
m_0, m_1, m_2, m_3	Mass of each body, [kg]
M_{ij}	Components of the space robot inertia matrix
$\mathbf{M}$	Space robot inertia matrix
$\bar{\mathbf{M}}$	Modified space robot inertia matrix
$\mathbf{M}_b$	Inertia matrix for the base of the space robot
$\mathbf{M}_{bm}$	Coupled inertia matrix of the space robot
$\mathbf{M}_m$	Inertia matrix for the links of a space robot
$\mathbf{M}_t$	Impedance controller characteristic mass
$\mathbf{n}$	Normal vector
$\mathbf{p}$	Designed function vector
$\mathbf{p}_e$	Position vector from inertial frame to end-effector, [m]
$\mathbf{p}_c$	Position vector from inertial frame to contact point, [m]
$\mathbf{p}_0$	Position vector from inertial frame to spacecraft COM, [m]
P_{l_N}	N^{th} -degree Legendre polynomial defining the Gauss points
$\mathbf{q}$	Generalised state vector
q_0, q_1, q_2, q_3	Spacecraft attitude and joint angles, [rad]

$\mathbf{r}_e$	Compensating position vector, [m]
$\mathbf{r}_d$	Desired trajectory without compensation, [m]
$\mathbf{r}_a$	Desired trajectory with compensation, [m]
$\mathbf{R}$	Upper triangular matrix
t	Time, [s]
t_0	Initial time, [s]
t_f	Final time, [s]
T	Kinetic energy component of the Lagrange equation
$\mathbf{T}_v$	Translational kinetic energy
$\mathbf{T}_\omega$	Rotational kinetic energy
U	Potential energy component of the Lagrange equation
$\mathbf{U}$	Controls discretized at the Gauss points
$\mathbf{v}$	Endpoint covector
$\mathbf{v}_c$	Linear velocity of the COM for each body, [m/s]
V	Candidate Lyapunov function
W	Sum of A , B , and C force readings, [N]
w	Width of the platform, [m]
w_1	Weight measured at scale #1 during COM measurement, [kg]
w_2	Weight measured at scale #2 during COM measurement, [kg]
w_T	Total weight of linkage during COM measurement, [kg]
w_k	Gauss weight at each Gauss point
x	Center of mass in the x -direction, [m]
$\mathbf{x}_b$	Linear and angular position for the base spacecraft, [m, rad]
$\mathbf{x}$	State-space vector, [m, rad]
$\mathbf{x}_0$	Initial state-space vector, [m, rad]
$\mathbf{x}_f$	Final state-space vector, [m, rad]
$\mathbf{X}$	States discretized at the Gauss points, [m, rad]
$\mathbf{x}_e$	Error between reference and actual end-effector pose, [m, rad]
$\mathbf{x}_v$	Vector of n -variables
x^I	X-axis of inertial space reference frame

x^b	X-axis of body reference frame
x_{ee}	X-position of the end-effector in inertial space, [m]
x_{el}	X-position of the elbow in inertial space, [m]
x_w	X-position of the wrist in inertial space, [m]
y	Center of mass in the y -direction, [m]
y_{ee}	Y-position of the end-effector in inertial space, [m]
y_{el}	Y-position of the elbow in inertial space, [m]
y_w	Y-position of the wrist in inertial space, [m]
y^I	y-axis of inertial space reference frame
y^b	y-axis of body reference frame
$\mathbf{z}$	Auxiliary variable vector
α	Angular acceleration, [rad/s ²]
β	Vector of boundary constraints on the NLP
η	Nondimensionalized time
θ	Vector of manipulator joint angles, [rad]
λ	Vector of costates
Λ_l	Lower triangular matrix
Λ_u	Upper triangular matrix
μ	State and control constraint covector
ϕ	Angle between the x -component of the body frame and $\mathbf{b}_0$, [rad]
Φ	Mayer component of the cost function
τ	Torque, [Nm]
τ_i	Torque at joint i , [Nm]
$\tau_0, \tau_1, \tau_2, \tau_3$	Torque on body and joints, [Nm]
$\boldsymbol{\tau}$	Vector of joint torques, [Nm]
$\boldsymbol{\tau}_c$	Vector of joint control torques, [Nm]
ω	Frequency of oscillations during inertia measurement, [rad/s]
ω_c	Angular velocity of the COM for each body, [rad/s]

List of Acronyms and Initialisms

BP	Bifilar Pendulum
COM	Center of Mass
CSA	Canadian Space Agency
DO	Disturbance Observer
DOF	Degree-of-Freedom
EE	End-Effector
GN&C	Guidance, Navigation, and Control
GUI	Graphical User Interface
HMC	Hamiltonian Minimization Condition
LED	Light Emitting Diode
MATLAB	Matrix Laboratory
NDO	Nonlinear Disturbance Observer
NLP	Nonlinear Problem
ODE45	4(5)th Order Runge-Kutta-Fehlburg
PS	Pseudospectral
PID	Proportional-Integral-Derivative
RP3	Raspberry Pi 3
SPOT	Spacecraft Proximity Operations Testbed
SNOPT	Sparse Nonlinear Optimizer
SRCL	Spacecraft Robotics and Control Laboratory
SQP	Sequential Quadratic Programming
QP	Quadratic Programming

Chapter 1

Introduction

Here, the motivations for this thesis will be presented, followed by an elaboration on the problems being solved. Following this, a comprehensive literature review will be provided for the field of optimal trajectory planning as well as the field of compliant spacecraft capture. This chapter will then be concluded with the objectives for this thesis, the contributions of the author, and a summary of each chapter’s content.

1.1 Motivation

There are currently over 500,000 pieces of debris being tracked as they orbit around the Earth. Sources of large debris include malfunctioning or de-commissioned spacecraft and depleted rocket engines, some of which are travelling at speeds of up to 28,000 km/h. It is now widely known that, even with no future launches, orbital debris has reached the point where any collisions among large-body debris will lead to an unstable growth in debris [1]. This was predicted over 30 years ago, when the term “Kessler Syndrome” was coined. Kessler Syndrome, in brief, refers to the concept of collisional cascading of objects. Two orbiting objects that pass through the same distance from the object that they are orbiting about will eventually collide [2, 3] and break up into a number of smaller fragments, thus creating an even larger number of objects. However, research has also shown that removing as few as five large objects each year can stabilize debris growth [2].

1.2 Problem Statement

One of the main technological challenges inherent to space debris removal missions is related to the autonomous robotic capture manoeuvres. Specifically, the disturbance torque applied to the servicer robotic spacecraft due to the physical motion of a

manipulator may cause the destabilization of the servicer spacecraft or severe damage to the robotic arm. The most common approach to solving this issue is the concept of trajectory optimization, which involves determining the best set of control inputs to minimize a given cost function. There exists a plethora of software tools that are designed to hide the bulk of the background calculations from the user, thus in concept creating turnkey software that requires minimal effort from the user. However, the problem of optimal control is more than the implementation of the problem, and each software tool has different nuances that can be confusing to first time users.

Once the robotic manipulator has been deployed using trajectory optimization techniques, the next task is to capture the target spacecraft. However, laboratory experiments in this area are still ongoing due to the high risk of complications during capture. The primary culprit behind said complications is the impact forces between the space robot and the target. If proper caution is not exercised, the impact of the end-effector (EE) on the target can cause both to become unstable, thereby rendering any servicing or capture highly improbable. In order to achieve safe capture, the contact forces between the EE and the target must be known. Once these forces are known, control laws can be developed to ensure the safe and compliant capture of the target. The simplest solution is to equip the manipulator with a force-moment sensor. However, the added weight and complexity associated with the installation of a force-moment sensor, coupled with their high cost, may render this option undesirable.

This thesis is therefore concerned with the stable deployment of robotic manipulators and the subsequent sensorless compliant capture of a target spacecraft.

1.3 Previous Work

The past work that has been published in both fields is extensive, and a summary of any important publications will be presented in this section. This literature review will be divided according to the two primary topics addressed in this thesis: optimal trajectory planning and compliant spacecraft capture.

1.3.1 Optimal Trajectory Planning

The list of possible manoeuvres that could benefit from trajectory optimization is extensive, yet there have been many researchers who have proposed solutions to the problem of optimal trajectory planning in the context of robotics. Dubowsky and Torres [4,5] worked on preliminary path planning for space manipulators by presenting a technique called the “enhanced disturbance map.” This map was used to aid in the understanding of the complex problem of trajectory optimization, as well as in the development of algorithms to reduce disturbances. Agrawal and Xu [6] proposed a global optimum path planning technique for redundant space manipulators. This technique used a variational approach to minimize the objective function; that is to say, the holonomic and nonholonomic constraint equations for the free-floating robot were adjoined to the objective function using the Lagrange multiplier technique, which leads to a system of differential and algebraic equations that are then numerically integrated. Papadopoulos and Abu-Abed [7] designed a three degree-of-freedom (DOF) redundant manipulator based on force and moment transmission analysis, with the objective of achieving reactionless motions. This was achieved by eliminating the dynamic reaction forces through force balancing, while reaction moments were eliminated by following reactionless paths. However, to simplify the problem, the authors chose to render the dynamics of the system decoupled and invariant.

Lapariello et al. [8] presented a time-optimal motion planning method for free-flying robots while considering full-state actuation such that large spacecraft displacements were allowed. The proposed solution from the authors did not require any spacecraft actuation to achieve the end goal, as the robot motion alone proved to be sufficient for the proposed task. Aghili [9] focused on the guidance of a robot manipulator to capture and detumble a spacecraft with unknown dynamic parameters. A Kalman filter was used to estimate the states and all dynamic parameters, and then the optimal control actions were determined such that the time of travel and the weighted norms of the EE velocity and acceleration were minimized. Experimental validation of the proposed technique was also performed, and successful grasping was achieved despite the unknown parameters of the target spacecraft. Yoshida et

al. [10] discussed the minimization of the base attitude deviation before and after the contact with the target from the viewpoint of angular momentum distribution. Their proposed methodology for the capture and detumbling of a spacecraft used bias momentum approach during the approaching phase, impedance control during the impact, and distributed momentum control during the post-impact phase.

Huang and Xu [11] proposed a particle swarm optimization strategy to determine globally time-optimal trajectories for free-floating robots, and the control strategy demonstrated satisfactory performance. Oki et al. [12] investigated an optimal control method for capturing a tumbling spacecraft, although they focused primarily on minimizing the operational time for fast capturing while still paying attention to limitations on the grasping force and torque, detumbling operations along an arbitrary arm motion, and parameter uncertainty in the target spacecraft. This concept of detumbling speed proved to be beneficial when handling targets with uncertain dynamics. Cascio [13], meanwhile, looked at the concept of optimal path planning for multi-arm, multi-link fixed-base robotic manipulators using pseudospectral (PS) methods. The optimal control problem was posed for fixed-based manipulators only, yet the resulting optimization was still highly nonlinear and was solved using the software package DIDO. The optimal solution obtained from the software package was verified using Pontryagin's minimum principle.

Liu et al. [14] proposed an optimal trajectory planning method for the stabilization of coupled space robots following a capture using particle swarm optimization. In their work, the trajectory planning problem was transformed into a nonlinear optimization problem, where the attitude deflection of the base was used as the cost function. The authors found that the particle swarm optimization technique was effective at solving the problem. Xu et al. [15] investigated an adaptive version of reaction null-space based control for free-floating robots with uncertain kinematics and dynamics. The contribution of their work lies in the derivation of a linear expression for use as the basis of the adaptive control; the authors ultimately found that their proposed controller achieved both the base attitude regulation and the continuous path tracking of the EE.

Angel et al. [16] showed that disturbances to the attitude of the free-floating robot

could be greatly reduced through offline nonlinear optimal guidance laws which made use of PS techniques to predict the optimal manipulator trajectory. Similar to the work published by Cascio [13], Angel et al. [16] solved the nonlinear optimization problem using the software package TOMLAB. These results were independently validated by Crain and Ulrich [17], where the nonlinear optimization problem was solved using the software package GPOPS-I, and it was determined that both tools converged to approximately the same local minimum.

Rybus et al. [18] presented an improved manipulator trajectory optimization method for systems with non-conservative momentum. Their proposed technique allowed for the minimization of a quadratic norm connected with the power use of motors in manipulator joints. This differs from existing techniques due to the fact that it includes the possibility to constrain the final velocity of the EE by making the period of the capture manoeuvre an optimization variable. Virgili-Llop et al. [19, 20] described a set of laboratory-based experiments using a free-floating robot, where they demonstrated the autonomous capture of a non-moving space object using a robotic manipulator. The authors made use of the POSEIDYN testing platform [21], which can create a quasi-frictionless planar environment. Using this testing platform, the authors were able to implement and test a variety of control methods in a dynamically accurate environment.

Other publications in the field includes the work of Misra and Bai [22], which consists of task-constrained trajectory planning for a free-floating robot system using convex optimization. In brief, based on the conservation of the system moment, Misra and Bai [22] formulated a trajectory planner as a convex quadratic programming problem in the joint space. By formulating the problem in such a way, they were able to rapidly generate optimal trajectories for the EE, which in turn minimized the disturbance to the base of the space robot. Additionally, Misra and Bai [23] have published work in the field of optimal path planning for free-flying space robots where the attitude of the base spacecraft is actively controlled. The main contribution was in the formulation of the final EE pose constraint and its subsequent convex relaxation; specifically, the proposed formulation allowed the problem to be solved efficiently through convex optimization.

1.3.2 Compliant Spacecraft Capture

The problem of compliant spacecraft capture can be sub-divided into two smaller categories: the estimation (or measurement) of EE forces, and the development of a control law that uses said forces to ensure compliant contact.

In the field of force estimation, Yoshikawa and Yamada [24] studied the relationship between the initial relative velocities of a robot's EE and the resulting contact impulse. From this relationship, they were able to estimate the contact forces between the target and the EE. Hacksel and Salcudean [25] demonstrated that a good estimate of environment forces can be made by approximating the observer dynamics as a damped spring-mass system driven by said environment forces. Then, by choosing sufficiently-large gains to simulate a stiff spring, it became possible to estimate the forces by measuring the displacement of the spring from equilibrium. Kim et al. [26] proposed a robust learning control algorithm based on a disturbance observer (DO). Of particular importance, their algorithm allowed for accurate force estimation in the presence of parameter variations, mechanical nonlinearities, unmodelled dynamics, and external disturbances.

Meanwhile, Moosavian et al. [27] developed the multiple impedance control method, which can be used to manipulate space objects. The multiple impedance control law is based on the concept of impedance control, and predicts the contact forces from the equations of motion for a free-flying robot. Eom et al. [28] proposed a method for force control without the need for a force sensor. Their proposed technique used a DO at each joint, a simple equivalent robot dynamics model, and a DO output estimator. This output estimator adjusted uncertain parameters for the robot manipulator via the gradient method, which minimized the quadratic form of the error between the output of the DO and the output estimator. Using this estimated force, a force controller was then designed. Chen et al. [29] developed a nonlinear disturbance observer (NDO) with global asymptotic stability. Their proposed observer did not require any knowledge of the second derivative of the states and has been shown to work well for highly-nonlinear systems, such as fixed-base robotic manipulators.

Van Damme et al. [30] presented two different ways of estimating the interaction force at the EE of a robot. However, the second technique was mainly used as a

baseline for their main contribution: utilizing a combination of the filtered dynamics equation with a recursive least squares estimation algorithm to provide a smooth force signal. This technique was shown to be effective at estimating the EE force. Wahrburg et al. [31] developed a process to estimate the contact force and torques for robotic manipulators in task space using Kalman filters and generalized momentum. This approach introduced additional DOF in the design of the estimator to increase the robustness with respect to disturbances such as uncertainty in the joint friction.

Once an estimate of the contact force is available, a common approach for controlling the behavior of a manipulator when in contact with an object is the use of an impedance controller. This, too, has been addressed in past work. Prevot et al. [32] performed research on the control of multi-manipulator systems in the context of the Canadian DEXTER robot arm. Their proposed scheme allowed for the control of an object’s position, the internal force on the object, and the external force of the object on the environment. Experiments were successfully performed using the Canadian Space Agency (CSA) Automation and Robotics Testbed, and good precision was achieved. Moosavian et al. [33] designed an impedance control law to manipulate space objects using multiple arms; however, they assumed that the contact force was measured. Simulation results for their proposed technique demonstrated smooth performance and negligible tracking errors in the presence of impacts. Abiko et al. [34] introduced an impedance controller and derived a novel and computationally-efficient form of the dynamics for a free-flying robot. Using these dynamics, they considered the grasped target as a disturbance force on the EE, and thus the proposed control method is completely independent of the target’s mass properties.

Uyama et al. [35] outlined an impedance-based control law for a free-flying robot utilizing a compliant wrist for the capture of non-cooperative spacecraft. The control law uses a contact dynamics model in conjunction with the open loop impedance controller to achieve a desired coefficient of restitution defined between the EE and a contact point on the target. Experiments were performed by the authors, and the technique proved to be effective at realizing the desired coefficient of restitution with fairly small errors. Similar to the contact force estimation, research in the

field of impedance control extends beyond the field of spacecraft capture. One such example is the research performed by Roveda et al. [36], who presented a force-tracking impedance controller which granted an overshoot-free contact force to allow interactions with partially-unknown environments using a modified linear quadratic regulator to analytically determine the optimal gains. A probing task was carried out and results have shown that the controller avoids force overshoots and instabilities.

Combining force estimation techniques with impedance controllers is also not a new idea. Murakami et al. [37] described a force sensorless controller for multi-DOF manipulators, which they combined with active compliance control to ensure safe contact with an object. Their approach entailed calculating the disturbance torque using a model of the manipulator. However, their work assumed that the manipulator parameters are unknown, and the authors made use of the DO to determine the unknown manipulator properties. Bickel and Tomizuka [38] investigated the usefulness of implementing a DO alongside an existing impedance strategy referred to as the Anderson and Spong Hybrid Impedance Control. This controller separates the task space of a manipulator into two subspaces: an explicit force controlled subspace and an impedance controlled position subspace. More recently, Angel et al. [39] developed a simple model-based DO which they paired with an impedance controller to ensure the safe and compliant capture of a target spacecraft, and were able to demonstrate - in numerical simulations - safe and stable contact with a rigid surface.

1.4 Thesis Objectives

This work examines the optimal deployment of robotic manipulators using the PS methods implemented in three popular commercial optimization tools, followed by the compliant capture of spacecraft using a nonlinear disturbance observer-based impedance control.

In the first part of this work, three commonly-used PS optimization tools - DIDO, GPOPS-I, and TOMLAB - are used to determine zero-attitude change deployment manoeuvres for a free-floating robot. This type of manoeuvre is concerned with determining the EE trajectory which ensures that the attitude of the spacecraft returns to its initial state. By performing this type of manoeuvre, it is possible

to save on fuel as well as reduce a reaction wheel’s saturation level. The nonlinear problem (NLP) to be addressed takes into consideration the constraints on the system while minimizing the difference between the initial and final attitude. Contrary to past work in this area, the minimization of any running costs is ignored in favour of simply minimizing the change in attitude over the course of the deployment, thereby reducing the overall computational effort required to converge to a solution. The reduced computational effort allows for faster manoeuvre development, which minimizes the length of time that the control designer must wait between defining the NLP and obtaining a solution. The results from each tool are verified using three important steps: first, the solutions are numerically checked using Pontryagin’s minimum principle. Then, Bellman’s principle of optimality is used to verify that an optimal control policy has been found. Finally, the optimal solutions from each tool are propagated forward using a numerical integrator to determine whether the solutions followed the dynamic constraints. Following the verification of the results obtained, the optimal solutions are experimentally validated at Carleton University using the Spacecraft Proximity Operations Testbed (SPOT).

In the second part of this work, an NDO-based impedance controller is used to capture a target spacecraft without causing the destabilization of either the servicer or the target. First, the derivation of the observer is presented and the observer is shown to be asymptotically stable using Lyapunov’s stability criterion. Then, the force estimated from the observer is used as the input into an impedance controller, thereby ensuring the compliant capture of a spacecraft. Simulations are performed to show that the technique achieves compliant contact. Following the simulations, experiments are performed using SPOT to validate the results.

1.5 Contributions

The contributions of this work to the fields of optimal trajectory planning and compliant spacecraft capture are:

- The formulation of the nonlinear guidance problem for zero-attitude disturbance space robot deployment manoeuvres;

- The transcription of the nonlinear guidance problem into three pseudospectral optimization tools and subsequent numerical validation using Bellman’s principle of optimality, Pontryagin’s minimum principle, and by propagating the model using the optimal control inputs;
- The integration of a robotic manipulator at Carleton University’s Spacecraft Robotics and Control Laboratory (SRCL), including both software and hardware;
- The selection of a design vector for the nonlinear disturbance observer and subsequent stability proof for free-flying space robots;
- The experimental validation of the optimal guidance solutions from each pseudospectral tool;
- The experimental validation of the asymptotically-stable nonlinear disturbance observer for free-flying robots; and
- The development of the SPOT 2.0 experiment software, which integrates the capabilities of the previous version with the simplicity of a graphical user interface (GUI) while also integrating all C/C++ code into the matrix laboratory (MATLAB)-Simulink environment.

1.6 Organization

This thesis is organized as follows:

Since all subsequent chapters involve experimental validation in some form, Chapter 2: *Development of Experimental Facility* showcases the capabilities of the laboratory. This includes the contributions from the author to the laboratory environment as well as the specifications for any relevant hardware.

To ensure that simulations are as close to reality as possible, Chapter 3: *Kinematics and Dynamics of Space Robots* presents the equations of motion for a general space robot. This includes the forward kinematics, the dynamics derived using the Euler-Lagrange method, and the generalized Jacobian formulation. This chapter

concludes by deriving the equations of motion for the planar space robot used in SPOT.

To deploy the manipulator while minimizing the nominal attitude deviation, Chapter 4: *Optimal Trajectory Planning for Space Robots* provides an overview of the theory specifically used in Gauss PS methods. The sparse nonlinear optimizer (SNOPT) used is briefly detailed, and the equations required to check the necessary conditions of optimality are summarized. Simulations are then performed using GPOPS-I, TOMLAB, and DIDO. The results from each tool are validated using the aforementioned necessary conditions of optimality. This chapter is concluded by presenting the results of the experimental validation.

Having established the process by which the manipulator will be deployed, Chapter 5: *Compliant Spacecraft Capture* outlines the derivation of the NDO, including an analysis of its stability. An impedance controller is also shown in this chapter, and the observer is paired with the impedance controller to ensure compliant capture in the absence of any force sensors. Similar to the previous chapter, this chapter includes simulations to demonstrate the success of the proposed capture technique, in addition to experiments which validate the technique.

Chapter 6: *Conclusions* ends this thesis with a concluding statement, a summary of the contributions in this work, a list of publications that have resulted from this research, a comment on the practicality of the proposed techniques, and suggestions on future work.

Chapter 2

Development of Experimental Facility

In order to assess the performance of the techniques proposed in this thesis, experimental validation was required. Carleton University’s SPOT is a unique facility in that it is the largest spacecraft and robotics testing facility in Canada. When the author began his MASc program, the facility was mostly complete, though it lacked certain functionalities. The author made several contributions to the development of the facility, which will be detailed in this chapter. These improvements include the design and implementation of the robotic manipulator and the development of SPOT 2.0 - the latest iteration of the software environment used to perform experiments.

2.1 Introduction

The facility at Carleton University has two main components: a large gravity offset table and two spacecraft simulator platforms. The platforms utilize compressed air and air bearings to create a near-frictionless environment. These platforms are self-contained and completely autonomous during any experiments. A laboratory computer is used to upload the guidance, navigation, and control (GN&C) software, initiate experiments, and download the experimental data at the end of each test [40].

The platforms - shown in Fig. 2.1 alongside the gravity offset table - utilize a Raspberry Pi 3 (RP3) running a Raspbian Jessie Linux distribution, allowing for the execution of control commands and the logging of data while also providing an interface for the various instruments on-board. The GN&C software is first designed in MATLAB-Simulink, where simulations can be performed prior to running an experiment. Following the simulations, the Simulink diagram is exported to an executable onboard the RP3. Once the main diagram has been designed, the experiment can be initiated via GUI. Any instrumentation on the platform is integrated into the GN&C software using custom Simulink driver blocks.

Figure 2.1: SPOT platforms and gravity offset table.

The platforms are capable of performing a multitude of different experiments such as formation flying manoeuvres, stabilization of non-cooperative targets, and robotic manipulation tasks. Several experiments have already been performed using the experimental testbed, including an experimental validation for tethered capture of a spinning space debris [40, 41] as well as iterative learning control of spacecraft proximity operations based on confidence level [42].

To determine the position of the platforms during an experiment, an array of eight PhaseSpace motion capture cameras track the inertial position of light emitting diodes (LEDs), each with their own unique pulse frequency. The reported resolution of this camera system is 0.01 mm, and position measurements are made in x , y , and z directions. Both platforms have 4 LEDs, with one located at each corner of the platform's top panel. Using these, the PhaseSpace software can derive the quaternions of the rigid body to provide a measurement of the attitude, as well as track the positions of the spacecraft in inertial space. There are also 3 LEDs on the robotic manipulator, which are used to track the positions of the elbow joint, the wrist joint, and the EE in inertial space. The LED positions for the RED platform are shown in Fig. 2.2.

Figure 2.2: RED platform with labels for each LED.

In order to obtain deployment manoeuvres with a high level of dynamic fidelity, it was necessary to accurately determine the mass properties for the RED and BLACK platforms, as well as the robotic manipulator.

2.2 Spacecraft Properties

The measurement of the center of mass (COM) was performed using the three point method as described in Boyton [43]. This technique requires three scales in the configuration shown in Fig. 2.3.

Figure 2.3: Configuration for COM measurement.

A free-body diagram for this configuration would take the form:

Figure 2.4: Free-body diagram for COM measurement.

From this free-body diagram, the following equations can be derived by taking the moments about A [43]:

$$\begin{aligned} x &= \frac{(B + C)E}{W} \\ y &= \frac{(C - B)D}{2W} \end{aligned} \tag{2.1}$$

where $A \in \mathbb{R} \geq 0$, $B \in \mathbb{R} \geq 0$, and $C \in \mathbb{R} \geq 0$ are the force readings at each corner (or, in this case, the scale measurement multiplied by the gravity constant), and $W \in \mathbb{R} \geq 0$ is the total of all three force measurements. The final masses and COM locations for each platform are summarized in Table 2.1.

Table 2.1: Mass properties and COM for each SPOT platform.

Platform	m (kg)	x (m)	y (m)
RED	16.9478	-0.0070	0.0146
BLACK	12.3341	0.0131	-0.0149

Note, both estimations included a full air tank. A visual representation of the COM and relative LED positions is shown in Fig. 2.5.

Figure 2.5: COM for RED (left) and BLACK (right).

To obtain the mass moment of inertia for the base of the spacecraft (i.e., the configuration shown in Fig. 2.3), model fitting was used. For a planar rigid body in a frictionless environment, the attitude dynamics are given by the following:

$$\tau = I\alpha \quad (2.2)$$

where $\alpha \in \mathbb{R}$ is the angular acceleration of the body, $I \in \mathbb{R} > 0$ is the inertia, and $\tau \in \mathbb{R}$ is the torque applied to the body. Therefore, by applying a known torque using a reaction wheel and measuring the angular acceleration, it is possible to estimate the inertia of the platform by matching the experiment data to the dynamics model through the parameter estimation tool in Simulink. The result of this parameter estimation is presented in Fig. 2.6, and is the result of applying a torque of 0.035 Nm for the duration of the experiment. For the purpose of this thesis, it is assumed that the torque measurement from the reaction wheel is accurate and thus can be used here to estimate the mass moment of inertia. However, if the torque measurement should prove inaccurate, then this estimation would need to be redone.

Figure 2.6: RED platform - mass moment of inertia estimation.

The final estimated inertia can then be compared to the inertia for a simple solid cube with uniform density and the mass of the platform, using the following equation:

$$I = \frac{1}{12}m(w^2 + d^2) \quad (2.3)$$

where $w \in \mathbb{R} \geq 0$ is the width of the platform (0.3 m) and $d \in \mathbb{R} \geq 0$ is the depth of the platform (0.3 m). Note that in the case of the BLACK platform, it is not possible to experimentally estimate the inertia since there is no reaction wheel on-board. Thus, only the mass moment of inertia equation will be used. In the context of this thesis, this estimation is assumed to be accurate enough.

Table 2.2: Inertia properties for each SPOT platform.

Platform	I (kgm ²) - Experimental	I (kgm ²) - Theoretical
RED	0.2873	0.2542
BLACK	—	0.1850

In all simulations, the experimentally obtained inertia estimate will be used for RED, while the theoretical estimate will be used for BLACK.

2.3 Robotic Manipulator Properties

The robotic manipulator used in this thesis is a modified version of the pro-series robotic arms available from CrustCrawler (<http://www.crustcrawler.com/products/ProRoboticArm/>). The list of all components and their prices (as of February 2018) can be found in Table 2.3.

Table 2.3: Components required to build the robotic manipulator.

Product Name	Price (CAD)	Quantity	Source
Girder	\$53.58	2	http://www.crustcrawler.com
Axis Mount	\$18.69	4	http://www.crustcrawler.com
MX-64	\$453.53	3	http://www.crustcrawler.com
Power Supply	\$110.89	1	http://www.crustcrawler.com
USB2Dynamixel	\$67.17	1	https://www.robotshop.com/ca/
SMPS2Dynamixel	\$7.90	1	https://www.robotshop.com/ca/

Building the manipulator is a fairly straightforward process, though some modifications had to be made to ensure compatibility with the spacecraft, as well as improve the rigidity of the manipulator to reduce any vertical deflection. The final configuration and the exploded view of a linkage can be found in Fig. 2.7.

Figure 2.7: Assembled (top) and exploded view (bottom) of a link.

The completed manipulator consists of two links in the configuration shown in Fig. 2.7 attached in series, followed by the electromagnetic gripper shown in Fig. 2.8.

Figure 2.8: Exploded view of the magnetic gripper/dock.

This gripper/dock combination was designed by Justin Kernot, another graduate student. The gripper component is attached to the distal link of the robotic manipulator. Once the gripper is inserted into the dock, the electromagnet on the BLACK platform is turned on, thus ensuring a rigid capture. The fully assembled robotic manipulator can be seen in Fig 2.9.

Figure 2.9: Photo of the fully assembled robotic manipulator.

Each joint is actuated by a Dynamixel MX-64 actuator. These actuators are called “smart” actuators, in reference to the fact that they contain a controller and sensors in addition to the motor itself. The hardware specifications are summarized

in Table 2.4.

Table 2.4: Dynamixel MX-64 hardware specifications.

Parameter	Value
Stall Torque (12 V)	6.0 Nm
No-load Speed	63 RPM
Weight	126 g
Resolution	0.088 deg

The control logic for the internal proportional-integral-derivative (PID) position controller implemented on-board each actuator can be found in Fig. 2.10.

Figure 2.10: Dynamixel MX-64 PID control logic (<https://goo.gl/1wAM6R>).

In order to command a certain torque, a calibration curve is required to convert the current into a torque. The calibration curve provided by Dynamixel is assumed to be correct, and is presented in Fig. 2.11.

Figure 2.11: Calibration curve to convert current into torque.

The curve can be described by the following linear equation:

$$\tau = 0.0056118C_b - 0.04459 \quad (2.4)$$

where $C_b \in \mathbb{R} \geq 0$ is the current in bits. To determine the COM of each link, two scales in the configuration below were used:

Figure 2.12: Configuration for COM measurement (link).

This can then be represented by the following free-body diagram:

Figure 2.13: Free-body diagram of a link for COM measurement.

Knowing the weight at each end of the link, the COM can be calculated by taking the sum of moments about w_T . The resulting general equation takes the form of:

$$d_1 = \frac{w_2(d_T)}{(w_1 + w_2)} \quad (2.5)$$

where $d_1 \in \mathbb{R} \geq 0$ is the distance between the contact point with scale #1 and the COM, $w_1 \in \mathbb{R} \geq 0$ is the weight measured at scale #1, $w_2 \in \mathbb{R} \geq 0$ is the weight measured at scale #2, and $d_T \in \mathbb{R} \geq 0$ is the distance between both scales. The final masses and COM locations for each link are summarized in Table 2.5.

Table 2.5: Mass properties for each link.

Link i	m_i (kg)	a_i (m)	b_i (m)
Link #1 (Proximal)	0.3377	0.1933	0.1117
Link #2 (Medial)	0.3281	0.1993	0.1057
Link #3 (Distal)	0.0111	0.0621	0.0159

Measurement of the mass moment of inertia for the linkages was first performed using the bifilar pendulum (BP) technique [44], then validated through experimental measurement. In brief, the BP technique involves suspending the rigid body at two points equidistant from its COM, applying a torque to induce oscillations, and then measuring the frequency of said oscillations. The experimental setup is shown below in Fig. 2.14.

Figure 2.14: Experimental setup for inertia measurement using BP technique.

Once the frequency of the oscillations has been measured, the mass moment of inertia can be approximated as:

$$I = \frac{mgd_c^2}{4L_c\omega^2} \quad (2.6)$$

where $d_c \in \mathbb{R} \geq 0$ is the distance between the suspension cables, $L_c \in \mathbb{R} \geq 0$ is the length of the cables, and $\omega \in \mathbb{R} \geq 0$ is the frequency of the oscillations.

To validate the mass moment of inertia of the links obtained using the PS method, each link was added to the body and an inertia estimate of the entire body was experimentally obtained using the technique presented in Sec. 2.2 - an example of the result is shown in Fig. 2.15 for link #1. Then, knowing the change in COM for the entire body, the inertia of the body with and without each link, and the COM/mass of the added link, it is possible to extract the inertia via parallel axis theorem.

Figure 2.15: RED platform - mass moment of inertia with link #1.

Once the inertia of the base/link combination is known, parallel axis theorem can be used to obtain an equation that may be used to calculate the mass moment of the link at it's own COM. The final equation used takes the form:

$$I_{link} = (I_{base/link} - I_{base}) - m_{base}\Delta d_1^2 - m_{link}\Delta d_2^2 \quad (2.7)$$

where $I_{link} \in \mathbb{R} \geq 0$ is the mass moment of inertia for the linkage at it's COM, $I_{base/link} \in \mathbb{R} \geq 0$ is the mass moment of inertia of the base/link combination at it's COM, $I_{base} \in \mathbb{R} \geq 0$ is the mass moment of inertia of the base at it's COM, $m_{base}, m_{link} \in \mathbb{R} \geq 0$ are the masses of the base and linkages respectively, $\Delta d_1 \in \mathbb{R} \geq 0$ is the distance between the COM of the base and the COM of the base/link combination, and $\Delta d_2 \in \mathbb{R} \geq 0$ is the distance between the COM of the linkage and the COM of the base/link combination.

This process was repeated for each link. The final inertia measurements obtained using each technique are summarized below:

Table 2.6: Inertia properties for each link.

Link i	I_i (kgm ²) BP Method	I_i (kgm ²) Experimental
Link #1 (Proximal)	0.003750	0.004171
Link #2 (Medial)	0.003413	0.003967
Link #3 (Distal)	0.00005640	0.00006722

In all simulations, the estimations obtained using the BP method will be used.

2.4 Development of SPOT 2.0 Software

In addition to the robotic manipulator, the other major contribution of the author to the facility is the development of SPOT 2.0. This latest update to the experiment software includes many major improvements to both performance and ease of use.

The redesign of the software is the result of lessons learned from the previous iteration of the software. Specifically, the following issues were the primary problems addressed in the latest version of the software:

- **Variable Rates:** Though Simulink is inherently capable of handling multi-rate blocks, the original design of the software prevented users from taking advantage of this feature. Originally, any experiment diagram was converted into C++ code and manually compiled into a Linux executable. This forced the experiment to run at a pre-defined rate, which could cause inconsistencies between the rates in the diagram and the so-called “global” rate defined during code compilation. This issue was remedied by using the **Simulink Support Package for Raspberry Pi Hardware**. This toolbox provides options in Simulink that allow the diagram to be compiled directly into a Linux executable. The rate at which the diagram operates can now be set at the top level, but inter-multiples of the rate are allowed in subsystems if desired.
- **Hardware Access:** In order to operate any external hardware - including the robotic manipulator, PhaseSpace cameras, flotation pucks, thrusters, and inertial measurement unit - it was initially necessary to use S-Function blocks. Though theoretically just as capable as the driver blocks used in SPOT 2.0,

the S-Function blocks required a significant number of extra files in order to function correctly. These extra files added to the existing clutter in the software, decreasing both its performance and usability. The S-Function blocks were replaced by dedicated driver blocks for each piece of hardware. As of January 2018, driver blocks have been developed for the PhaseSpace camera system, the Dynamixel actuators, and GPIO functionality.

- **Graphical User Interface:** The lack of a user-friendly interface in the original software was highly detrimental to new graduate students, and it was decided early on in the development of SPOT 2.0 that a new graphical user interface was a requirement. The framework for the SPOT 2.0 GUI was created using MATLABs GUIDE tool. Custom code was then required to build desired functionality into the interface.
- **User-Defined Functions:** Prior to SPOT 2.0, any user-defined function - which is to say any function not composed of Simulink blocks - had to be written in C++ code and implemented using S-Function blocks. This further added to the number of unnecessary files, which decreased overall performance. By compiling the diagram into an executable directly from Simulink, user-defined functions in MATLAB can now be used. This improves the overall usability of the software, since most graduate students are more familiar with MATLAB code than C++ code.

For the experiments performed in Chap. 4, v2.10 of the experiment software was used. Instructions on setting up this version of the software can be found in Appendix A. However, the experiments performed in Chap. 5 required some significant changes, resulting in a newer version - v2.11 - of the software. Since the user manual for the latest version is still in progress, only the instructions for v2.10 are provided in this thesis. This version is functional, and the changes made for v2.11 are significant only in the context of the controller presented in Chap. 5.

Chapter 3

Kinematics and Dynamics of Space Robots

In this chapter, the kinematics and dynamics for the general case of a space robot are derived. The chapter will also include the case-specific dynamics for the planar space robot used in SPOT - shown in Fig. [2.2](#).

3.1 Primary Assumptions

Determining the kinematics and dynamics model for a space robot is a problem that has been addressed several times in the past, but will still be described in detail here for completeness. First, the derivation of the dynamics and kinematics model chosen for this thesis requires the following assumptions:

1. The spacecraft is assumed to be a rigid body. This assumption allows for a simpler dynamics model which neglects the effect of flexible linkages.
2. The friction in the joints - while significant enough to render experimental model validation difficult, as explained in Sec. [3.6](#) - are not so significant that the techniques proposed in this thesis will not function properly, and are therefore neglected in the dynamics model to reduce computation time.
3. Since the primary focus of this thesis is the experimental validation of each technique in the laboratory used at Carleton University, no external perturbations typically found on-orbit are included in the model.
4. The kinematics and dynamics are exclusively planar in nature. No considerations are made in the model development for spatial motion; this includes the techniques proposed in this thesis which make use of the model developed in this chapter.

3.2 Kinematics Model

The forward kinematics for a space robot involves determining the EE position in inertial space. This can be geometrically written as:

$$\mathbf{p}_e = \mathbf{p}_0 + \mathbf{b}_0 + \sum_{i=1}^n \mathbf{L}_i \quad (3.1)$$

where $\mathbf{p}_e \in \mathbb{R}^3$ is the position vector pointing from the origin of the inertial reference frame to the EE, $\mathbf{p}_0 \in \mathbb{R}^3$ is the position vector pointing from the origin of the inertial reference frame to the COM of the spacecraft body, $\mathbf{b}_0 \in \mathbb{R}^3$ is the position vector pointing from the COM to the first joint i , $\mathbf{L}_i \in \mathbb{R}^3$ is the position vector pointing from joint i to joint $i + 1$, and n is the number of links forming the manipulator.

3.3 Dynamics Model

The nonlinear dynamic equation of a multi-link robot with rigid links is derived in terms of kinetic and potential energies stored in the system by the Euler-Lagrange formulation, as follows [45]:

$$L = T - U \quad (3.2)$$

where $L \in \mathbb{R}$ is the Lagrangian, $T \in \mathbb{R}$ is the kinetic energy component of the Lagrange equation, and $U \in \mathbb{R}$ is the potential energy component of the Lagrange equation. The kinetic and potential energies are defined in terms of generalized coordinates $\mathbf{q} \in \mathbb{R}^n$, which encompasses the position of the base in inertial space as well as the joint angles. The equations of motion for a free-floating robot subjected to joint torques $\tau_i \in \mathbb{R}$ are derived according to the following equation:

$$\tau_i = \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) - \left(\frac{\partial L}{\partial q_i} \right) \quad (3.3)$$

The kinetic energy is assumed to be a quadratic function of the generalized coordinates $\mathbf{q}$ of the form:

$$T = \frac{1}{2} \sum_{i=0}^n \sum_{j=0}^n M_{ij} \dot{q}_i \dot{q}_j \quad (3.4)$$

where $M_{ij} \in \mathbb{R}$ are the components of the robots inertia matrix, denoted $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n \times n}$, which is symmetric and positive definite; and where $\dot{q}_i \in \mathbb{R}$ is the generalized joint velocity for joint i . In the case of a free-floating robot with rigid links, there is no source of potential energy. Therefore, the Lagrangian can be simplified to:

$$L = \frac{1}{2} \sum_{i=0}^n \sum_{j=0}^n M_{ij} \dot{q}_i \dot{q}_j \quad (3.5)$$

The partial derivative of the Lagrangian with respect to the k^{th} joint position is given by:

$$\frac{\partial L}{\partial q_k} = \frac{1}{2} \sum_{i=0}^n \sum_{j=0}^n \frac{\partial M_{ij}}{\partial q_k} \dot{q}_i \dot{q}_j \quad (3.6)$$

Similarly, the partial derivative of the Lagrangian with respect to the k^{th} joint velocity is:

$$\frac{\partial L}{\partial \dot{q}_k} = \sum_{j=0}^n M_{kj} \dot{q}_j \quad (3.7)$$

Therefore, the derivative of Eq. (3.7) with respect to time is:

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_k} \right) = \sum_{j=0}^n M_{kj} \ddot{q}_j + \sum_{i=0}^n \sum_{j=0}^n \frac{\partial M_{ij}}{\partial q_i} \dot{q}_i \dot{q}_j \quad (3.8)$$

Thus, for each joint, the Euler-Lagrange equations are given by:

$$\tau_k = \sum_{j=0}^n M_{kj} \ddot{q}_j + \sum_{i=0}^n \sum_{j=0}^n \left(\frac{\partial M_{ij}}{\partial q_i} - \frac{1}{2} \frac{\partial M_{ij}}{\partial q_k} \right) \dot{q}_i \dot{q}_j \quad (3.9)$$

It can then be shown that:

$$\sum_{i=0}^n \sum_{j=0}^n \left(\frac{\partial M_{ij}}{\partial q_i} - \frac{1}{2} \frac{\partial M_{ij}}{\partial q_k} \right) \dot{q}_i \dot{q}_j = \sum_{i=0}^n \sum_{j=0}^n c_{ijk} \dot{q}_i \dot{q}_j \quad (3.10)$$

where the terms c_{ijk} are known as the Christoffel symbols. The Euler-Lagrange

equations of motion can then be written as:

$$\tau_k = \sum_{j=0}^n M_{ij} \ddot{q}_j + \sum_{i=0}^n \sum_{j=0}^n c_{ijk} \dot{q}_i \dot{q}_j \quad (3.11)$$

This expression can be rewritten once more into a more intuitive matrix form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \boldsymbol{\tau} \quad (3.12)$$

where the k, j^{th} element of the centrifugal/Coriolis matrix as denoted by $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n \times n}$ is defined as:

$$C_{ij} = \sum_{i=1}^n c_{ijk} \dot{q}_i \quad (3.13)$$

From here, Eq. (3.12) can then be partitioned into the following [46]:

$$\begin{bmatrix} \mathbf{M}_b & \mathbf{M}_{bm} \\ \mathbf{M}_{bm}^T & \mathbf{M}_m \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{x}}_b \\ \ddot{\boldsymbol{\theta}} \end{bmatrix} + \begin{bmatrix} \mathbf{c}_b \\ \mathbf{c}_m \end{bmatrix} = \begin{bmatrix} \mathbf{f}_b \\ \boldsymbol{\tau}_c \end{bmatrix} + \begin{bmatrix} \mathbf{J}_b^T \\ \mathbf{J}_m^T \end{bmatrix} \mathbf{f}_e \quad (3.14)$$

where $\mathbf{M}_b \in \mathbb{R}^{6 \times 6}$ is the inertia matrix for the base, $\mathbf{M}_m \in \mathbb{R}^{n \times n}$ is the inertia matrix for the links, $\mathbf{M}_{bm} \in \mathbb{R}^{6 \times n}$ is the coupled inertia matrix, $\ddot{\mathbf{x}}_b \in \mathbb{R}^6$ is the linear and angular acceleration vector for the base spacecraft, $\ddot{\boldsymbol{\theta}} \in \mathbb{R}^n$ is the manipulator joint angular acceleration vector, $\mathbf{c}_b \in \mathbb{R}^6$ is the velocity dependent nonlinear matrix for the base, $\mathbf{c}_m \in \mathbb{R}^n$ is the velocity dependent nonlinear matrix for the manipulator, $\mathbf{f}_b \in \mathbb{R}^6$ is the vector of forces and moments exerted on the COM of the base, $\mathbf{f}_e \in \mathbb{R}^6$ is the vector of forces and moments exerted on the EE, $\boldsymbol{\tau}_c \in \mathbb{R}^n$ is the vector of joint torques, $\mathbf{J}_b \in \mathbb{R}^{6 \times 6}$ is the Jacobian matrix for the base, and finally $\mathbf{J}_m \in \mathbb{R}^{6 \times n}$ is the Jacobian matrix for the manipulator.

3.4 Generalized Jacobian

It is important to review the concept of the generalized Jacobian here for a number of reasons. First, in the case of the NDO proposed in Chap. 5, the generalized Jacobian is used to calculate the EE disturbance forces based on the disturbance torques estimated by the NDO. Second, the transpose Jacobian control law used throughout

this thesis requires the analytical expression for the generalized Jacobian.

The concept of the generalized Jacobian was first introduced by Umetani and Yoshida [47]. The generalized Jacobian can be expressed in terms of the partitions presented in Eq. (3.14):

$$\mathbf{J}^* = \mathbf{J}_m - \mathbf{J}_b \mathbf{M}_b^{-1} \mathbf{M}_{bm} \quad (3.15)$$

where $\mathbf{J}^* \in \mathbb{R}^{6 \times n}$ is the generalized Jacobian matrix. Using this matrix, past work in the field has established that any forces acting at the EE can be determined from the joint torques by pre-multiplying the disturbance torque vector by the pseudo-inverse transpose of the generalized Jacobian matrix [47, 48]:

$$\mathbf{f}_e = (\mathbf{J}^{*+})^T \mathbf{d} \quad (3.16)$$

where $\mathbf{d} \in \mathbb{R}^n$ is the vector of disturbance torques. It is also possible to use the transpose of the generalized Jacobian matrix to develop a simple control law for a space robot, referred to in this thesis as the transpose Jacobian control law:

$$\boldsymbol{\tau}_c = (\mathbf{J}^*)^T \left(\mathbf{K}_p \mathbf{x}_e + \mathbf{K}_d \frac{d\mathbf{x}_e}{dt} + \mathbf{K}_I \int \mathbf{x}_e dt \right) \quad (3.17)$$

where $\mathbf{x}_e \in \mathbb{R}^6$ is the error between the reference and actual EE pose, $\mathbf{K}_p \in \mathbb{R}^{6 \times 6}$ is the proportional gain matrix, $\mathbf{K}_d \in \mathbb{R}^{6 \times 6}$ is the derivative gain matrix $\mathbf{K}_I \in \mathbb{R}^{6 \times 6}$ is the integral gain matrix. Note that these dimensions assume each DOF is a 6-DOF joint: the dimensions will differ for different types of joints.

3.5 Kinematics and Dynamics of the SPOT Space Robot

In this section, the kinematics, inverse kinematics, and dynamics for the experimental platform shown in Fig. 2.2 are derived.

Figure 3.1: Representation of the reference frame and generalized states.

Referring to the geometry of the planar space robot shown in Fig. 3.1, the kinematics at position-level for this system can be derived using Eq. (3.1). The resulting equations are given by:

$$\begin{aligned}
 x_{ee} = & x_0 + b_0 \cos(\phi + q_0) + (a_1 + b_1) \cos(\pi/2 + q_0 + q_1) \\
 & + (a_2 + b_2) \cos(\pi/2 + q_0 + q_1 + q_2) \\
 & + (a_3 + b_3) \cos(\pi/2 + q_0 + q_1 + q_2 + q_3)
 \end{aligned} \tag{3.18}$$

$$\begin{aligned}
 y_{ee} = & y_0 + b_0 \sin(\phi + q_0) + (a_1 + b_1) \sin(\pi/2 + q_0 + q_1) \\
 & + (a_2 + b_2) \sin(\pi/2 + q_0 + q_1 + q_2) \\
 & + (a_3 + b_3) \sin(\pi/2 + q_0 + q_1 + q_2 + q_3)
 \end{aligned} \tag{3.19}$$

where $x_{ee} \in \mathbb{R}$ and $y_{ee} \in \mathbb{R}$ are the scalar components of the EE position in the inertial frame, $x_0 \in \mathbb{R}$ and $y_0 \in \mathbb{R}$ are the positions of spacecraft COM in the inertial frame, $q_i \in \mathbb{R}$ is the joint angle for each body i , $b_0 \in \mathbb{R} \geq 0$ is the magnitude of the

vector $\mathbf{b}_0$, $a_i \in \mathbb{R}_{\geq 0}$ is distance from joint i to the COM of link i , $b_i \in \mathbb{R}_{\geq 0}$ is distance from the COM of link i and joint $i+1$, and $\phi \in \mathbb{R}$ is the angle describing the location of the first joint in the body-fixed reference frame. The scalar components of the EE velocity, $\dot{x}_{ee} \in \mathbb{R}$ and $\dot{y}_{ee} \in \mathbb{R}$, can be obtained by differentiating Eq. (3.18) and Eq. (3.19) with respect to time.

Classically, the inverse kinematics for a space robot is concerned with determining the states of the system knowing the inertial position of the EE. In this thesis, inverse kinematics are applied to determine the manipulator joint angles during any given experiment, as the joint encoders do not operate at a fast enough rate to be usable. Given the locations of the LEDs for the PhaseSpace camera system, as detailed in Chap. 2, the following assumption is made:

- The inertial location and attitude of the spacecraft, as well as the EE position, wrist position, and elbow position, are all measured quantities during any experiment.

Thus, given the assumption listed above, the shoulder joint angle can be obtained using:

$$q_1 = \text{atan2}(C_1, C_2) - \frac{\pi}{2} - q_0 \quad (3.20)$$

where:

$$C_1 = y_{el} - y_0 - b_0 \sin(\phi + q_0)$$

$$C_2 = x_{el} - x_0 - b_0 \cos(\phi + q_0)$$

Then, the elbow and wrist joints can be obtained from:

$$q_2 = \text{atan2}(y_w - y_{el}, x_w - x_{el}) - \frac{\pi}{2} - q_0 - q_1 \quad (3.21)$$

and:

$$q_3 = \text{atan2}(y_{ee} - y_w, x_{ee} - x_w) - \frac{\pi}{2} - q_0 - q_1 - q_2 \quad (3.22)$$

where $x_w \in \mathbb{R}$ and $y_w \in \mathbb{R}$ are the scalar components of the wrist position in the inertial frame, and $x_{el} \in \mathbb{R}$ and $y_{el} \in \mathbb{R}$ are the scalar components of the elbow position in the inertial frame. It is important to note that these joint angles must be wrapped such that they fall in the range of $[-2\pi, 2\pi]$. This can be done using the following equation:

$$\theta_{wrapped} = \theta - 2\pi \text{ floor} \left(\frac{(\theta + \pi)}{2\pi} \right) \quad (3.23)$$

where $\theta \in \mathbb{R}$ is the angle to be wrapped. Note that the `floor` function used in the above equation simply rounds the value down. For any n -link manipulator, the inertia matrix $\mathbf{M}(\mathbf{q})$ can be derived in terms of the linear and angular velocity components of the kinetic energy of the system, which are functions of the Jacobian matrix and the derivatives of the joint variables. The Jacobian matrix is a more convenient way to formulate the velocity kinematics of the manipulator. A suitable Jacobian that relates the linear and angular velocity of the COM for each body, denoted by $\mathbf{v}_c, \boldsymbol{\omega}_c \in \mathbb{R}^3$, to the joint velocity, can be obtained as follows:

$$\mathbf{v}_c = \mathbf{J}_v(\mathbf{q})\dot{\mathbf{q}} \quad (3.24)$$

$$\boldsymbol{\omega}_c = \mathbf{J}_\omega(\mathbf{q})\dot{\mathbf{q}} \quad (3.25)$$

where by $\mathbf{J}_v(\mathbf{q}), \mathbf{J}_\omega(\mathbf{q}) \in \mathbb{R}^{3 \times 6}$, Eq. (3.24) and Eq. (3.25) can be grouped together and represented by:

$$\begin{bmatrix} v_{cx} & v_{cy} & v_{cz} & \omega_{cx} & \omega_{cy} & \omega_{cz} \end{bmatrix}^T = \mathbf{J}_c \begin{bmatrix} \dot{x}_0 & \dot{y}_0 & \dot{q}_0 & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 \end{bmatrix}^T \quad (3.26)$$

where:

$$\mathbf{J}_c = \begin{bmatrix} \mathbf{J}_v(\mathbf{q}) \\ \mathbf{J}_\omega(\mathbf{q}) \end{bmatrix} \quad (3.27)$$

Then, the Jacobian for each body is given by:

$$\mathbf{J}_{c_0}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{v_0}(\mathbf{q}) \\ \mathbf{J}_{\omega_0}(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (3.28)$$

Similarly, this can be done for the first joint as:

$$\mathbf{J}_{c_1}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{v_1}(\mathbf{q}) \\ \mathbf{J}_{\omega_1}(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{J}_{c_1}[1, 3] & \mathbf{J}_{c_1}[1, 4] & 0 & 0 \\ 0 & 1 & \mathbf{J}_{c_1}[2, 3] & \mathbf{J}_{c_1}[2, 4] & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \quad (3.29)$$

Then, for the second joint, the Jacobian is given by:

$$\mathbf{J}_{c_2}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{v_2}(\mathbf{q}) \\ \mathbf{J}_{\omega_2}(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{J}_{c_2}[1, 3] & \mathbf{J}_{c_2}[1, 4] & \mathbf{J}_{c_2}[1, 5] & 0 \\ 0 & 1 & \mathbf{J}_{c_2}[2, 3] & \mathbf{J}_{c_2}[2, 4] & \mathbf{J}_{c_2}[2, 5] & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \quad (3.30)$$

Finally, for the third joint, the Jacobian is:

$$\mathbf{J}_{c_3}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{v_3}(\mathbf{q}) \\ \mathbf{J}_{\omega_3}(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \mathbf{J}_{c_3}[1, 3] & \mathbf{J}_{c_3}[1, 4] & \mathbf{J}_{c_3}[1, 5] & \mathbf{J}_{c_3}[1, 6] \\ 0 & 1 & \mathbf{J}_{c_3}[2, 3] & \mathbf{J}_{c_3}[2, 4] & \mathbf{J}_{c_3}[2, 5] & \mathbf{J}_{c_3}[2, 6] \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (3.31)$$

where:

$$\begin{aligned}
\mathbf{J}_{c_1}[1, 3] &= -b_0 S_0 - a_1 S_1 \\
\mathbf{J}_{c_1}[1, 4] &= -a_1 S_1 \\
\mathbf{J}_{c_1}[2, 3] &= b_0 C_0 + a_1 C_1 \\
\mathbf{J}_{c_1}[2, 4] &= a_1 C_1 \\
\mathbf{J}_{c_2}[1, 3] &= -b_0 S_0 - L_1 S_1 - a_2 S_2 \\
\mathbf{J}_{c_2}[1, 4] &= -L_1 S_1 - a_2 S_2 \\
\mathbf{J}_{c_2}[1, 5] &= -a_2 S_2 \\
\mathbf{J}_{c_2}[2, 3] &= b_0 C_0 + L_1 C_1 + a_2 C_2 \\
\mathbf{J}_{c_2}[2, 4] &= L_1 C_1 + a_2 C_2 \\
\mathbf{J}_{c_2}[2, 5] &= a_2 C_2 \\
\mathbf{J}_{c_3}[1, 3] &= -b_0 S_0 - L_1 S_1 - L_2 S_2 - a_3 S_3 \\
\mathbf{J}_{c_3}[1, 4] &= -L_1 S_1 - L_2 S_2 - a_3 S_3 \\
\mathbf{J}_{c_3}[1, 5] &= -L_2 S_2 - a_3 S_3 \\
\mathbf{J}_{c_3}[1, 6] &= -a_3 S_3 \\
\mathbf{J}_{c_3}[2, 3] &= b_0 C_0 + L_1 C_1 + L_2 C_2 + a_3 C_3 \\
\mathbf{J}_{c_3}[2, 4] &= L_1 C_1 + L_2 C_2 + a_3 C_3 \\
\mathbf{J}_{c_3}[2, 5] &= L_2 C_2 + a_3 C_3 \\
\mathbf{J}_{c_3}[2, 6] &= a_3 C_3
\end{aligned}$$

and:

$$\begin{aligned}
S_0 &= \sin(\phi + q_0) \\
S_1 &= \sin(\pi/2 + q_0 + q_1) \\
S_2 &= \sin(\pi/2 + q_0 + q_1 + q_2) \\
S_3 &= \sin(\pi/2 + q_0 + q_1 + q_2 + q_3) \\
C_0 &= \cos(\phi + q_0) \\
C_1 &= \cos(\pi/2 + q_0 + q_1) \\
C_2 &= \cos(\pi/2 + q_0 + q_1 + q_2) \\
C_3 &= \cos(\pi/2 + q_0 + q_1 + q_2 + q_3)
\end{aligned}$$

Hence, the translational contribution to the total kinetic energy, $\mathbf{T}_v \in \mathbb{R}^{6 \times 6}$, is:

$$\mathbf{T}_v = \frac{1}{2} \dot{\mathbf{q}}^T (m_0 \mathbf{J}_{v_0}^T \mathbf{J}_{v_0} + m_1 \mathbf{J}_{v_1}^T \mathbf{J}_{v_1} + m_2 \mathbf{J}_{v_2}^T \mathbf{J}_{v_2} + m_3 \mathbf{J}_{v_3}^T \mathbf{J}_{v_3}) \dot{\mathbf{q}} \quad (3.32)$$

Similarly, the rotational contribution to the total kinetic energy, $\mathbf{T}_\omega \in \mathbb{R}^{6 \times 6}$, is:

$$\mathbf{T}_\omega = \frac{1}{2} \dot{\mathbf{q}}^T (I_0 \mathbf{J}_{\omega_0}^T \mathbf{J}_{\omega_0} + I_1 \mathbf{J}_{\omega_1}^T \mathbf{J}_{\omega_1} + I_2 \mathbf{J}_{\omega_2}^T \mathbf{J}_{\omega_2} + I_3 \mathbf{J}_{\omega_3}^T \mathbf{J}_{\omega_3}) \dot{\mathbf{q}} \quad (3.33)$$

Therefore, the inertia matrix $\mathbf{M}(\mathbf{q})$ for the case presented in this thesis is given by the sum of the linear and angular kinetic energy, and results in the following matrix:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} & M_{15} & M_{16} \\ M_{21} & M_{22} & M_{23} & M_{24} & M_{25} & M_{26} \\ M_{31} & M_{32} & M_{33} & M_{34} & M_{35} & M_{36} \\ M_{41} & M_{42} & M_{43} & M_{44} & M_{45} & M_{46} \\ M_{51} & M_{52} & M_{53} & M_{54} & M_{55} & M_{56} \\ M_{61} & M_{62} & M_{63} & M_{64} & M_{65} & M_{66} \end{bmatrix} \quad (3.34)$$

where $M_{ij} \forall i = 1 \dots 6, j = 1 \dots 6$ are defined in Appendix B. Using the components of the inertia matrix, the Christoffel symbols can be derived using Eq. (3.10). Using the Christoffel symbols with Eq. (3.13), the centrifugal/Coriolis matrix can be obtained:

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{bmatrix} \quad (3.35)$$

where $C_{ij} \forall i = 1 \dots 6, j = 1 \dots 6$ are also defined in Appendix B.

3.6 A Note on Model Validation

Due to the frictionless joints assumption made in this thesis, experimental model validation could not be performed. However, as demonstrated by the experimental results presented in this thesis, the friction does not have any major impact on the performance of the proposed techniques. In brief:

- For the guidance problem solved in Chap. 4, the effect of the joint torques is negligible due to the nature of the guidance problem. Since the joint controllers are simply instructed to follow a set path over a set time, they will overcome the friction and the resulting torque should be very similar.
- For the capture problem solved in Chap. 5, the NDO is not required to accurately estimate the EE force. The NDO is only required to provide an *estimate*, and the impedance controller gains are tuned accordingly. Even if a joint friction model was included in this thesis, the result would be a different EE force estimation, which would then require that the gains be tuned accordingly.

However, it is important to note that several aspects of the spacecraft have been validated. All inertia values were experimentally validated in Chap. 2, alongside the basic model for a planar body in a frictionless environment.

Chapter 4

Optimal Trajectory Planning for Space Robots

Here, the guidance problem for a free-floating robot is solved via PS methods. To provide readers with an introduction to PS methods, the technique used by GPOPS-I is summarized, along with the NLP solver SNOPT. Following the introduction, the optimal guidance problem is defined and simulations are performed. A validation campaign concludes this chapter to ensure that the results from the simulation are optimal. The validation campaign includes Pontryagin’s minimum principle, Bellman’s principle of optimality, the propagation of the results, and experiments performed using SPOT. All relevant code for this chapter can be found in [Appendix C](#).

4.1 Approach Overview

In this thesis, the optimal trajectory problem was defined and solved within the MATLAB-Simulink environment using three different commercial optimization tools. With each tool, some form of spectral discretization was used to transform the continuous problem into a discretized problem. SNOPT was then used to solve the discretized optimal control problem. To provide readers with an introduction to the theory applied by the tools, a summary of the spectral method used in GPOPS-I will be presented. Since the focus of this thesis is on the application and verification of solutions obtained from these tools, a similarly detailed summary of the techniques used in TOMLAB and DIDO is beyond the scope of this thesis. For a more comprehensive description of each tool, interested readers may refer to the following: for GPOPS-I, refer to Benson, Huntington, and Rao [\[49–53\]](#). For further information on DIDO, refer to Ross and Karpenko [\[54, 55\]](#). Finally, for TOMLAB, readers may refer to the user manual [\[56\]](#), as well as Gill et al. [\[57\]](#).

4.1.1 Gauss Pseudospectral Collocation

Each optimization tool was implemented alongside the dynamics model presented by Eq. (3.12) in Chap. 3 to obtain the results, as well as to validate the capability of the MATLAB-Simulink environment to accurately optimize the manipulator trajectory. Each tool requires the user to define the dynamics of the problem using differential equations, an associated cost function, and finally the limits and initial guesses for each of the states and controls.

In its general form, any optimal guidance/control problem can be formulated so as to minimize the cost function:

$$J = \Phi(\mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} g_L(\mathbf{x}(t), \mathbf{u}(t), t) dt \quad (4.1)$$

where $\Phi \in \mathbb{R}$ is the Mayer component, $g_L \in \mathbb{R}$ is the Lagrange component, $J \in \mathbb{R}$ is the cost function, $\mathbf{u} \in \mathbb{R}^n$ is the vector of control inputs, $t_0 \in \mathbb{R} \geq 0$ is the initial time, and $t_f \in \mathbb{R} \geq 0$ is the final time. The vector $\mathbf{x} \in \mathbb{R}^{2n}$ defines the state-space formulation of the dynamics, which can be written as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} \quad (4.2)$$

The system is subject to the dynamic constraints given by:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \quad (4.3)$$

and the associated boundary conditions:

$$\boldsymbol{\beta}(\mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f) = 0 \quad (4.4)$$

where $\boldsymbol{\beta} \in \mathbb{R}$ is the vector of boundary conditions for the guidance problem. The optimal guidance problem defined is solved by GPOPS-I using a direct transcription method called the Gauss PS method [49]. TOMLAB, meanwhile, uses a combined modelling, compilation, and solver engine named PROPT [58] to solve the NLP. However, the underlying PS collocation is similar to GPOPS-I, though TOMLAB has the ability to use Chebychev points [59]. Finally, DIDO uses a unique spectral

algorithm originally developed by Ross [60, 61]. As mentioned earlier, covering the techniques used by each tool is beyond the scope of this thesis, and thus the focus will be placed on the algorithm used in GPOPS-I due to its open-source nature. For a single-phase optimal control problem such as the one presented in this thesis, the Gauss PS method can be summarized as follows. First, the original time interval $t \in [t_0, t_f]$ is transformed to the non-dimentionalized time interval $\eta \in [-1, 1]$ using:

$$t = \frac{(t_f - t_0)\eta + (t_f + t_0)}{2} \quad (4.5)$$

Following the temporal transformation in Eq. (4.5), the cost function in Eq. (4.1) can be rewritten in terms of η , such that the following equation is obtained:

$$J = \Phi(\mathbf{x}(-1), t_0, \mathbf{x}(1), t_f) + \frac{(t_f - t_0)}{2} \int_{-1}^1 g_L(\mathbf{x}(\eta), \mathbf{u}(\eta), \eta; t_0, t_f) d\eta \quad (4.6)$$

In the same fashion, the dynamic constraints from Eq. (4.3) can be rewritten in terms of η , such that the following equation is obtained:

$$\frac{2}{t_f - t_0} \frac{d\mathbf{x}}{d\eta} = \mathbf{f}(\mathbf{x}(\eta), \mathbf{u}(\eta), \eta; t_0, t_f) \quad (4.7)$$

Then, the boundary conditions can be given in terms of η as:

$$\beta(\mathbf{x}(-1), t_0, \mathbf{x}(1), t_f) = 0 \quad (4.8)$$

The state $\mathbf{x}(t)$ can then be approximated in terms of a basis $N + 1$ Lagrange interpolating polynomial on the interval from $[-1, 1]$ as:

$$\mathbf{x}(t) \approx \mathbf{X}(t) = \sum_{k=0}^N \mathbf{X}(t_k) L_k(t) \quad (4.9)$$

Then, suppose that $N + 1$ points are selected on the interval $\eta \in [-1, 1]$ for discretization of the continuous-time problem. Specifically, the initial point $\eta = -1$ is used, along with N Gauss points $\eta_k \forall k = 1, \dots, N$, which lie on the interval $[-1, 1]$. The approximation of the state derivative at the Gauss points is expressed as:

$$\left[\frac{d\mathbf{x}}{dt} \right]_{t_i} \approx \left[\frac{d\mathbf{X}^N}{dt} \right]_{t_i} = \sum_{k=0}^N \mathbf{X}_k \left(\frac{dL_k}{dt} \right)_{t_i} = \sum_{k=0}^N D_{ik} \mathbf{X}_k = \mathbf{f}(\mathbf{X}_i, \mathbf{U}_i, t_i) \quad (4.10)$$

where D_{ik} is referred to as the *Differentiation Matrix*. The differential operator $\mathbf{D} \in \mathbb{R}^{N \times (N+1)}$ and $\bar{\mathbf{D}} \in \mathbb{R}^N$ are computed using the exact derivative of the Lagrange interpolating polynomial L_k as:

$$\left[\frac{d\mathbf{x}}{dt} \right]_{t_i} = \dot{\mathbf{x}}(t_i) \approx \dot{\mathbf{X}}(t_i) = \mathbf{X}(t_0) \bar{D}_i + \sum_{k=1}^N \mathbf{X}(t_k) D_{ik} \quad (4.11)$$

where $D_{ik} = \dot{L}_k(t_i)$ and $\bar{D}_i = \dot{L}_0(t_i)$ are the differential operators. The optimal control problem can then be discretized using the variables $\mathbf{X}_k \in \mathbb{R}^n$ and $\mathbf{U}_k \in \mathbb{R}^m$ for the states and controls at the Gauss points, respectively, for $k = 1, \dots, N$. The initial and final states, namely $\mathbf{X}(t_0) \in \mathbb{R}^n$ and $\mathbf{X}(t_f) \in \mathbb{R}^n$, respectively, are also included as variables. The continuous-time cost function in Eq. (4.6) is first approximated using a Gauss quadrature, as follows:

$$J = \Phi(\mathbf{x}(1), t_f) + \frac{(t_f - t_0)}{2} \sum_{k=1}^N w_k g_L(\mathbf{X}_k, \mathbf{U}_k, t_k) \quad (4.12)$$

where $w_k \in \mathbb{R}_{\geq 0}$, the Gauss weight at each Gauss point, is given by the equation:

$$w_k = \frac{2}{(1 - (\eta_k)^2)[P_N']^2} \quad (4.13)$$

where P_N' is the N^{th} -degree Legendre polynomial defining Gauss points. The differential equation constraints in Eq. (4.7) are also discretized at the Gauss points as:

$$\frac{2}{t_f - t_0} \bar{D} \mathbf{X}(t_0) + \frac{2}{t_f - t_0} \sum_{k=1}^N D_{ik} \mathbf{X}_k = \mathbf{f}(\mathbf{x}_i, \mathbf{U}_i, t_i), \quad \forall i = 1, \dots, N \quad (4.14)$$

where

$$\begin{bmatrix} \bar{D}_i & D_{ik} \end{bmatrix} = \begin{bmatrix} (dL_0/d\eta)_{\eta_i} & (dL_k/d\eta)_{\eta_i} \end{bmatrix} \in \mathbb{R}^{N \times (N+1)} \quad (4.15)$$

Note, the differential equations are collocated only at the Gauss points, and not at the boundary points [49–53]. Therefore, the discretized boundary conditions are given as:

$$\beta(\mathbf{X}(t_0), t_0, \mathbf{X}(t_f), t_f) = 0 \quad (4.16)$$

Finally, the terminal state, $\mathbf{X}(t_f)$, is defined using a quadrature approximation to the dynamics through:

$$\mathbf{X}(t_f) = \mathbf{X}(t_0) + \frac{t_f - t_0}{2} \sum_{k=1}^N w_k \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, t_k) \quad (4.17)$$

It is important to note that the cost function and constraints that define the NLP are the result of this method, and that the solution of this NLP is an *approximate* solution to the continuous-time optimal control problem.

4.1.2 Sparse Nonlinear Optimizer

The NLP described in Sec. 4.2 was solved with the MATLAB `mex` interface of the NLP solver SNOPT using analytical first-order derivatives for the Jacobian and the gradient of the cost function.

This NLP solver is a general-purpose system for constrained optimization. It minimizes a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. The NLP solver uses a sequential quadratic programming (SQP) algorithm [62] to solve the problem. The search direction is obtained through quadratic programming (QP) sub-problems that minimize a quadratic model of a Lagrangian function subject to linearized constraints. To ensure that convergence is independent of the starting point, an augmented Lagrangian merit function is reduced along each search direction.

Details regarding the MATLAB implementation of the NLP solver may be found in the Users Guide [63]. Interested readers may also refer to Gill et al. [57] for details

on the algorithm itself. In the context of this thesis, it is sufficient to note that this algorithm is suitable for solving NLPs of the form:

$$\text{minimize } \mathbf{x}_v \text{ for } f(\mathbf{x}_v) \text{ subject to } \mathbf{l}_b \leq \begin{pmatrix} \mathbf{x}_v \\ \mathbf{c}(\mathbf{x}_v) \\ \mathbf{A}\mathbf{x}_v \end{pmatrix} \leq \mathbf{u}_b$$

where $\mathbf{x}_v \in \mathbb{R}^n$ is a vector of n -variables, $\mathbf{l}_b \in \mathbb{R}$ and $\mathbf{u}_b \in \mathbb{R}$ are constant lower and upper bounds whose dimensions depend on the number of constraints and variables, $f(\mathbf{x}_v) \in \mathbb{R}$ is a smooth scalar objective function, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a sparse matrix, and $\mathbf{c}(\mathbf{x}_v) \in \mathbb{R}^m$ is a vector of m smooth nonlinear constraint functions $c_i(\mathbf{x}_v) \in \mathbb{R}$ with sparse derivatives. Upper and lower bounds are specified for all variables and constraints.

Ultimately, the ideal output from the NLP solver is that the *optimality conditions are satisfied*; in other words, the returned solution is feasible, the reduced gradient is negligible, the reduced costs are optimal, and the upper-triangular matrix $\mathbf{R}$ is nonsingular, which ensures that the sparse matrix $\mathbf{A}$ is invertible [57]. Note that the matrix $\mathbf{R}$ is obtained from a dense Cholesky factorization and is referred to as the Cholesky factor of the sparse matrix $\mathbf{A}$.

4.2 Simulations

The simulation example presented in this section uses the free-floating robot in Fig. 3.1, whose parameters are defined in Chap. 2. These parameters represent the properties of the experimental platforms used at Carleton University.

4.2.1 Optimal Guidance - Problem Setup

The guidance problem solved in this thesis is the optimal deployment of a robotic manipulator on a free-floating robot. First, the cost function is defined as the square of the difference between the initial and final attitude. This was selected as an objective function instead of minimizing the overall attitude displacement to reduce computation time.

The problem is subject to the dynamic constraints for the free-floating robot used in SPOT - as derived in Chap. 3 - as well as the initial states and their derivatives. Furthermore, the final state derivatives are also constrained such that the platform is at rest by the end of the manoeuvre. The final EE pose is also constrained, as well as the final wrist pose, thereby ensuring that the free-floating robot is in its ideal capture position. Additionally, the controls and states are subject to upper and lower bounds. The constraints on the states were selected based on the physical limits of the manipulator used in SPOT, and the control constraints were selected to ensure that the motion of the robotic manipulator was smooth.

Minimize:

- Cost Function:

$$J(\mathbf{x}, \dot{\mathbf{x}}, \boldsymbol{\tau}) = (q_0(t_f) - q_0(t_0))^2$$

Subject to:

- Dynamic Constraints (Chap. 3)

- Kinematic Boundaries:

$$\mathbf{x}(t_0) = [0 \text{ m}, 0 \text{ m}, 0 \text{ rad}, \pi/2 \text{ rad}, \pi/2 \text{ rad}, 0 \text{ rad}]^T$$

$$\dot{\mathbf{x}}(t_0) = [0 \text{ m/s}, 0 \text{ m/s}, 0 \text{ rad/s}, 0 \text{ rad/s}, 0 \text{ rad/s}, 0 \text{ rad/s}]^T$$

$$x_{ee}(t_f) = -0.08 \text{ m}$$

$$y_{ee}(t_f) = 0.70 \text{ m}$$

$$x_w(t_f) = -0.08 \text{ m}$$

$$\dot{\mathbf{x}}(t_f) = [0 \text{ m/s}, 0 \text{ m/s}, 0 \text{ rad/s}, 0 \text{ rad/s}, 0 \text{ rad/s}, 0 \text{ rad/s}]^T$$

- Final Time:

$$t_f = 30 \text{ s}$$

- Control and Path Constraints:

$$F_x = 0 \text{ N}, F_y = 0 \text{ N}, \tau_0 = 0 \text{ N} \cdot \text{m}$$

$$-0.05 \text{ N} \cdot \text{m} \leq \tau_1 \leq 0.05 \text{ N} \cdot \text{m}, -0.05 \text{ N} \cdot \text{m} \leq \tau_2 \leq 0.05 \text{ N} \cdot \text{m},$$

$$-0.05 \text{ N} \cdot \text{m} \leq \tau_3 \leq 0.05 \text{ N} \cdot \text{m}$$

$$-\pi/2 \text{ rad} \leq q_1 \leq \pi/2 \text{ rad}, -\pi/2 \text{ rad} \leq q_2 \leq \pi/2 \text{ rad}, -\pi/2 \text{ rad} \leq q_3 \leq \pi/2 \text{ rad}$$

Any states and controls not listed in the constraints section are to be considered unconstrained.

4.2.2 Results

The optimal joint trajectories (and resulting attitude disturbance) are shown below:

Figure 4.1: Joint trajectories for joint #1.

Figure 4.2: Joint trajectories for joint #2.

Figure 4.3: Joint trajectories for joint #3.

Figure 4.4: Attitude during the optimized manipulator motion.

These joint trajectories result in the following EE trajectories:

Figure 4.5: EE trajectory - transpose Jacobian.

Figure 4.6: EE trajectory - GPOPS-I.

Figure 4.7: EE trajectory - DIDO.

Figure 4.8: EE trajectory - TOMLAB.

Based solely on the results presented above, it could be concluded that each optimization tool successfully converged to a solution which minimized the attitude

disturbance by the end of the manoeuvre. The fact that each solution is unique also supports the conclusion that this NLP has several local minima, all of which result in reduced disturbances. Note that the optimization was first performed with 30 nodes and no initial guess. The 30 node solution was then used as an initial guess to obtain a 100 node solution. The convergence time for both cases is provided in Table 4.1, but only the final attitude for the 100 node case is presented.

Table 4.1: Convergence time and final attitude for each solution.

Controller	Convergence Time (s) 30 Nodes (No Guess)	Convergence Time (s) 100 Nodes (With Guess)	Final Attitude (rad) 100 Nodes (With Guess)
Jacobian	–	–	0.5490
GPOPS-I	14.856	41.203	0
DIDO	10.497	36.656	0
TOMLAB	5.734	12.974	0

By comparing these results to those obtained by Angel et al. [16], who solved a similar optimization problem using TOMLAB, it can be observed that the final attitude displacement has been greatly reduced. However, it is important to note that the optimal solution obtained by Angel et al. [16] was the result of a different cost function. The advantages present in their work included a smaller maximum transient attitude displacement, as well as generally smoother motion and smaller joint torques. This comes as a direct result of their chosen cost function - which was the torque applied by the manipulator on the body of the spacecraft. In contrast, the solution presented in this thesis provides an optimal trajectory that ensures the spacecraft body returns to its starting attitude. As a consequence, the maximum attitude displacement during the robotic deployment is greater, and the overall trajectories are more complex and require more energy to perform.

4.3 Verification of Optimal Solutions

The results obtained from each tool are first checked using Pontryagin’s minimization principle, then using Bellman’s principle of optimality, and finally by propagating the dynamics model using the control torques.

4.3.1 Pontryagin's Necessary Conditions

To be considered optimal, any candidate solution must satisfy Pontryagin's necessary conditions for optimality. The theory for these conditions will be summarized here, however a more thorough explanation can be found in Ross [55].

First, the Hamiltonian $H(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{u}, t) \in \mathbb{R}$ is defined to be a function of the running cost $F(\mathbf{x}, \mathbf{u}, t) \in \mathbb{R}$, the vector of costates $\boldsymbol{\lambda} \in \mathbb{R}^n$, and the right-hand side of the dynamics, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$ as shown in Eq. (4.18). For a given control function $\mathbf{u}$ to be considered a candidate optimal solution, it is necessary that it globally minimizes the Hamiltonian for all time. Put another way, minimizing the Hamiltonian with respect to the control function (while ensuring that $\boldsymbol{\lambda}$ and $\mathbf{x}$ are constants) will yield a potential solution for the optimal control problem.

$$H(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{u}, t) = F(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \quad (4.18)$$

The path constraints must also be taken into account. In this case, the sole path constraints are the limits on the control and state vectors as defined in the previous section. The Lagrangian of the Hamiltonian can then be defined as:

$$\bar{H}(\boldsymbol{\mu}, \boldsymbol{\lambda}, \mathbf{x}, \mathbf{u}, t) = H(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{u}, t) + \boldsymbol{\mu}^T \mathbf{h}(\mathbf{x}, \mathbf{u}, t) \quad (4.19)$$

where the parameters in $\boldsymbol{\mu} \in \mathbb{R}^m$ are the covectors corresponding to the m -path constraints on the states and controls. The lower Hamiltonian $\mathcal{H}(\boldsymbol{\lambda}, \mathbf{x}, t) \in \mathbb{R}$ is the minimized Hamiltonian with respect to the controls, and can be expressed by:

$$\mathcal{H}(\boldsymbol{\lambda}, \mathbf{x}, t) = \min_{\mathbf{u}} H(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{u}, t) \quad (4.20)$$

Finally, to account for the constraints on the start and end points of the optimal control problem, one must define the endpoint Lagrangian for the problem:

$$\bar{E}(t_0, t_f, \mathbf{x}_0, \mathbf{x}_f, \mathbf{v}) = E_c(t_0, t_f, \mathbf{x}_0, \mathbf{x}_f) + \mathbf{v}^T \mathbf{e}(t_0, t_f, \mathbf{x}_0, \mathbf{x}_f) \quad (4.21)$$

where $E_c(t_0, t_f, \mathbf{x}_0, \mathbf{x}_f) \in \mathbb{R}$ is the endpoint constraint, $\mathbf{v} \in \mathbb{R}^n$ is the n -endpoint covector, and $\mathbf{e}(t_0, t_f, \mathbf{x}_0, \mathbf{x}_f) \in \mathbb{R}^n$ are the n -path constraints.

There are a total of five necessary conditions based on the previously defined scalar equations that must be true for any proposed optimal solution. The first necessary condition for optimality is the *Hamiltonian minimization condition* (HMC), which states the following equality must be true at all points:

$$\frac{\partial \bar{H}}{\partial \mathbf{u}} = \mathbf{0} \quad (4.22)$$

Next, the *adjoint condition* states that the time derivative of the costates can be obtained from the equation below. Note that this condition, for the case presented in this thesis, will not provide any meaningful insight into the validity of any solutions as the partial derivatives prove to be large nonlinear equations:

$$-\dot{\boldsymbol{\lambda}} = \frac{\partial \bar{H}}{\partial \mathbf{x}} \quad (4.23)$$

The next condition is known as the *transversality condition*, and in essence it ensures that the optimal solution has achieved the desired start and end points. This condition can be expressed as:

$$\boldsymbol{\lambda}(t_0) = -\frac{\partial \bar{E}}{\partial \mathbf{x}_0} \quad (4.24)$$

and:

$$\boldsymbol{\lambda}(t_f) = \frac{\partial \bar{E}}{\partial \mathbf{x}_f} \quad (4.25)$$

The *Hamiltonian value condition* then requires that the equality in Eq. (4.26) be true. For a case where the final time is unconstrained, this condition indicates that the final value for the lower Hamiltonian should be zero.

$$\mathcal{H}(t_f) = -\frac{\partial \bar{E}}{\partial t_f} = 0 \quad (4.26)$$

where the states, controls and costates being fed into the Hamiltonian are the optimal values at the final time. Finally, the *Hamiltonian evolution equation* requires that the optimal trajectory satisfies the equality in Eq. (4.27). This equality implies that the lower Hamiltonian should be a constant for all time.

$$\frac{d\mathcal{H}}{dt} = 0 \quad (4.27)$$

Checking each condition does not always provide useful information, however, as certain conditions amount to comparing one unknown to another. Checking a few conditions when possible can be a strong indication that a local minimum has been found. Yet one must be aware that if all checked conditions hold true, it does not *guarantee* that a minimum has been found.

In this thesis, validation will be performed by first checking that the Hamiltonian value condition in Eq. (4.26) is zero (since the endpoint Lagrangian is not directly time-dependent). In addition, the transversality condition will be used to check the start and end points of each solution. Finally, knowing that the time derivative of the lower Hamiltonian must be zero to satisfy the Hamiltonian evolution equation, the lower Hamiltonian can be calculated at each time step and should be a constant for all time. Note that all costates and covectors were obtained through the optimization software.

Hamiltonian Value Condition

Substituting the optimal solution from each tool into Eq. (4.18) yields the following results:

- GPOPS-I:

$$\mathcal{H}(t_f) = -4.1735 \times 10^{-8}$$

- DIDO:

$$\mathcal{H}(t_f) = -5.3123 \times 10^{-9}$$

- TOMLAB:

$$\mathcal{H}(t_f) = -5.3259 \times 10^{-8}$$

The Hamiltonian value at the final node is very close to zero for all tools, and therefore the Hamiltonian value condition holds true.

Transversality Condition

Next, the transversality conditions in Eq. (4.24) and Eq. (4.25) are checked:

- GPOPS-I:

$$\begin{aligned}\boldsymbol{\lambda}(t_0) &= \begin{bmatrix} 0 & 0 & 0 & -0.2384 & 0.0283 & -0.0007 & 0 & 0 & 0 & -0.0196 & -0.0052 & -0.0001 \end{bmatrix}^T \times 10^{-6} \\ -\frac{\partial \bar{E}}{\partial \mathbf{x}_0} &= \begin{bmatrix} 0 & 0 & 0 & -0.2384 & 0.0283 & -0.0007 & 0 & 0 & 0 & -0.0196 & -0.0052 & -0.0001 \end{bmatrix}^T \times 10^{-6}\end{aligned}$$

$$\begin{aligned}\boldsymbol{\lambda}(t_f) &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1293 & 0.1536 & 0.0026 \end{bmatrix}^T \times 10^{-6} \\ \frac{\partial \bar{E}}{\partial \mathbf{x}_f} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1293 & 0.1536 & 0.0026 \end{bmatrix}^T \times 10^{-6}\end{aligned}$$

- DIDO:

$$\begin{aligned}\boldsymbol{\lambda}(t_0) &= \begin{bmatrix} 0 & 0 & 0.0004 & 0.1753 & -0.0279 & -0.0024 & 0 & 0 & 0 & 0.0016 & 0.0001 & 0 \end{bmatrix}^T \times 10^{-7} \\ -\frac{\partial \bar{E}}{\partial \mathbf{x}_0} &= \begin{bmatrix} 0 & 0 & 0.0004 & 0.1753 & -0.0279 & -0.0024 & 0 & 0 & 0 & 0.0016 & 0.0001 & 0 \end{bmatrix}^T \times 10^{-7}\end{aligned}$$

$$\begin{aligned}\boldsymbol{\lambda}(t_f) &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.1020 & -0.0072 \end{bmatrix}^T \times 10^{-8} \\ \frac{\partial \bar{E}}{\partial \mathbf{x}_f} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.1020 & -0.0072 \end{bmatrix}^T \times 10^{-8}\end{aligned}$$

- TOMLAB:

$$\begin{aligned}\boldsymbol{\lambda}(t_0) &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0.0053 & 0 & 0 & 0 & 0.6573 & 0.1914 & 0.0032 \end{bmatrix}^T \times 10^{-7} \\ -\frac{\partial \bar{E}}{\partial \mathbf{x}_0} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0.0053 & 0 & 0 & 0 & 0.6573 & 0.1914 & 0.0032 \end{bmatrix}^T \times 10^{-7}\end{aligned}$$

$$\begin{aligned}\boldsymbol{\lambda}(t_f) &= \begin{bmatrix} -0.2099 & 0 & 0.1319 & 0.0870 & 0.0603 & 0 & 0 & 0 & 0 & -0.0478 & -0.0193 & -0.0004 \end{bmatrix}^T \times 10^{-7} \\ \frac{\partial \bar{E}}{\partial \mathbf{x}_f} &= \begin{bmatrix} -0.2099 & 0 & 0.1033 & 0.0767 & 0.0164 & 0 & 0 & 0 & 0 & -0.0478 & -0.0193 & -0.0004 \end{bmatrix}^T \times 10^{-7}\end{aligned}$$

As discussed above, checking these conditions ensures that the solutions respected the constraints imposed on the initial and terminal states. For the most part, the results from the transversality condition are consistent with the observations made using the Hamiltonian value condition, thus further demonstrating that truly optimal solutions have been obtained. It is interesting to note, however, that the results for TOMLAB show some minor differences. Though it is not possible to draw any significant conclusions based solely on these differences.

Hamiltonian Evolution Equation

Next, the lower Hamiltonian can be evaluated at each time step to ensure that the Hamiltonian evolution equation holds true for all time, which, as previously stated, would mean that the lower Hamiltonian should be some constant for all time. This is indeed the case for each solution, as shown in Fig. 4.9.

Figure 4.9: Hamiltonian evolution equation at each time-step.

4.3.2 Bellman's Principle of Optimality

Next, Bellman's principle of optimality is checked. In brief, this principle states that any optimal control law will - regardless of the initial state and initial decision - converge to the same optimal policy [64]. To check this principle, each solution was run at different starting nodes. For example, for starting node 20, the optimal control problem was set up such that the initial conditions corresponded to the states at node 20, while the final conditions remained the same. If the principle holds true, then the states from nodes 20-100 should be the same as the original solution. For brevity's sake, the validation will be limited to showing the attitude disturbance over time for various starting nodes - shown in Fig. 4.10 to Fig. 4.12.

Figure 4.10: Applying Bellman's principle to the solution from GPOPS-I.

Figure 4.11: Applying Bellman's principle to the solution from DIDO.

Figure 4.12: Applying Bellman's principle to the solution from TOMLAB.

To further emphasize the results from this analysis, the sum of squares for each solution can be calculated using:

$$\mathbf{TSS} = \sum_{i=1}^n (\mathbf{y}_1(i) - \mathbf{y}_2(i))^2 \quad (4.28)$$

where $\mathbf{TSS} \in \mathbb{R}^n$ is the total sum of the squared error between the original solution and the solution obtained from each starting node, $\mathbf{y}_1(i) \in \mathbb{R}^n$ is the i th element of the original solution, and $\mathbf{y}_2(i) \in \mathbb{R}^n$ is the i th element from the solution obtained at a different starting node. Each solution has n elements, where n varies depending on the starting node. The total sum of squares for each starting node is shown in Fig. 4.13.

Figure 4.13: Total sum of squares for each solution at various starting nodes.

The results from this analysis begin to show some inconsistencies within the solutions obtained. Namely, using this principle, it would appear that DIDO provided the best result, followed by GPOPS-I. Meanwhile, the solution obtained from TOMLAB would appear to be the weakest.

4.3.3 Propagation of Optimal Solutions

In addition to validating the optimality of the solutions obtained from the tools, it is equally important to validate their dynamic fidelity using a propagator. To propagate the optimal solution, MATLAB-Simulink was used along with the Runge-Kutta 4(5)th Order Ordinary Differential Equations Solver, ODE45, with relative and absolute tolerances set to 10^{-9} . Using the first data points from each optimal solution, as well as the control torques, each solution was propagated forward to obtain a data set at a higher sampling rate than the original 100 nodes. Similar to what was done for Bellman's principle of optimality, only the attitude disturbance over time will be shown. The results from the propagation are shown from Fig. 4.14 to Fig. 4.16.

Figure 4.14: Propagation of the solution from GPOPS-I.

Figure 4.15: Propagation of the solution from DIDO.

Figure 4.16: Propagation of the solution from TOMLAB.

From these results, it can be noted the states estimated by TOMLAB do not match the propagated results as well as the other two solutions. The decreased dynamic fidelity is consistent with the time required to obtain a solution from each tool. The slowest tools (i.e., GPOPS-I and DIDO) have the highest fidelity, while the fastest tool (TOMLAB) had the lowest fidelity.

Despite the lower fidelity in the solution from TOMLAB, the trajectories from each tool will be used as the desired joint angles for the actuators during the experimental validation campaign, as outlined in the following section. By using results that are known to be truly optimal - in addition to one solution which is known to be inconsistent in terms of its optimality - this thesis will demonstrate the importance of validating the solutions obtained from these so-called “black-box” tools.

4.4 Experiments

To experimentally validate the simulation results, the following experimental test plan was conceived. It is important to first note that experimental validation of the optimal guidance technique is performed in an open-loop fashion. In other words,

the optimal joint positions determined in simulation were supplied directly to the actuators, which house their own (tunable) PID joint position controller (as previously described in Fig. 2.10). A flowchart describing the experiment is shown below:

Figure 4.17: Control logic for experiment.

The experimental validation will be considered a success if the perturbed attitude of the platform is similar to the disturbance observed in simulation. In the case of the solution from TOMLAB, it has already been determined that the dynamic fidelity is poor, and it is therefore expected that the experiment results will be much closer to the results obtained when propagating the dynamics model (as opposed to the attitude estimated by TOMLAB itself). To ensure the repeatability of the results, each experiment was performed 10 times. The mean of the trials is shown in the figures present in the following subsection, along with error bars representing the 99.7% - i.e. 3σ confidence interval for the trials, and the expected attitude obtained from each tool; note that the error bars are calculated using the standard error. The gains for the MX-64 PID controller were chosen to be $K_P = 400$, $K_I = 0$, and $K_D = 200$. For the experiments performed in this chapter, v2.10 of the SPOT 2.0 software was used and data was sampled at 20 Hz. Video of the experiments can be viewed on the SRCL YouTube channel: <https://goo.gl/JwA5Mr>.

4.4.1 Results

The results are presented from Fig. 4.18 to Fig. 4.21:

Figure 4.18: Experimental attitude disturbance - transpose Jacobian.

Figure 4.19: Experimental attitude disturbance - GPOPS-I.

Figure 4.20: Experimental attitude disturbance - DIDO.

Figure 4.21: Experimental attitude disturbance - TOMLAB.

The results obtained compare favourably with the simulated results, with the exception of TOMLAB. However, in the case of TOMLAB, the attitude observed

in experiment is very similar to the results obtained when propagating the solution using ODE45. This further supports the fact that the solutions obtained from any optimization tool must be validated. Without the analysis presented in Sec. 4.3, a user would have no indication that their optimal solution was, in fact, sub-optimal or even unfeasible.

4.4.2 Conclusions and Context

It is important to address the sources of error and assumptions made in this thesis. In no particular order, the following items all contributed to errors observed in the experiments:

- The mass properties for the platform do not account for the change in mass associated with air used for the air bearings and thrusters, which amounts to approximately 0.5 kg when going from a full tank to an empty tank of compressed air. The assumption that the inertia of the platform remains constant is likely one of the main contributing factors to the drifting confidence interval that can be observed in the experimental results. Since the platform inertia is actually changing with each experiment, it is expected that the measured attitude would drift away from the attitude predicted in simulation.
- The robotic manipulator on the free-floating robot is currently self-supported, and while this does provide some benefits in terms of ease-of-use, it also reduces the validity of the planar assumption made in the derivation of the dynamics model. This vertical flex in the linkages means that for the same joint angles, the manipulator will not reach the same EE coordinate expected from simulation.
- The facilities at the SRCL are not kept in a clean room, and dust accumulation is a serious hindrance during experiments. If careful preparations are not made before each trial, large drifts in attitude have been observed. These drifts have been attributed to the air bearings getting stuck momentarily on dust particles. Before each trial, a lengthy cleaning process was followed, and thus this problem should not be a major contributor to the experimental error.

To provide visual context for the experimental results, several photos from the experiments showing the final spacecraft attitude are shown in Fig. 4.22.

(a) Platform position at terminus of GPOPS-I experiment.

(b) Platform position at terminus of DIDO experiment.

(c) Platform position at terminus of TOMLAB experiment.

(d) Platform position at terminus of baseline experiment.

Figure 4.22: Final platform attitudes following deployment manoeuvre.

Based on the data presented in this chapter and the images above, the experimental validation campaign was a success: the manipulator was deployed while ensuring that the attitude returned to its starting position. The sole exception - as expected - was TOMLAB, which ended its manoeuvre with a deviated attitude. In

fact, this puts the platform in an even worse position than what was observed using the baseline controller, since the EE is no longer aligned in the correct direction for capture.

Chapter 5

Compliant Spacecraft Capture

Here, an NDO is proposed as a means of estimating the disturbance forces acting on the EE of a space robot. This force estimate is then used in conjunction with an impedance controller to ensure the stable and compliant capture of a target spacecraft. All relevant code for this chapter can be found in [Appendix D](#).

5.1 Primary Assumptions

The NDO and impedance control law proposed in this thesis require the following assumptions:

1. Both spacecraft are assumed to be close enough for proximity operations. This section of the thesis will focus on the compliant capture and is concerned with neither the deployment nor the approach phase (refer to [Chap. 4](#) for details on the deployment phase).
2. The states of the spacecraft and their derivatives are assumed to be measured quantities. Note, however, that the second derivative of the states are not required for the proposed NDO, but are required for the baseline observer used to compare the performance of the NDO.
3. Contact occurs between the tip of the EE and the target spacecraft. If the contact results in a distributed load, the NDO will not provide an accurate estimate of the disturbance. However, the accuracy of the force estimate - in terms of magnitude - is not important to the performance of the impedance controller, as explained in the body of this chapter.

Of these assumptions, the final one is particularly important since it indicates that any errors in the model will require additional tuning of the impedance controller.

5.2 Baseline Disturbance Observer

In order to provide a baseline against which the NDO can be compared, a simple DO based on the generalized Jacobian concept [47] is used to estimate the contact forces generated from the impact between the EE of the space robot and the target spacecraft [39]. It should be noted here that the author's contributions to the paper published by Angel et al. [39], which is the origin of this baseline observer, focus on software development and validation in simulation and not the development of the concept itself. As such, the baseline observer is not included in the list of contributions in Chap. 1.

The theory for this baseline observer is as follows: when the EE comes into contact with the target at some instantaneous time of contact, impact forces $\mathbf{f}_e$ are generated at the EE. Following the impact, the total joints torques $\boldsymbol{\tau}$, which include the effects of the disturbances, are assumed to be known. Thus, since the control torques $\boldsymbol{\tau}_c$ are a known quantity obtained from the chosen joint controller, it is possible to obtain the disturbance torque vector $\mathbf{d}$ by subtracting the commanded torques from the actual torques. As established in Chap. 3, the contact forces at the EE may be estimated from the generalized Jacobian matrix - described in Eq. (3.24) - without knowledge of the linear position and linear velocity of the spacecraft base.

Ideally, the total torque at the joints can be measured using a current sensor. However, since the current sensor available in SPOT suffers from poor resolution, the total joint torques cannot be measured and is therefore approximated using an inverse dynamics model of the form:

$$\mathbf{H}_{bm}^T \ddot{\mathbf{x}}_b + \mathbf{H}_m \ddot{\boldsymbol{\theta}} + \mathbf{c}_m = \boldsymbol{\tau} \quad (5.1)$$

It is important to note that such an approximation will induce errors due to discrepancies between the model and reality, but it is assumed that the discrepancies are negligible enough that they will not affect the stability of the controller.

5.3 Nonlinear Disturbance Observer

The observer proposed in this section of the thesis is an expansion on the work published by Chen et al. [29]. Thus, it is necessary to first present the theory for the original NDO proposed by Chen et al. [29].

5.3.1 Basic Observer

Deriving a basic disturbance estimator involves changing the estimator output by the error between the estimated and actual disturbance. First, consider that Eq. (3.12) can be expanded and rewritten into:

$$\mathbf{d} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \boldsymbol{\tau}_c \quad (5.2)$$

where $\mathbf{d} \in \mathbb{R}^n$ is the vector of disturbance forces and torques. Then, a simple DO can be proposed:

$$\dot{\hat{\mathbf{d}}} = -\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})\hat{\mathbf{d}} + \mathbf{L}(\mathbf{q}, \dot{\mathbf{q}}) (\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \boldsymbol{\tau}_c) \quad (5.3)$$

where $\hat{\mathbf{d}} \in \mathbb{R}^n$ is the estimated vector of disturbance forces and torques and $\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n \times n}$ is the observer gain matrix. Because there is no information about the derivative of the disturbance prior to t_0 , it is common practice to assume:

$$\dot{\mathbf{d}} = \mathbf{0} \quad (5.4)$$

which would imply that the disturbances vary slowly over time relative to the observer dynamics. This is a common assumption in the field of disturbance estimation [29, 65–70] and is not particularly restrictive. In other words, this assumption is approximately satisfied when the convergence speed is faster than changes in the external forces, indicating that the estimated forces from the NDO will be lagged behind the actual force. However, past work has shown that this kind of observer can still track quickly varying disturbances reasonably well. Next, the observer error $\mathbf{e}_{ob}(t) \in \mathbb{R}^n$ can be defined as:

$$\mathbf{e}_{ob}(t) \triangleq \mathbf{d} - \hat{\mathbf{d}} \quad (5.5)$$

which then results in the following equation when combined with Eq. (5.3) and Eq. (5.4):

$$\dot{\mathbf{e}}_{ob}(t) = \mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})\hat{\mathbf{d}} - \mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{d} \quad (5.6)$$

It has been proven that this type of observer is globally asymptotically stable by choosing:

$$\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}}) = c [\mathbf{I}^{n \times n}] \quad (5.7)$$

where the parameter $c \in \mathbb{R} > 0$ is a constant positive gain value chosen by the control designer.

5.3.2 Modified Observer

The observer presented in the previous subsection makes several assumptions that render it inconvenient for use in a real-world scenario. More specifically, it is unlikely that the second derivative of the states will be readily available for use. This is simply because measurement noise can make it difficult to obtain acceleration data by differentiating the position data twice.

So, the previous observer is modified by Chen et al. [29] such that acceleration data is no longer required. To accomplish this, one must first define an auxiliary variable vector $\mathbf{z} \in \mathbb{R}^n$:

$$\mathbf{z} = \hat{\mathbf{d}} - \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}}) \quad (5.8)$$

where $\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^n$ denotes the designed function vector. To derive the NDO proposed by Chen et al. [29], one must first assume that the matrix $\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})$ can be redefined as the following nonlinear equation:

$$\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \begin{bmatrix} \frac{\partial \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \mathbf{q}} & \frac{\partial \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} \quad (5.9)$$

Combining the above equation with Eq. (5.3) and Eq. (5.8), the following equation can be obtained:

$$\begin{aligned}
 \dot{\mathbf{z}} &= \hat{\dot{\mathbf{d}}} - \frac{d\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{dt} \\
 &= \hat{\dot{\mathbf{d}}} - \left[\frac{\partial \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \mathbf{q}} \quad \frac{\partial \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}} \right] \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} \\
 &= -\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}}) (\mathbf{z} + \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})) + \mathbf{L}(\mathbf{q}, \dot{\mathbf{q}}) (\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \boldsymbol{\tau}_c)
 \end{aligned} \tag{5.10}$$

Rearranging this equation yields the modified observer originally proposed by Chen et al. [29]:

$$\dot{\mathbf{z}} = -\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{z} + \mathbf{L}(\mathbf{q}, \dot{\mathbf{q}}) (\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \boldsymbol{\tau}_c - \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})) \tag{5.11}$$

and:

$$\hat{\mathbf{d}} = \mathbf{z} + \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}}) \tag{5.12}$$

where $\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})$ is given in Eq. (5.9). However, the matrix $\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})$ in its current form still includes the second state derivatives. The elimination of the second state derivatives occurs in the subsequent stability analysis, where the design vector is selected; the final form of the gain matrix $\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})$ for the space robot in Chap. 3 can be found in Eq. (5.17) or alternatively in Eq. (5.19).

5.3.3 Observer Stability

Having presented an overview of the NDO proposed by Chen et al. [29], it is now necessary to show the stability of the observer for the specific case of a space robot. The first step towards accomplishing this goal is to rewrite the observer error equation originally derived in Eq. (5.5):

$$\begin{aligned}
 \dot{\mathbf{e}}_{ob}(t) &= \dot{\mathbf{d}} - \hat{\dot{\mathbf{d}}} \\
 &= -\dot{\mathbf{z}} - \frac{d\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{dt} \\
 &= -\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{e}_{ob}(t)
 \end{aligned} \tag{5.13}$$

From this, it can be concluded that the error between the estimated disturbance and the actual disturbance will asymptotically approach zero if one chooses a value for $\mathbf{L}(\mathbf{q}, \dot{\mathbf{q}})$, and therefore $\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})$ is asymptotically stable. As explained by Chen et al. [29], this is not a simple task for multi-variable systems; however, there exists a fairly systematic approach for determining the nonlinear function $\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})$. The following nonlinear function is proposed for the planar space robot used in SPOT:

$$\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}}) = c \begin{bmatrix} \dot{x}_0 \\ \dot{x}_0 + \dot{y}_0 \\ \dot{x}_0 + \dot{y}_0 + \dot{q}_0 \\ \dot{x}_0 + \dot{y}_0 + \dot{q}_0 + \dot{q}_1 \\ \dot{x}_0 + \dot{y}_0 + \dot{q}_0 + \dot{q}_1 + \dot{q}_2 \\ \dot{x}_0 + \dot{y}_0 + \dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3 \end{bmatrix} \quad (5.14)$$

where if:

$$\det \left(c\mathbf{\Lambda}_u^{-1} + c\mathbf{\Lambda}_u^{-T} - \frac{\partial \bar{\mathbf{M}}(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}} \right) > 0 \quad (5.15)$$

the observer is guaranteed to be globally asymptotically stable. To prove that the chosen nonlinear function ensures a stable observer, one must start with the solution and show that it meets the Lyapunov stability criterion. First, consider the following equation:

$$\begin{aligned} \frac{d\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{dt} &= \begin{bmatrix} \frac{\partial \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \mathbf{q}} & \frac{\partial \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} \\ &= \mathbf{\Lambda}_l \ddot{\mathbf{q}} \end{aligned} \quad (5.16)$$

where $\mathbf{\Lambda}_l \in \mathbb{R}^{6 \times 6}$ represents a lower triangular matrix. The reason for this alteration will become clear; however, in brief, it allows certain terms to be cancelled further in the derivation. Then, combining the result with Eq. (5.9) yields:

$$\mathbf{L}(\mathbf{q}) = c\mathbf{\Lambda}_l \mathbf{M}(\mathbf{q})^{-1} \quad (5.17)$$

Note that the inertia matrix $\mathbf{M}(\mathbf{q})$ is positive definite for all states and is therefore

invertible. The next definition we require is a modified inertia matrix $\bar{\mathbf{M}}(\mathbf{q})$, which can be defined as:

$$\bar{\mathbf{M}}(\mathbf{q}) = \mathbf{\Lambda}_u^{-1} \mathbf{M}(\mathbf{q}) \mathbf{\Lambda}_l^{-1} \quad (5.18)$$

where $\mathbf{\Lambda}_u \in \mathbb{R}^{6 \times 6}$ represents an upper triangular matrix. Thus:

$$\mathbf{L}(\mathbf{q}) = c \bar{\mathbf{M}}(\mathbf{q})^{-1} \mathbf{\Lambda}_u^{-1} \quad (5.19)$$

Because the modified inertia matrix is also positive definite for all states and is therefore invertible, a candidate Lyapunov function for the NDO can be chosen:

$$V(\mathbf{e}_{ob}, \mathbf{q}) = \mathbf{e}_{ob}^T \bar{\mathbf{M}}(\mathbf{q}) \mathbf{e}_{ob} \quad (5.20)$$

Then, by taking the time derivative of the candidate function, one can determine what gain values will guarantee asymptotic stability. For the candidate Lyapunov function, the following identity is known:

$$\begin{aligned} \dot{V}(\mathbf{e}_{ob}, \mathbf{q}) &= \mathbf{e}_{ob}^T \bar{\mathbf{M}}(\mathbf{q}) \dot{\mathbf{e}}_{ob} + \dot{\mathbf{e}}_{ob}^T \bar{\mathbf{M}}(\mathbf{q}) \mathbf{e}_{ob} + \frac{\partial V(\mathbf{e}_{ob}, \mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}} \\ &= -c \mathbf{e}_{ob}^T \bar{\mathbf{M}}(\mathbf{q}) [\bar{\mathbf{M}}(\mathbf{q})^{-1} \mathbf{\Lambda}_u^{-1}] \mathbf{e}_{ob} \dots \\ &\quad -c \mathbf{e}_{ob}^T [\bar{\mathbf{M}}(\mathbf{q})^{-1} \mathbf{\Lambda}_u^{-1}]^T \bar{\mathbf{M}}(\mathbf{q}) \mathbf{e}_{ob} \dots \\ &\quad + \mathbf{e}_{ob}^T \left[\frac{\partial \bar{\mathbf{M}}(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}} \right] \mathbf{e}_{ob} \\ &= -\mathbf{e}_{ob}^T \left[c \mathbf{\Lambda}_u^{-1} + c \mathbf{\Lambda}_u^{-T} - \frac{\partial \bar{\mathbf{M}}(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}} \right] \mathbf{e}_{ob} \end{aligned} \quad (5.21)$$

So, from the equations presented above, one can say that $\dot{V}(\mathbf{e}_{ob}, \mathbf{q}) < 0$, and thus that the observer is stable, if:

$$\det \left(c \mathbf{\Lambda}_u^{-1} + c \mathbf{\Lambda}_u^{-T} - \frac{\partial \bar{\mathbf{M}}(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}} \right) > 0 \quad (5.22)$$

It is clear that for complex dynamics, the stability criterion is a highly complex

nonlinear function. However, because the operational workspace of the space robot is known, one can tune the gain value c to ensure stability prior to any experiments. Nonetheless, the author acknowledges that future work on this observer should involve a deeper analysis on the stability criterion in order to determine exactly which parameters have an effect on the stability and which parameters can be ignored.

5.4 Impedance Controller

Once an estimate of the contact forces is available, it is necessary to propose a suitable controller strategy to perform safe and compliant spacecraft capture. In this thesis, a simple impedance controller - which makes use of the estimated disturbance forces - is proposed for the capture of the target spacecraft. Using an impedance controller gives the control designer the ability to tune the gains such that the desired contact response is achieved. The equation governing the impedance control law is given by Nakanishi and Yoshida [71]:

$$\mathbf{M}_t \delta \ddot{\mathbf{r}}_e + \mathbf{C}_t \delta \dot{\mathbf{r}}_e + \mathbf{K}_t \delta \mathbf{r}_e = \hat{\mathbf{f}}_e \quad (5.23)$$

where $\mathbf{M}_t \in \mathbb{R}^{6 \times 6}$ is the impedance controller characteristic mass, $\mathbf{C}_t \in \mathbb{R}^{6 \times 6}$ is the impedance controller damping, $\mathbf{K}_t \in \mathbb{R}^{6 \times 6}$ is the impedance controller stiffness, $\ddot{\mathbf{r}}_e \in \mathbb{R}^6$ is the compensating acceleration vector, $\dot{\mathbf{r}}_e \in \mathbb{R}^6$ is the compensating velocity vector, and $\mathbf{r}_e \in \mathbb{R}^6$ is the compensating position vector. All impedance controller gain matrices are assumed to be symmetric and positive. The admittance model can then be obtained by rearranging the control law:

$$\delta \ddot{\mathbf{r}}_e = \mathbf{M}_t^{-1} \left(\hat{\mathbf{f}}_e - \mathbf{C}_t \delta \dot{\mathbf{r}}_e - \mathbf{K}_t \delta \mathbf{r}_e \right) \quad (5.24)$$

The compensating position vector can be obtained from:

$$\delta \mathbf{r}_e = \mathbf{r}_d - \mathbf{r}_a \quad (5.25)$$

where $\mathbf{r}_d \in \mathbb{R}^6$ is the desired EE trajectory without compensation, and $\mathbf{r}_a \in \mathbb{R}^6$ is the actual trajectory that the EE must follow when considering the estimated contact force.

5.5 Simulations

In this section, simulations will be performed to demonstrate the capabilities of the proposed NDO-based control law. This will be done by first presenting the contact model used to mimic contact with the target, by performing simulations using the baseline observer, and finally by showing the simulation results obtained using the nonlinear observer. Note that because the mass properties are the same as those found in Chap. 2 they will not be repeated here.

5.5.1 Contact Model

In order to simulate contact with another rigid body, a simple contact model is utilized. More specifically, the Hertz model [72, 73] is used in simulation to apply a “real”, but unknown, force $\mathbf{f}_e$ to the EE. Thus, the NDO will be attempting to calculate this unknown force, which will serve as feedback for the impedance controller. The Hertz model used is shown below:

$$\mathbf{f}_e = \begin{cases} \left(k_c |d_p|^{3/2} + c_c \dot{d}_p \right) \mathbf{n}, & \text{if } d_p \leq 0 \\ \mathbf{0}, & \text{otherwise} \end{cases} \quad (5.26)$$

where $k_c \in \mathbb{R} > 0$ and $c_c \in \mathbb{R} > 0$ are material properties of the contact surface, $\mathbf{n} \in \mathbb{R}^6$ is the normal vector, and $d_p \in \mathbb{R}$ is the distance between the target contact point and the EE as given by the following equation:

$$d_p = \sqrt{(\mathbf{p}_e - \mathbf{p}_c)^T (\mathbf{p}_e - \mathbf{p}_c)} \quad (5.27)$$

where $\mathbf{p}_c \in \mathbb{R}^3$ is the capture point in inertial space. For the simulation presented in this thesis, the target capture point will simply be a point slightly inside the target spacecraft, thus mimicking an overshoot.

5.5.2 Simulation Setup

The simulations presented in this section were performed using MATLAB-Simulink 2017b. Numerical integration was performed using a 4(5)th order Runge-Kutta-Fehlberg (ODE45) integrator with relative and absolute tolerance values of 10^{-9} . White Gaussian noise with a variance of 10^{-7} was added to the second state derivatives to demonstrate how each observer handles noise.

The initial conditions for both simulations are as follows: first, the initial location of the BLACK platform was set to $[x_{0_b}, y_{0_b}, q_{0_b}]^T = [-1.05 \text{ m}, 0 \text{ m}, 0 \text{ rad}]^T$. The capture point - shown in Fig. 5.1 - is initially placed at the surface of BLACK. When contact is achieved, the resulting forces push the BLACK platform away, and thus the contact point moves. This was implemented into simulation in an effort to replicate the experiment conditions as accurately as possible. Unfortunately, there still exist several significant differences between the experiments and the simulation, thereby preventing any direct comparisons as discussed in Sec. 5.6. Control of BLACK in simulation is performed via PD controller with the following gains:

$$\mathbf{K}_P = \begin{bmatrix} 2555.54 & 0 & 0 \\ 0 & 2555.54 & 0 \\ 0 & 0 & 45.65 \end{bmatrix} \quad \mathbf{K}_D = \begin{bmatrix} 118.97 & 0 & 0 \\ 0 & 118.97 & 0 \\ 0 & 0 & 2.16 \end{bmatrix}$$

Figure 5.1: Reference frame and capture point on BLACK platform.

The material coefficients for the BLACK platform were chosen to be $k_c = 20,000$

$\text{N/m}^{3/2}$ and $c_c = 0.1 \text{ Nsm}^{-1}$; the initial position and attitude of the RED platform were chosen to be $[x_0, y_0, q_0]^T = [0 \text{ m}, 0 \text{ m}, \pi/2 \text{ rad}]^T$. The initial joint angles were set to $[q_1, q_2, q_3]^T = [1.053, -1.360, 0.307]^T \text{ rad}$. All initial velocities and rates were set to zero.

Meanwhile, the gains for the transpose Jacobian controller were chosen to be $\mathbf{K}_p = 500\mathbf{I}_{2 \times 2}$, $\mathbf{K}_I = 0\mathbf{I}_{2 \times 2}$, and $\mathbf{K}_d = 500\mathbf{I}_{2 \times 2}$, while the gains for the impedance controller were set to $M_t = 25$, $C_t = 50$, and $K_t = 0.01$. A PD controller with the following gains controlled the base of RED:

$$\mathbf{K}_P = \begin{bmatrix} 9 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0.2 \end{bmatrix} \quad \mathbf{K}_D = \begin{bmatrix} 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}$$

Finally, the NDO gain was set to $c = 5$. The controller logic can be found in Fig. 5.2.

5.5.3 Results - Baseline Observer

Here, the simulation results for the baseline observer are presented. For all simulations, only the x -component of the results will be shown, since contact is achieved with a flat surface. The set point is presented in Fig. 5.3. This figure also contains the changing set point which is the result of the impedance controller, as well as the location of the contact point and the final position of the end-effector at the end of the simulation.

Figure 5.2: Control logic for DO-based impedance controller.

Figure 5.3: Change in reference point due to impedance controller - baseline.

Meanwhile, Fig. 5.4 shows the force estimated by the baseline DO and the actual contact force, while Fig. 5.5 shows the error between the two results.

Figure 5.4: Estimated vs. actual EE force - baseline.

Figure 5.5: Error between estimated and actual EE force - baseline.

The contact force estimated by the DO compares favourably with the actual contact force. This result is expected, given that this baseline observer is model-based and the sole difference between the model in the propagator and the estimator is the added noise. However, as supported by the NDO and experiment results, the baseline observer is particularly vulnerable to noise as it requires knowledge of the second state derivatives.

5.5.4 Results - Nonlinear Observer

Here, the simulation results for the NDO are presented. Recall, the initial parameters and gains have been previously defined in Sec. 5.5.2. The sole difference arises from the chosen observer. Similar to the baseline simulations, the set point is presented in Fig. 5.6. This figure also contains the changing set point which is the result of the impedance controller, as well as the location of the contact point and the final position of the end-effector at the end of the simulation.

Figure 5.6: Change in reference point due to impedance controller - NDO.

Fig. 5.7 shows the comparison between the force estimated by the NDO and the actual contact force.

Figure 5.7: Estimated vs. actual EE force - NDO.

Figure 5.8: Error between estimated and actual EE force - NDO.

Similar to the baseline observer, the estimated contact forces compare favourably to the actual contact forces simulated using the Hertz model. However, there are two key differences that can be noted from these simulation results. First, the estimated force lags behind the observer: this is the result of the estimator requiring some time to converge to the actual contact force. The second observation is that the nonlinear observer is significantly less susceptible to noise, likely because it does not require knowledge of the second derivative. This will be further supported by the experiment results in Sec. 5.6.

5.5.5 Verifying Observer Stability

Finally, it is important to check that the observer was actually stable during the capture scenario. As derived in Sec. 5.3.3, this can be done by checking that the following inequality is met for all states and $t \in [t_0, \infty)$:

$$\det \left(c\Lambda_u^{-1} + c\Lambda_u^{-T} - \frac{\partial \bar{\mathbf{M}}(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}} \right) > 0 \quad (5.28)$$

Thus, Fig. 5.9 shows the stability criterion evaluated over the entire simulation.

From this figure, we can conclude that for the chosen gain the observer is stable, since the values are always positive.

Figure 5.9: Stability criterion evaluated for the duration of the manoeuvre.

5.6 Experiments

To validate the performance of the observer-based controllers, two case studies were performed using v2.11 of SPOT 2.0. The major difference with the newer software lies in the way it handles the transfer of data, but as this version is a work in progress, no additional details will be provided in this thesis. For the first case study, contact was achieved with a rigid wall, using both the baseline DO, as well as with the proposed NDO. For the second case, contact was achieved with a free-flying spacecraft - as was done in the simulations - to demonstrate the applicability of the technique towards the compliant capture of a spacecraft. In both cases, the data was sampled at 100 Hz. Note that all gains in this section were determined through trial and error. Video of the experiments can be viewed on the SRCL YouTube channel: <https://goo.gl/JwA5Mr>.

5.6.1 Case #1 - Contact with a Static Wall

For the first case, the BLACK platform was placed on a block such that it would not move during contact. RED was then placed at some distance away and was allowed to float on the granite table, as shown in Fig 5.10.

Figure 5.10: Experiment setup for case #1.

The EE was then moved close to - but not in contact with - the side of the BLACK platform, as shown in Fig. 5.11.

Figure 5.11: Experiment setup for case #1 - EE position.

In this experiment, the objective was not to experimentally validate the accuracy of the estimated forces, but rather to demonstrate that, with proper tuning, it is

possible to achieve compliant contact regardless of the accuracy of the EE force estimation.

For the nonlinear observer, the gains for the transpose Jacobian controller were chosen to be $\mathbf{K}_p = 800\mathbf{I}_{2 \times 2}$, $\mathbf{K}_I = 25\mathbf{I}_{2 \times 2}$, and $\mathbf{K}_d = 300\mathbf{I}_{2 \times 2}$, while the gains for the impedance controller were set to $M_t = 4$, $C_t = 1$, and $K_t = 0.01$. Finally, the NDO gain was set to $c = 5$.

For the baseline observer, the only differences were the gains for the impedance controller, which were set to $M_t = 5$, $C_t = 1$, and $K_t = 0.01$. Additionally, in order to achieve stable performance, the second state derivatives had to be filtered. The analogue filter design block in Simulink was used, and a 2nd-order lowpass Butterworth filter with a passband edge frequency of 20 rad/s was selected based on trial and error.

Similar to what was done in Chap. 4, the experiment was performed 10 times to showcase repeatability, and the arithmetic mean and 3σ error bars are presented in each figure. First, the force estimated in the x -direction for both observers is shown in the figures below.

Figure 5.12: Contact force in the x-direction - baseline.

Figure 5.13: Contact force in the x-direction - NDO.

Furthermore, the effect of this estimated force can be seen in how the reference point changes over time:

Figure 5.14: Change in reference point due to impedance controller - baseline.

Figure 5.15: Change in reference point due to impedance controller - NDO.

Comparing the results from each observer in case #1 demonstrates that both observers were able to achieve soft contact with the rigid surface. In fact, the largest spike in the force - which occurs during the initial contact - has a similar magnitude across both observers. Additionally, the change in reference point is also similar, though it should be noted that the nonlinear observer has a slightly larger overshoot, which is the result of the slight differences between the forces estimated from each observer from the chosen gains.

5.6.2 Case #2 - Capture of a Controlled Target

For the second experiment, the BLACK platform was placed on the granite table at some distance from the RED platform, and was allowed to float. In this particular case, BLACK was controlled via simple PD controller and was instructed to hold its initial position throughout the experiment. The experimental setup is shown in Fig. 5.16.

Figure 5.16: Experiment setup for case #2.

Since the approach was not the primary concern of the experiments performed in this thesis, prior to each test the EE was aligned with the dock present on BLACK. For the nonlinear observer, only the impedance gains were changed from the values in case #1. The new gains were set to $M_t = 1.5$, $C_t = 1$, and $K_t = 0.01$. Meanwhile, the gains for the baseline observer were not changed from the values selected for case #1. The experimental results for the second case are presented below. First, the estimated contact force obtained using both observers is shown in Fig. 5.17 and Fig. 5.18:

Figure 5.17: Contact force in the x-direction - baseline.

Figure 5.18: Contact force in the x-direction - NDO.

Furthermore, the effect of this estimated force can be seen in how the reference point changes over time:

Figure 5.19: Change in reference point due to impedance controller - baseline.

Figure 5.20: Change in reference point due to impedance controller - NDO.

Once again, compliant capture was achieved using both observers, though the behaviour between each technique is slightly different. It should be noted, however,

that this behaviour is consistent with that observed in case #1. The larger overshoot in case #2 for the NDO comes as a result of the different gains.

5.6.3 Conclusions and Context

From the results presented in this chapter, it can be stated that the nonlinear observer-based controller proposed in this thesis successfully captured the target spacecraft without resulting in any destabilization. Additionally, the performance of the NDO compared favourably with the chosen baseline observer, and the differences observed in simulation - less susceptibility to noise and a lag in observer response - were consistent with what was observed in experiment. The major sources of error for the proposed technique are as follows:

- Excluding joint friction in the dynamics model means that the estimated contact force is inaccurate. As such, the estimated forces should not be used as a substitute for a force-moment sensor when the sole objective is accurate force measurement. However, as mentioned at the beginning of this chapter, impedance controllers do not require an accurate estimate of the forces since the gains will inevitably be tuned until the desired response is achieved, regardless of the actual magnitude of the contact force.
- Similar to Chap. 4, the mass and inertia of the platforms change over the course of repeated experiments due to the air being used. However, as with the previous point, slight inaccuracies in the model do not have any significant effect on the stability of the observer-based controller. A proper analysis of the stability should be a major point when considering improvements to the proposed technique.

To demonstrate the importance of the proposed technique, several photos from the experiments are shown in Fig. 5.21 and Fig. 5.22. These images compare the final position of the platforms by the end of the experiments with and without an impedance controller.

(a) Platform Positions at terminus of experiment with control.

(b) Platform Positions at terminus of experiment without control.

Figure 5.21: Platform behaviour without impedance control - case #1.

(a) Platform positions at terminus of experiment with control.

(b) Platform positions at terminus of experiment without control.

Figure 5.22: Platform behaviour without impedance control - case #2.

From these images, it can be concluded that the use of the impedance controller in both cases allows for stable contact. Removing said impedance control results in the RED platform pushing itself away from the target, which in turn results in the destabilization of both platforms.

Chapter 6

Conclusion

In this chapter, a brief review of the motivation, research methodology, and results are presented. Any publications - including those in progress or under review - are listed, the practicality of the results are discussed, and potential future work is suggested. Finally, this thesis is concluded by emphasizing its contributions to the fields of optimal trajectory planning and spacecraft capture.

6.1 Thesis Motivation

Launching spacecraft into orbit has become increasingly risky, due in large part to the growing volume of debris. Though efforts are continuously being made to regulate the volume being sent into orbit, the point has been reached where it is now necessary to actively reduce the quantity already present. This problem is being addressed using a variety of novel techniques, ranging from capturing spacecraft using tethers and dragging them out of orbit, to docking with damaged spacecraft in an attempt to supply enough fuel such that it can de-orbit itself.

In this thesis, the first two of three phases most commonly associated with debris removal using a robotic manipulator were addressed. The first phase is the deployment phase, where the question posed is whether or not there exists a zero-attitude change deployment manoeuvre. This type of manoeuvre refers to one in which the base of the free-flying robot is guaranteed to return to its initial attitude in the absence of on-orbit disturbances. A zero-attitude manoeuvre allows the spacecraft to save fuel while also reducing any reaction wheel saturation, since no torque is required to stabilize the attitude.

The second phase addressed concerns the capture of a spacecraft - or debris - without causing the destabilization of the target. This is vital when attempting to remove debris, since a non-compliant capture could result in a collision and/or

permanent damage to both objects, thus further compounding the debris problem.

6.2 Significance of Work

The significance of this work is demonstrated through the publications that have resulted from this research.

6.2.1 Published Works

Conference Proceedings

Flores-Abad, A., **Crain, A.**, Nandayapa, M., Garcia-Teran, M., and Ulrich, S., “Disturbance Observer-based Impedance Control for a Compliance Capture of an Object in Space,” *AIAA Guidance, Navigation, and Control Conference*, Kissimmee, Florida, 2018, AIAA Paper 2018-1329. [\[39\]](#) **Best Presentation in Session Award**

Crain, A., and Ulrich, S., “Nonlinear Optimal Trajectory Planning for Free-Floating Space Manipulators using a Gauss Pseudospectral Method,” *AIAA/AAS Astrodynamics Specialist Conference*, Long Beach, California, 2016, AIAA Paper 2016-5272. [\[17\]](#)

6.2.2 Under Review

Conference Proceedings

Crain, A., and Ulrich, S., “Asymptotically-Stable Nonlinear Disturbance Observer for Compliant Free-Flying Robot Capture,” *IEEE Conference on Decision and Control*, Miami Beach, Florida, 2018.

Journal Articles

Crain, A., and Ulrich, S., “Experimental Validation of Pseudospectral-based Optimal Trajectory Planning for a Free-Floating Robot,” *Journal of Guidance, Control, and Dynamics*, 2018.

6.3 A Note on Practicality

Several aspects surrounding the practicality of the techniques in this thesis must be addressed. For the deployment phase, accurate knowledge of the spacecraft properties and the surrounding perturbations must be known in order to generate an off-line solution that can be of any use. If the properties of the model are known, then it is possible to generate an optimal trajectory. Regarding the on-line implementation of the optimal theory used in the optimization tools, powerful hardware would be a necessity. Current experiments in this area are ongoing, with some success. However, for the techniques applied in this thesis, highly specialized equipment would be required. The disadvantage of using off-line tools like the ones found in this thesis is that variable parameters - such as fuel - can be difficult to account for due to the fact that there exist multiple local minima, which would change the controller inputs and thus the fuel required.

Regarding the intermediate phase between the deployment of the manipulator and the capture of the target, there exists the approach sub-phase, which was not addressed in this thesis. This sub-phase can be particularly difficult as it requires that the spacecraft carefully approach the target without overshooting and causing a collision.

With regards to the contact phase presented in this thesis, the author acknowledges that there are several caveats to the technique. First, the work presented here is limited to planar motion, and thus could not be applied to a real spacecraft at this time. More importantly, the impedance gains in the impedance controller are fixed values which must be tuned prior to contact. If trial and error is not a possibility, then simulations are the only means for ensuring that compliant capture occurs. Without a doubt, changes to the algorithm - perhaps an implementation of some form of adaptive control - would be greatly beneficial for real use cases where trial and error is not possible. This phase of the thesis also assumes that a rigid capture is possible, and has not been tested with more complex capture mechanisms.

6.4 Recommendations for Future Work

Areas where this thesis could be elaborated upon are plentiful:

1. Despite the model assumptions in Chap. 3 resulting in negligible performance decreases for the cases in this thesis, there is no guarantee that the result will be the same for a different application. As such, one potential area for improvement would be in expanding the dynamics and kinematics model to include the effects of flexible linkages and friction.
2. The use of off-line optimization, while convenient for complex nonlinear dynamics, may not be practical for all scenarios. Thus, future work could include the development of on-line optimization using PS techniques.
3. Though beneficial for platforms with little fuel or a saturated reaction wheel, the proposed zero-attitude change manoeuvre may not be ideal if the energy usage of the actuators is of concern. In such a case, an alternate manoeuvre that makes use of free-flying space robots may provide an interesting research area.
4. The capture technique proposed in this thesis was only tested using two platforms which were controlled, which in essence should be similar to achieving contact with a soft wall. To expand on this technique, future work could entail attempting to capture a free-floating target, thereby ensuring that the technique would be viable for use in the capture of tumbling debris.
5. In this work, no sensitivity analysis was performed on either the impedance controller or the NDO. Performing a sensitivity study may lead to interesting insights into the effects of each gain, and may also provide a better guide for tuning the gains beyond simple trial and error.
6. In simulation, the contact model used is a poor representation of the actual contact occurring. Thus, the development of a proper contact model would be beneficial. Developing a more dynamically accurate contact model would also give control designers the ability to compare their simulations to experiments

more accurately. As a by-product, this would also give designers the ability to tune the gains before conducting any experiments.

7. Future work on the NDO-based capture technique could also entail the development of an adaptive algorithm, which could retroactively change the impedance gains to ensure the ideal contact response regardless of the properties of the contact surface.
8. One significant area that could prove an interesting area of research would be the development of a “phase #3.” This third phase would ensure that the spacecraft does not become unstable following the capture phase.
9. In this thesis, the stability analysis for the nonlinear observer was broadly derived. To further improve on this concept, future work should expand on the stability criteria such that more specific conclusions can be drawn regarding which parameters have the largest impact on the stability of the observer.

6.5 Primary Contributions

The main contributions of this thesis are divided into the two main topics outlined in Sec. 6.1. First, three commercial optimization tools were used to generate solutions for the zero-attitude manoeuvre guidance problem. The solutions obtained from said tools were validated using well-established optimization criteria and experimentally validated using SPOT. Thus, this thesis provides readers with the general processes required to properly solve optimal problems, in addition to solving the optimal guidance problem of manipulator deployment.

The second contribution of this thesis is a solution to the problem of compliant spacecraft capture via nonlinear disturbance observer-based impedance control. While the observer used in this thesis is not a novel development, proving its stability in the case of a space robot has not been done before, to the best of the author’s knowledge. The observer used in this phase was shown to be analytically stable, and simulations demonstrated the viability of the technique. Experiments using the capture technique were also performed, and it was concluded that the capture technique successfully achieved contact with the target without causing instability.

In addition to the main contributions listed above, this thesis also characterised the properties for the platforms used in SPOT and resulted in the addition of a robotic manipulator to the Spacecraft Robotics and Control Laboratory. The implementation of the robotic manipulator required both software and hardware development. The final major contribution was the design of SPOT 2.0, the latest version of the software used in SPOT to perform experiments.

Bibliography

- [1] Nations, U., “Technical Report on Space Debris,” Tech. rep., United Nations, New York, 1999.
- [2] Kessler, D. J., Johnson, N. L., Liou, J. C., and Matney, M., “The Kessler Syndrome: Implications to Future Space Operations,” *Advances in the Astronautical Sciences*, Vol. 137, No. 8, 2010, pp. 2010.
- [3] Kessler, D. J. and Cour-Palais, B. G., “Collision Frequency of Artificial Satellites: The Creation of a Debris Belt,” *Journal of Geophysical Research*, Vol. 83, No. A6, June 1978, pp. 2637–2646.
- [4] Dubowsky, S. and Torres, M. A., “Path Planning for Space Manipulators to Minimize Spacecraft Attitude Disturbances,” *1991 IEEE International Conference on Robotics and Automation*, IEEE, 1991, pp. 2522–2528.
- [5] Torres, M. A. and Dubowsky, S., “Minimizing Spacecraft Attitude Disturbances in Space Manipulator Systems,” *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 4, 1992, pp. 1010–1017.
- [6] Agrawal, O. P. and Xu, Y., “Global Optimum Path Planning for a Redundant Space Robot,” Tech. rep., Carnegie-Mellon University, 1991.
- [7] Papadopoulos, E. and Abu-Abed, A., “Design and Motion Planning for a Zero-Reaction Manipulator,” *1994 IEEE International Conference on Robotics and Automation*, IEEE, 1994, pp. 1554–1559.
- [8] Lampariello, R., Agrawal, S., and Hirzinger, G., “Optimal Motion Planning for Free-Flying Robots,” *2003 IEEE International Conference on Robotics and Automation*, Vol. 3, IEEE, 2003, pp. 3029–3035.
- [9] Aghili, F., “Optimal Control for Robotic Capturing and Passivation of a Tumbling Satellite with Unknown Dynamics,” *AIAA Guidance, Navigation and Control Conference and Exhibit*, American Institute of Aeronautics and Astronautics, Aug. 2008.
- [10] Yoshida, K., Dimitrov, D., and Nakanishi, H., “On the Capture of Tumbling Satellites by a Space Robot,” *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, pp. 4127–4132.
- [11] Huang, P. and Xu, Y., “PSO-based Time-Optimal Trajectory Planning for Space Robot with Dynamic Constraints,” *2006 IEEE International Conference on Robotics and Biomimetics*, IEEE, 2006, pp. 1402–1407.

- [12] Oki, T., Nakanishi, H., and Yoshida, K., "Time-Optimal Manipulator Control of a Free-Floating Space Robot with Constraint on Reaction Torque," *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008.*, IEEE, 2008, pp. 2828–2833.
- [13] Cascio, J. A., *Optimal Path Planning for Multi-Arm, Multi-Link Robotic Manipulators*, Master's thesis, Naval Postgraduate School, 2008.
- [14] Liu, H.-d., Shi, Y., Liang, B., Xu, W.-f., and Wang, X.-q., "An Optimal Trajectory Planning Method for Stabilization of Coupled Space Robotic System after Capturing," *Procedia Engineering*, Vol. 29, 2012, pp. 3117–3123.
- [15] Xu, S., Wang, H., Zhang, D., and Yang, B., "Adaptive Reactionless Motion Control for Free-Floating Space Manipulators with Uncertain Kinematics and Dynamics," *IFAC Proceedings Volumes*, Vol. 46, No. 20, 2013, pp. 646–653.
- [16] Flores-Abad, A., Wei, Z., Ma, O., and Pham, K., "Optimal Control of Space Robots for Capturing a Tumbling Object with Uncertainties," *Journal of Guidance, Control, and Dynamics*, Vol. 37, No. 6, Nov. 2014, pp. 2014–2017.
- [17] Crain, A. and Ulrich, S., "Nonlinear Optimal Trajectory Planning for Free-Floating Space Manipulators using a Gauss Pseudospectral Method," *AIAA/AAS Astrodynamics Specialist Conference*, American Institute of Aeronautics and Astronautics, Sept. 2016.
- [18] Rybus, T., Seweryn, K., and Sasiadek, J. Z., "Trajectory Optimization of Space Manipulator with Non-zero Angular Momentum During Orbital Capture Maneuver," *AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, Jan. 2016.
- [19] Virgili-Llop, J., Drew, J., Zappulla, R., and Romano, M., "Autonomous Capture of a Resident Space Object by a Spacecraft with a Robotic Manipulator: Analysis, Simulation and Experiments," *AIAA/AAS Astrodynamics Specialist Conference*, American Institute of Aeronautics and Astronautics, Sept. 2016.
- [20] Virgili-Llop, J., Drew, J. V., Zappulla, R., and Romano, M., "Laboratory Experiments of Resident Space Object Capture by a Spacecraft Manipulator System," *Aerospace Science and Technology*, Vol. 71, Dec. 2017, pp. 530–545.
- [21] Zappulla, R., Virgili-Llop, J., Zagaris, C., Park, H., and Romano, M., "Dynamic Air-Bearing Hardware-in-the-Loop Testbed to Experimentally Evaluate Autonomous Spacecraft Proximity Maneuvers," *Journal of Spacecraft and Rockets*, Vol. 54, No. 4, July 2017, pp. 825–839.
- [22] Misra, G. and Bai, X., "Task-Constrained Trajectory Planning of Free-Floating Space-Robotic Systems Using Convex Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 11, Nov. 2017, pp. 2857–2870.

- [23] Misra, G. and Bai, X., “Optimal Path Planning for Free-Flying Space Manipulators via Sequential Convex Programming,” *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 11, Nov. 2017, pp. 3026–3033.
- [24] Yoshikawa, S. and Yamada, K., “Impact Estimation of a Space Robot at Capturing a Target,” *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, Vol. 3, IEEE, 1994, pp. 1570–1577.
- [25] Hacksel, P. J. and Salcudean, S. E., “Estimation of Environment Forces and Rigid-Body Velocities using Observers,” *1994 IEEE International Conference on Robotics and Automation*, IEEE, 1994, pp. 931–936.
- [26] Kim, B. K., Chung, W. K., and Youm, Y., “Robust Learning Control for Robot Manipulators Based on Disturbance Observer,” *Proceedings of the 1996 IEEE IECON 22nd International Conference on Industrial Electronics, Control, and Instrumentation*, Vol. 2, IEEE, 1996, pp. 1276–1282.
- [27] Moosavian, S. A. A. and Papadopoulos, E., “On the Control of Space Free-Flyers Using Multiple Impedance Control,” *1997 IEEE International Conference on Robotics and Automation*, Vol. 1, IEEE, 1997, pp. 853–858.
- [28] Eom, K. S., Suh, I. H., Chung, W. K., and Oh, S.-R., “Disturbance Observer Based Force Control of Robot Manipulator Without Force Sensor,” *1998 IEEE International Conference on Robotics and Automation*, Vol. 4, IEEE, 1998, pp. 3012–3017.
- [29] Chen, W.-H., Ballance, D. J., Gawthrop, P. J., and O’Reilly, J., “A Nonlinear Disturbance Observer for Robotic Manipulators,” *IEEE Transactions on Industrial Electronics*, Vol. 47, No. 4, 2000, pp. 932–938.
- [30] Van Damme, M., Beyl, P., Vanderborght, B., Grosu, V., Van Ham, R., Vanderniepen, I., Matthys, A., and Lefeber, D., “Estimating Robot End-Effector Force from Noisy Actuator Torque Measurements,” *IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 1108–1113.
- [31] Wahrburg, A., Morara, E., Cesari, G., Matthias, B., and Ding, H., “Cartesian Contact Force Estimation for Robotic Manipulators using Kalman Filters and the Generalized Momentum,” *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, 2015, pp. 1230–1235.
- [32] Prvot, A., Gourdeau, R., Aghili, F., and Piedboeuf, J. C., “Impedance Control for Multi-Manipulator System Cooperation,” *55th International Astronautical Congress of the International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law*, American Institute of Aeronautics and Astronautics, Oct. 2004.

- [33] A. Moosavian, S. A., Rastegari, R., and Papadopoulos, E., “Multiple Impedance Control for Space Free-Flying Robots,” *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 5, Sept. 2005, pp. 939–947.
- [34] Abiko, S., Lampariello, R., and Hirzinger, G., “Impedance Control for a Free-Floating Robot in the Grasping of a Tumbling Target with Parameter Uncertainty,” *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, pp. 1020–1025.
- [35] Uyama, N., Nakanishi, H., Nagaoka, K., and Yoshida, K., “Impedance-based Contact Control of a Free-Flying Space Robot with a Compliant Wrist for Non-Cooperative Satellite Capture,” *2012 IEEE/RSJ international conference on Intelligent robots and systems (IROS)*, IEEE, 2012, pp. 4477–4482.
- [36] Roveda, L., Iannacci, N., Vicentini, F., Pedrocchi, N., Braghin, F., and Tosatti, L. M., “Optimal Impedance Force-Tracking Control Design With Impact Formulation for Interaction Tasks,” *IEEE Robotics and Automation Letters*, Vol. 1, No. 1, Jan. 2016, pp. 130–136.
- [37] Murakami, T., Nakamura, R., Yu, F., and Ohnishi, K., “Force Sensorless Impedance Control by Disturbance Observer,” *Conference Record of the Power Conversion Conference*, IEEE, 1993, pp. 352–357.
- [38] Bickel, R. J. and Tomizuka, M., “Disturbance Observer-based Hybrid Impedance Control,” *Proceedings of the 1995 American Control Conference*, Vol. 1, IEEE, 1995, pp. 729–733.
- [39] Flores-Abad, A., Crain, A., Nandayapa, M., Garcia-Teran, M. A., and Ulrich, S., “Disturbance Observer-based Impedance Control for a Compliance Capture of an Object in Space,” *AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, Jan. 2018.
- [40] Hovell, K. and Ulrich, S., “Experimental Validation for Tethered Capture of Spinning Space Debris,” *AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, Jan. 2017.
- [41] Hovell, K. and Ulrich, S., “Postcapture Dynamics and Experimental Validation of Subtethered Space Debris,” *Journal of Guidance, Control, and Dynamics*, Sept. 2017, pp. 1–7.
- [42] Ulrich, S. and Hovell, K., “Iterative Learning Control of Spacecraft Proximity Operations Based on Confidence Level,” *AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, Jan. 2017.

- [43] Boyton, R., “Measuring Weight and all Three Axes of the Center of Gravity of a Rocket Motor Without Having to Re-position the Motor,” *61st Annual Conference of the Society of Allied Weight Engineers*, 2002.
- [44] Habeck, J. and Seiler, P., “Moment of Inertia Estimation Using a Bifilar Pendulum,” Tech. rep., University of Minnesota, 2016.
- [45] Ulrich, S., *Direct Adaptive Control Methodologies for Flexible-Joint Space Manipulators with Uncertainties and Modeling Errors*, Ph.D. thesis, Carleton University, 2012.
- [46] Yoshida, K., Hashizume, K., and Abiko, S., “Zero Reaction Maneuver: Flight Validation with ETS-VII Space Robot and Extension to Kinematically Redundant Arm,” *2001 IEEE International Conference on Robotics and Automation*, Vol. 1, IEEE, 2001, pp. 441–446.
- [47] Umetani, Y. and Yoshida, K., “Resolved Motion Rate Control of Space Manipulators with Generalized Jacobian Matrix,” *IEEE Transactions on Robotics and Automation*, Vol. 5, No. 3, 1989, pp. 303–314.
- [48] Park, J., “The Relationship Between Controlled Joint Torque and End-Effector Force in Underactuated Robotic Systems,” *Robotica*, Vol. 29, No. 04, July 2011, pp. 581–584.
- [49] Benson, D., *A Gauss Pseudospectral Transcription for Optimal Control*, Ph.D. thesis, Massachusetts Institute of Technology, 2005.
- [50] Huntington, G. T., *Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control Problems*, Ph.D. thesis, Massachusetts Institute of Technology, 2007.
- [51] Benson, D. A., Huntington, G. T., Thorvaldsen, T. P., and Rao, A. V., “Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method,” *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, Nov. 2006, pp. 1435–1440.
- [52] Huntington, G. T., Benson, D., and Rao, A. V., “Optimal Configuration of Tetrahedral Spacecraft Formations,” *The Journal of the Astronautical Sciences*, Vol. 55, No. 2, 2007, pp. 141–169.
- [53] Huntington, G. T. and Rao, A. V., “Optimal Reconfiguration of Spacecraft Formations Using the Gauss Pseudospectral Method,” *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 3, May 2008, pp. 689–698.
- [54] Ross, I. M. and Karpenko, M., “A Review of Pseudospectral Optimal Control: From Theory to Flight,” *Annual Reviews in Control*, Vol. 36, No. 2, Dec. 2012, pp. 182–197.

- [55] Ross, I. M., *A Primer on Pontryagin's Principle in Optimal Control*, Collegiate Publishers, 2015.
- [56] Holmström, K., Gran, A. O., and Edvall, M. M., *User's Guide for TOMLAB/SOL*, 2008.
- [57] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, Vol. 47, No. 1, Jan. 2005, pp. 99–131.
- [58] Rutquist, P. E. and Edvall, M. M., *PROPT-Matlab Optimal Control Software*, 2010.
- [59] Wright, K., "Chebyshev Collocation Methods for Ordinary Differential Equations," *The Computer Journal*, Vol. 6, No. 4, 1964, pp. 358–365.
- [60] Gong, Q., Fahroo, F., and Ross, I. M., "Spectral Algorithm for Pseudospectral Methods in Optimal Control," *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 3, May 2008, pp. 460–471.
- [61] Ross, I. M. and Fahroo, F., "Pseudospectral Knotting Methods for Solving Nonsmooth Optimal Control Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 3, May 2004, pp. 397–405.
- [62] Philip, E. and Elizabeth, W., "Sequential Quadratic Programming Methods," Tech. rep., UCSD Department of Mathematics, 2010.
- [63] Gill, P. E., Saunders, M. A., and Murray, W., "User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming," User Manual, University of California, 2008.
- [64] Sniedovich, M., "Dynamic Programming and Principles of Optimality," *Journal of Mathematical Analysis and Applications*, Vol. 65, No. 3, Oct. 1978, pp. 586–606.
- [65] Mita, T., Hirata, M., Murata, K., and Zhang, H., "H-Infinity Control versus Disturbance-Observer-Based Control," *IEEE Transactions on Industrial Electronics*, Vol. 45, No. 3, 1998, pp. 488–495.
- [66] Nikoobin, A. and Haghghi, R., "Lyapunov-Based Nonlinear Disturbance Observer for Serial n-Link Robot Manipulators," *Journal of Intelligent and Robotic Systems*, Vol. 55, No. 2-3, July 2009, pp. 135–153.
- [67] Mohammadi, A., Tavakoli, M., Marquez, H., and Hashemzadeh, F., "Nonlinear Disturbance Observer Design for Robotic Manipulators," *Control Engineering Practice*, Vol. 21, No. 3, March 2013, pp. 253–267.

- [68] Lichiardopol, S., van de Wouw, N., and Nijmeijer, H., “Robust Disturbance Estimation for Human-Robotic Comanipulation,” *International Journal of Robust and Nonlinear Control*, Vol. 24, No. 12, Aug. 2014, pp. 1772–1796.
- [69] Li, Z., Su, C.-Y., Wang, L., Chen, Z., and Chai, T., “Nonlinear Disturbance Observer-Based Control Design for a Robotic Exoskeleton Incorporating Fuzzy Approximation,” *IEEE Transactions on Industrial Electronics*, Vol. 62, No. 9, Sept. 2015, pp. 5763–5775.
- [70] Ueki, S., Mouri, T., and Kawasaki, H., “Nonlinear Disturbance Observer for Object Grasping/Manipulation by Multi-Fingered Robot Hand,” *International Federation of Automatic Control*, 2017.
- [71] Nakanishi, H. and Yoshida, K., “Impedance Control for Free-Flying Space Robots - Basic Equations and Applications,” *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, pp. 3137–3142.
- [72] Dintwa, E., Tijskens, E., and Ramon, H., “On the Accuracy of the Hertz Model to Describe the Normal Contact of Soft Elastic Spheres,” *Granular Matter*, Vol. 10, No. 3, March 2008, pp. 209–221.
- [73] Stronge, W. J., *Impact Mechanics*, Cambridge University Press, Cambridge, 2000, DOI: 10.1017/CBO9780511626432.

Appendix A

Supplementary Documentation - Chapter #2

In this appendix, a detailed description of the SPOT 2.0 software use will be presented. The objective of this chapter is to provide future members of the SRCL with the tools required to perform experiments with ease, as well as ensure the proper documentation of the internal API.

A.1 Getting Started

The SPOT 2.0 software was developed in MATLAB-Simulink 2017b, primarily using the Simulink Support Package for Raspberry Pi Hardware, in conjunction with custom hardware driver blocks. Setting up the software on a new Windows 10 PC is fairly straightforward:

1. Install and open **MATLAB 2017b**.
2. Under the **Home** tab, navigate to **Add-Ons**, and select **Get Hardware Support Packages**.
3. Find **Simulink Support Package for Raspberry Pi Hardware**, click once, then click **Install**.
4. Find **Embedded Coder Support Package for ARM Cortex-A Processors**, click once, then click **Install**.
5. Follow the instructions until you are asked to set up the OS on the Raspberry Pi 3; this step can be skipped for now.
6. Download the folder **SPOT 2.0 Software - v2.10.zip** from SVN - under the **Tools** folder - unzip, and place in whichever directory you want. It is recommended that users do not place the folder too deeply, as Windows 10

may throw out an error if a file name gets too long. Finally, change the folder name to something simple, such as **SRCL**.

7. Open MATLAB 2017b and add the folder **Low-Level Files** to the working directory.

8. Navigate to the Folder **Low-Level Files/newExperiment/** and open the file *newExperiment.m*. In this file, change:

```
cd 'F:\OneDrive\SRCL\Low-Level Files'
```

To whatever the path name is for *Low-Level Files*. Then, change:

```
copyfile('F:\OneDrive\SRCL\Low-Level Files\Experiment Files',...
```

Such that it matches the path name for *Experiment Files*.

9. Ensure that the *Template.slx* file is located under **Low-Level Files**; if the user correctly added this directory to the working directories in MATLAB, the latest SPOT 2.0 template should be visible under **My Templates** when starting a new Simulink diagram.
10. Navigate to the **SPOT Custom Library** folder and open *main_library.slx* and unlock the file (bottom left of the diagram) and in the MATLAB command prompt type `set_param(gcs,'EnableLBRepository','on')`. Save the Simulink diagram, open up the Simulink library browser, and press F5; this will refresh the library, and the custom SPOT Simulink blocks should now be visible.

Once the Windows 10 PC has been set up, the next step is to set up the Raspberry Pi 3 computers; the set up is mostly identical for all 3 computers (RED, BLACK, and GROUNDSTATION):

1. Download the latest release of Raspbian Jessie at <https://www.raspberrypi.org/downloads/raspbian/>. Install this distribution on each Raspberry Pi 3.
2. In the Linux command prompt, type `sudo apt-get update`. Once the updates have been downloaded, install them using the command `sudo apt-get upgrade`.

3. Navigate to the directory `/home/pi/` and create a new folder called **ExperimentSoftware**; inside this folder, create a unique text file for each Pi. Name the text files **WhoAmI.txt**, and inside each file add a unique number to identify the platforms according to the following: **RED** $\rightarrow$ **1** and **BLACK** $\rightarrow$ **2**.

A.2 Quick Start

In this section, the steps required to complete an experiment using the SPOT 2.0 software are outlined. The first subsection contains information on creating a new project folder:

A.2.1 Creating a New Experiment

1. Open **MATLAB 2017b** and navigate to the desired projects folder. Typically, this would be `C:/SRCL/Projects/`. In the command window, type *newExperiment*. This will initialize the set-up process for any new experiments. A text prompt requesting a name for the experiment should appear; give the experiment a name.
2. Click **OK**, and the basic files required for any experiment are copied into a folder with the project name chosen by the user. Navigate to this new folder.
3. Once in this folder, type **simulink** into the command windows. When Simulink has loaded, there should be a section labelled **My Templates**. The most recent version of the SPOT 2.0 template should be available; click once to open a new SPOT 2.0 template diagram.
4. When the diagram has loaded, be sure to save it **with a different name from the project folder created earlier**. If the diagram is not saved under a different name, this can create a conflict when trying to compile the code. A common name is *main_model.slx*, though any name will work.

A.2.2 Initialization

Assuming that a new project folder has been created correctly, and that the user has created a new experiment diagram, the next step is to set up the experiment initializer script.

1. Navigate to the directory created in Section [A.2.1](#). Inside this folder, open the file *Run_Initializer.m*.
2. First, go to line 50 in the code; this is the point where the user may want to implement changes. It should look like:

```
%% Set the base sampling rate.

% This variable will change the frequency at which the template
% runs. The template is tuned for 20 Hz, but with proper
% tuning users may be able to take it higher or lower. Note
% that the PhaseSpace data will always be read at 15 Hz unless
% the source file is changed.

baseRate                        = 0.05;      % 20 Hz

%% Set the duration of each major phase in the experiment,
%   in seconds:

Phase0_Duration                = 10;         % [s]
Phase1_Duration                = 5;          % [s]
Phase2_Duration                = 20;         % [s]
Phase3_Duration                = 17;         % [s]
Phase4_Duration                = 20;         % [s]
Phase5_Duration                = 5;          % [s]
```

3. The default sample rate is 20 Hz, however the user can change this value to meet his/her needs by editing the **baseRate** constant. It is recommended that the user does not set the sample rate lower than 20 Hz if they wish to use the

default PD controller (due to chattering); additionally, the upper limit of the sample rate is not currently known, though the template has been tested up to 100 Hz.

4. Next, the user must choose the duration for each phase of the experiment. This can be accomplished by changing the `Phase#_Duration` constants, where the `#` sign represents the phase number. The phases are segmented as follows:

- Phase `#0` - Length of time the program will wait for good data.
- Phase `#1` - Length of time the program will float before any kind of motion.
- Phase `#2` - Duration of the “move to initial location” phase.
- Phase `#3` - Duration of the experiment. For a multiphase experiment, the duration is the sum of all sub-phases. Duration of the sub-phases is defined from lines 66-74.
- Phase `#4` - Duration of the “move to home location” phase.
- Phase `#5` - Length of time over which the platform will de-float.

5. Typically, the user should only edit the duration of phases `#2` through `#4`, inclusively. The next parameters that users should edit are the drop, initial, and home coordinates. These constants are located from lines 119-126 in the code and should look like:

```
drop_states_RED      = [ 2.5; 1.75; 0];           % [m; m; rad]
drop_states_BLACK    = [ 1; 0.75; 0];           % [m; m; rad]

init_states_RED      = [ xLength/2; yLength/2; 0]; % [m; m; rad]
init_states_BLACK    = [ xLength/2; yLength/2; 0]; % [m; m; rad]

home_states_RED      = [ 2.5; 1.75; 0];           % [m; m; rad]
home_states_BLACK    = [ 1; 0.75; 0];           % [m; m; rad]
```

A.2.3 Preparing the Diagram

Once the initializer script has been prepared, it is time to set up the Simulink diagram for an experiment. Since each experiment is unique, only a broad overview of the diagram will be presented. Assuming the user correctly downloaded the SPOT 2.0 custom template, no settings should have to be changed.

1. First, open the Simulink diagram created in Section [A.2.1](#). The diagram is broken down into six distinct sections:

- **Experiment Loop - v2.10:** For the typical user, this section is the only section of interest as it contains the code dictating what the platforms will do during the experiment.
- **Memory Store Blocks:** This section contains Memory Storage blocks, which are required for any data that must pass through an IF statement (such as the states provided by the PhaseSpace cameras). Users should never edit any existing blocks, but can add blocks if new data is added to the diagram (for example, adding another linkage to the robotic arm).
- **Plant Dynamics (For Simulation Only):** This section contains the dynamics model for the platforms. Since the mass properties are set in the initialization script, users should only have to edit this section if the dynamics change (for example, if the user wished to simulate a free-flying robot).
- **External Hardware (For Experiment Only):** Any external hardware is read into the diagram in this section. As of v2.10, the only hardware being read in is the states of each platform (using the PhaseSpace) and the joint angles for the robotic manipulator (using encoders). Details on adding new hardware will be covered further on in this section.
- **Log Data:** Here, all of the simulation data is logged. New data can be logged by simply adding another **To Workspace** block to the **Store Sim Data** If-Action subsystem. Most experimental data is logged in the *runExperiment.slx* diagram; however, if the user wishes to log some signal

generated specifically in the template, they must send the data to the ground-station using a **UDP Send** block.

- **Hardware Control:** This section contains the code which converts the commanded forces and torques into thruster firings (or reaction wheel commands). Users should never have to edit this section.
2. Generally, users only need to edit the **Phase #3** block in the **Experiment Loop** section. Inside this block, the diagram is further subdivided according to the **RED** and **BLACK** platforms. For any control law for the **BLACK** platform, the code should be placed inside the **Execute BLACK Experiment** block. This is similar for the **RED** platform.

If data specific to the template must be logged, additional steps are required. First, open the *runExperiment.slx* diagram which should be present in the user generated project folder. From there, follow these steps:

1. Navigate to the **Log Data** section of the template, and open the **Store Exp Data** subsystem.
2. Using the memory store blocks, send the signal that must be logged into the **Mux** block. Any signal going into this muxer is then sent to the **UDP Send** block. Users should not have to edit the **UDP** block on the template side, all that is required is to ensure the relevant data is being sent into the block.
3. Navigate to the **Log Data** section of the **runExperiment** diagram. From there, open the subsystem labelled **Log Data from Experiment Diagram**. In this subsystem, edit the **Maximum Length for Message** parameter of the existing **UDP Receive** block. The maximum length of the message should correspond to the length of the data vector being sent from the template.
4. Edit the demuxer so that all of the signals are being extracted individually, and then simply use a **To Workspace** block to store the data.

A.2.4 Performing the Experiment

Once the Simulink diagram and initialization script have been prepared, the experiment can be performed. To begin, run the `Run_Initializer.m` script; this will initialize the constant parameters for the experiment and it will also start the GUI. The GUI is divided into nine sections as shown in Fig. A.1:

Figure A.1: Graphical user interface used in the SPOT 2.0 software.

Generally, when performing an experiment, the user should follow the following steps:

1. Step #1 - Select Diagram: In this section of the GUI, the user selects their experiment diagram. Note, **the experiment digram is the one created for the experiment, not the runExperiment.slx diagram**. Using the drop down menu, choose the correct diagram and click **Open Diagram**.
2. Step #2 - Select Platforms & Confirm: Here, the user must select which platforms will participate in the experiment. The user must select all platforms

that apply; i.e. if the experiment involves RED and BLACK, then the user must check the RED and BLACK option. Once the correct platforms have been selected, click **Confirm**.

3. Step #2.5 - Simulations: At this point, simulations should be performed to ensure that the diagram developed in Section [A.2.3](#) behaves as expected. To perform a simulation, simply select an animation speed from the **Choose Animation Speed & Run Simulation** box in the bottom right of the GUI, then click **Run Simulation**. Once the simulation is complete, click **Play Animation** to view the results in the GUI; the data for the simulation is also available in the workspace if the user wants to plot the results. To stop the animation and restart it, simply click **Stop Animation** followed by **Play Animation**.
4. Step #3 - Deploy Diagram: Once the user is satisfied with the simulation results, the next step is to compile the experiment diagram and deploy the code to all relevant platforms. This is accomplished by simply clicking on the **Deploy to RED** and the **Deploy to BLACK** buttons, depending on which platforms are included in the experiment.
5. Step #4 - Float Platforms: Once the build is successful (a message should appear in the command window), it is time to begin the experiment. First, float the platforms by using the **Float ON** button. Prior to placing the platforms on the table, ensure that the pucks have been cleaned by applying a small amount of isopropyl alcohol to a cleaning sheet and gently wiping the pucks. Once the pucks are cleaned, the platforms can be placed on the granite table at their approximate drop coordinates. Once in position, click the **Float OFF** buttons to stop the platforms from floating off the table.
6. Step #5 - Begin Experiment: To start an experiment, simply click **START**; if real-time visualization of the platforms is desired, simply check the **Real-Time Visualization** option. Pay close attention to the experiment to ensure no damage occurs to the platforms. Once the platform has defloated the user must manually click the **STOP/RESET** button to turn off the PhaseSpace cameras and stop the experiment.

7. After the experiment, the entire workspace should be saved by clicking the **Save Workspace** button. This saves the workspace in the **Saved Data** folder.
8. To remove the platforms from the table, click the **Float ON** button and carefully lift the platform from the table; place the platforms on their corresponding pieces of wood, being sure to carefully avoid having the pucks touch any surfaces.

A.3 Device Driver Blocks

To add new hardware, users must create a custom driver block. The process for doing so may seem intimidating at first, but is actually fairly straightforward. The general steps required to create a driver block can be found at <https://goo.gl/LhpQWR>; instead of providing general steps, this section will provide the code used in the creation of the device driver blocks for the Dynamixel actuators and the PhaseSpace cameras. Between the two examples provided here, and the guide provided by MATLAB, users should have a good starting point when developing new drivers.

A.3.1 Dynamixel Driver Blocks

In this subsection, the code for the Dynamixel position control and current control is presented. To properly compile the code presented here, users must also include the sources and header files provided by Dynamixel, which can be downloaded from http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/.

MATLAB System Function (Initialize Position Command)

```
classdef InitializeForPosition < matlab.System ...
    & coder.ExternalDependency ...
    & matlab.system.mixin.Propagates ...
    & matlab.system.mixin.CustomIcon
%
% System object template for a sink block.
%
% This template includes most, but not all, possible properties,
% attributes, and methods that you can implement for a System object in
% Simulink.
%
% NOTE: When renaming the class name Sink, the file name and
% constructor name must be updated to use the class name.
%

% Copyright 2016 The MathWorks, Inc.
```

```

%#codegen
%#ok<*EMCA>

```

```

properties

```

```

    P_GAIN          = 400;
    I_GAIN          = 0;
    D_GAIN          = 200;
    MAX_POSITION    = 3072;
    MIN_POSITION    = 1024;
    VELOCITY_LIMIT  = 0;

```

```

end

```

```

properties (Nontunable)

```

```

end

```

```

properties (Access = private)

```

```

end

```

```

methods

```

```

% Constructor
function obj = InitializeForPosition(varargin)
    % Support name-value pair arguments when constructing the object.
    setProperties(obj,nargin,varargin{:});
end
end

methods (Access=protected)
    function setupImpl(obj)
        if isempty(coder.target)
            % Place simulation setup code here
        else
            % Call C-function implementing device initialization
            coder.cinclude('dynamixel_sdk.h');
            coder.cinclude('dynamixel_functions.h');
            coder.ceval('initialize_dynamixel_position_control',obj.P_GAIN, obj.I_GAIN, ...
                obj.D_GAIN, obj.MAX_POSITION, obj.MIN_POSITION, obj.VELOCITY_LIMIT);
        end
    end
end

function stepImpl(~,~)
    if isempty(coder.target)
        % Place simulation output code here
    end
end

```

```

        else
            % Call C-function implementing device output
        end
    end
end

function releaseImpl(obj) %#ok<MANU>
    if isempty(coder.target)
        % Place simulation termination code here
    else
        % Call C-function implementing device termination
        % Call C-function implementing device initialization
        coder.cinclude('dynamixel_sdk.h');
        coder.cinclude('dynamixel_functions.h');
        coder.ceval('terminate_dynamixel');
    end
end
end

end

methods (Access=protected)
    %% Define input properties
    function num = getNumInputsImpl(~)
        num = 0;
    end
end

```



```

function num = getNumOutputsImpl(~)
    num = 0;
end

function flag = isInputSizeLockedImpl(~,~)
    flag = true;
end

function varargout = isInputFixedSizeImpl(~,~)
    varargout{1} = true;
end

function flag = isInputComplexityLockedImpl(~,~)
    flag = true;
end

function validateInputsImpl(~, ~)
    if isempty(coder.target)
        % Run input validation only in Simulation
        % validateattributes(u,{'double'},{'scalar'},'', 'u');
    end
end
end

```

```

function icon = getIconImpl(~)
    % Define a string as the icon for the System block in Simulink.
    icon = 'InitializeForPosition';
end

end

methods (Static, Access=protected)
    function simMode = getSimulateUsingImpl(~)
        simMode = 'Interpreted execution';
    end

    function isVisible = showSimulateUsingImpl
        isVisible = false;
    end

    function header = getHeaderImpl
        header = matlab.system.display.Header('InitializeForPosition','Title',...
            'Dynamixel Actuator - Initialize Position Control','Text',...
            ['This simulink block initializes the actuators for position control. '...
            'This block must be placed ONCE in the diagram, at the top level.' newline]);
    end
end

```

```
end
```

```
methods (Static)
```

```
function name = getDescriptiveName()
```

```
    name = 'InitializeForPosition';
```

```
end
```

```
function b = isSupportedContext(context)
```

```
    b = context.isCodeGenTarget('rtw');
```

```
end
```

```
function updateBuildInfo(buildInfo, context)
```

```
    if context.isCodeGenTarget('rtw')
```

```
        % Update buildInfo
```

```
        srcDir = fullfile(fileparts(mfilename('fullpath')),'src');
```

```
        includeDir = fullfile(fileparts(mfilename('fullpath')),'include');
```

```
        addIncludePaths(buildInfo,includeDir);
```

```
        % Add all SOURCE files for compiling robotis software
```

```
        addSourceFiles(buildInfo,'protocol2_packet_handler.cpp', srcDir);
```

```
        addSourceFiles(buildInfo,'protocol1_packet_handler.cpp', srcDir);
```

```
        addSourceFiles(buildInfo,'port_handler_windows.cpp', srcDir);
```

```

addSourceFiles(buildInfo,'port_handler_mac.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_linux.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_arduino.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler.cpp', srcDir);
addSourceFiles(buildInfo,'packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_read.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_read.cpp', srcDir);
addSourceFiles(buildInfo,'dynamixel_functions.cpp', srcDir);

% Add all INCLUDE files for compiling robotis software
addIncludeFiles(buildInfo,'protocol2_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'protocol1_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_windows.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_mac.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_linux.h',includeDir);
addIncludeFiles(buildInfo,'port_handler.h',includeDir);
addIncludeFiles(buildInfo,'packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'group_sync_write.h',includeDir);
addIncludeFiles(buildInfo,'group_sync_read.h',includeDir);
addIncludeFiles(buildInfo,'group_bulk_write.h',includeDir);
addIncludeFiles(buildInfo,'group_bulk_read.h',includeDir);

```

```

        addIncludeFiles(buildInfo,'dynamixel_sdk.h',includeDir);
        addIncludeFiles(buildInfo,'port_handler_arduino.h',includeDir);
        addIncludeFiles(buildInfo,'dynamixel_functions.h',includeDir)

        %addLinkFlags(buildInfo,{'-lSource'});
        %addLinkObjects(buildInfo,'sourcelib.a',srcDir);
        %addCompileFlags(buildInfo,{'-D_DEBUG=1'});
        %addDefines(buildInfo,'MY_DEFINE_1')
    end
end
end
end
end

```

MATLAB System Function (Command Position)

```

classdef MoveArm_Position < matlab.System ...
    & coder.ExternalDependency ...
    & matlab.system.mixin.Propagates ...
    & matlab.system.mixin.CustomIcon
%
% System object template for a sink block.
%
% This template includes most, but not all, possible properties,

```

```

% attributes, and methods that you can implement for a System object in
% Simulink.
%
% NOTE: When renaming the class name Sink, the file name and
% constructor name must be updated to use the class name.
%

% Copyright 2016 The MathWorks, Inc.
%#codegen
%#ok<*EMCA>

properties

end

properties (Nontunable)

end

properties (Access = private)

end

```

```

methods
    % Constructor
    function obj = MoveArm_Position(varargin)
        % Support name-value pair arguments when constructing the object.
        setProperties(obj,nargin,varargin{:});
    end
end

methods (Access=protected)
    function setupImpl(~)
        if isempty(coder.target)
            % Place simulation setup code here
        else
            % Call C-function implementing device initialization
        end
    end
end

function stepImpl(~,u1,u2,u3)
    if isempty(coder.target)
        % Place simulation output code here
    else
        % Call C-function implementing device output
        %coder.ceval('sink_output',u);
    end
end

```

```

        coder.cinclude('dynamixel_sdk.h');
        coder.cinclude('dynamixel_functions.h');
        coder.ceval('command_dynamixel_position', u1, u2, u3);
    end
end

function releaseImpl(obj) %#ok<MANU>
    if isempty(coder.target)
        % Place simulation termination code here
    else
        % Call C-function implementing device termination
        %coder.ceval('sink_terminate');
    end
end

end

methods (Access=protected)
    %% Define input properties
    function num = getNumInputsImpl(~)
        num = 3;
    end

    function num = getNumOutputsImpl(~)

```



```

        num = 0;
    end

function flag = isInputSizeLockedImpl(~,~)
    flag = true;
end

function varargout = isInputFixedSizeImpl(~,~)
    varargout{1} = true;
end

function flag = isInputComplexityLockedImpl(~,~)
    flag = true;
end

function validateInputsImpl(~, u1, u2, u3)
    if isempty(coder.target)
        % Run input validation only in Simulation
        validateattributes(u1,{'double'},{'scalar'},'', 'u1');
        validateattributes(u2,{'double'},{'scalar'},'', 'u2');
        validateattributes(u3,{'double'},{'scalar'},'', 'u3');
    end
end
end

```

```

function icon = getIconImpl(~)
    % Define a string as the icon for the System block in Simulink.
    icon = 'MoveArm_Position';
end

end

methods (Static, Access=protected)
    function simMode = getSimulateUsingImpl(~)
        simMode = 'Interpreted execution';
    end

    function isVisible = showSimulateUsingImpl
        isVisible = false;
    end

    function header = getHeaderImpl
        header = matlab.system.display.Header('MoveArm_Position','Title',...
            'Dynamixel Actuator - Position Control','Text',...
            ['This simulink block sends the input to the Dynamixel actuators. '...
            'The input must be the desired joint angle in radians.'...
            ' The inputs are joint #1, #2, and #3 going from top to bottom.' newline]);
    end
end

```

```

        end

    end

methods (Static)
    function name = getDescriptiveName()
        name = 'MoveArm_Position';
    end

    function b = isSupportedContext(context)
        b = context.isCodeGenTarget('rtw');
    end

    function updateBuildInfo(buildInfo, context)
        if context.isCodeGenTarget('rtw')
            % Update buildInfo
            srcDir = fullfile(fileparts(mfilename('fullpath')),'src');
            includeDir = fullfile(fileparts(mfilename('fullpath')),'include');
            addIncludePaths(buildInfo,includeDir);

            % Add all SOURCE files for compiling robotis software
            addSourceFiles(buildInfo,'protocol2_packet_handler.cpp', srcDir);
            addSourceFiles(buildInfo,'protocol1_packet_handler.cpp', srcDir);
        end
    end
end

```

```

addSourceFiles(buildInfo,'port_handler_windows.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_mac.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_linux.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_arduino.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler.cpp', srcDir);
addSourceFiles(buildInfo,'packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_read.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_read.cpp', srcDir);
addSourceFiles(buildInfo,'dynamixel_functions.cpp', srcDir);

% Add all INCLUDE files for compiling robotis software
addIncludeFiles(buildInfo,'protocol2_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'protocol1_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_windows.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_mac.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_linux.h',includeDir);
addIncludeFiles(buildInfo,'port_handler.h',includeDir);
addIncludeFiles(buildInfo,'packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'group_sync_write.h',includeDir);
addIncludeFiles(buildInfo,'group_sync_read.h',includeDir);
addIncludeFiles(buildInfo,'group_bulk_write.h',includeDir);

```

```

        addIncludeFiles(buildInfo,'group_bulk_read.h',includeDir);
        addIncludeFiles(buildInfo,'dynamixel_sdk.h',includeDir);
        addIncludeFiles(buildInfo,'port_handler_arduino.h',includeDir);
        addIncludeFiles(buildInfo,'dynamixel_functions.h',includeDir)

        %addLinkFlags(buildInfo,{'-lSource'});
        %addLinkObjects(buildInfo,'sourcelib.a',srcDir);
        %addCompileFlags(buildInfo,{'-D_DEBUG=1'});
        %addDefines(buildInfo,'MY_DEFINE_1')
    end
end
end
end
end

```

MATLAB System Function (Initialize Current Command)

```

classdef InitializeForTorque < matlab.System ...
    & coder.ExternalDependency ...
    & matlab.system.mixin.Propagates ...
    & matlab.system.mixin.CustomIcon
%
% System object template for a sink block.

```

```

%
% This template includes most, but not all, possible properties,
% attributes, and methods that you can implement for a System object in
% Simulink.
%
% NOTE: When renaming the class name Sink, the file name and
% constructor name must be updated to use the class name.
%

% Copyright 2016 The MathWorks, Inc.
%#codegen
%#ok<*EMCA>

properties

end

properties (Nontunable)
    CURRENT_LIMIT = 850;
end

properties (Access = private)

```

```

end

methods
    % Constructor
    function obj = InitializeForTorque(varargin)
        % Support name-value pair arguments when constructing the object.
        setProperties(obj,nargin,varargin{:});
    end
end

methods (Access=protected)
    function setupImpl(obj)
        if isempty(coder.target)
            % Place simulation setup code here
        else
            % Call C-function implementing device initialization
            coder.cinclude('dynamixel_sdk.h');
            coder.cinclude('dynamixel_functions.h');
            coder.ceval('initialize_dynamixel_current_control',obj.CURRENT_LIMIT);
        end
    end
end

```

```

function stepImpl(~,~)
    if isempty(coder.target)
        % Place simulation output code here
    else
        % Call C-function implementing device output
    end
end

function releaseImpl(obj) %#ok<MANU>
    if isempty(coder.target)
        % Place simulation termination code here
    else
        % Call C-function implementing device termination
        % Call C-function implementing device initialization
        coder.cinclude('dynamixel_sdk.h');
        coder.cinclude('dynamixel_functions.h');
        coder.ceval('terminate_dynamixel');
    end
end

end

methods (Access=protected)
    %% Define input properties

```



```

function num = getNumInputsImpl(~)
    num = 0;
end

function num = getNumOutputsImpl(~)
    num = 0;
end

function flag = isInputSizeLockedImpl(~,~)
    flag = true;
end

function varargout = isInputFixedSizeImpl(~,~)
    varargout{1} = true;
end

function flag = isInputComplexityLockedImpl(~,~)
    flag = true;
end

function validateInputsImpl(~, ~)
    if isempty(coder.target)
        % Run input validation only in Simulation
    end
end

```

```

        % validateattributes(u,{'double'},{'scalar'},'', 'u');
    end
end

function icon = getIconImpl(~)
    % Define a string as the icon for the System block in Simulink.
    icon = 'InitializeForTorque';
end

end

methods (Static, Access=protected)
    function simMode = getSimulateUsingImpl(~)
        simMode = 'Interpreted execution';
    end

    function isVisible = showSimulateUsingImpl
        isVisible = false;
    end

    function header = getHeaderImpl
        header = matlab.system.display.Header('InitializeForTorque','Title',...
            'Dynamixel Actuator - Initialize Torque Control','Text',...

```

```

        ['This simulink block initializes the actuators for torque control. '...
        'This block must be placed ONCE in the diagram, at the top level.' newline]);
    end

end

methods (Static)
    function name = getDescriptiveName()
        name = 'InitializeForTorque';
    end

    function b = isSupportedContext(context)
        b = context.isCodeGenTarget('rtw');
    end

    function updateBuildInfo(buildInfo, context)
        if context.isCodeGenTarget('rtw')
            % Update buildInfo
            srcDir = fullfile(fileparts(mfilename('fullpath')),'src');
            includeDir = fullfile(fileparts(mfilename('fullpath')),'include');
            addIncludePaths(buildInfo,includeDir);

            % Add all SOURCE files for compiling robotis software

```

```

addSourceFiles(buildInfo,'protocol2_packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'protocol1_packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_windows.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_mac.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_linux.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_arduino.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler.cpp', srcDir);
addSourceFiles(buildInfo,'packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_read.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_read.cpp', srcDir);
addSourceFiles(buildInfo,'dynamixel_functions.cpp', srcDir);

% Add all INCLUDE files for compiling robotis software
addIncludeFiles(buildInfo,'protocol2_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'protocol1_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_windows.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_mac.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_linux.h',includeDir);
addIncludeFiles(buildInfo,'port_handler.h',includeDir);
addIncludeFiles(buildInfo,'packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'group_sync_write.h',includeDir);

```

```

        addIncludeFiles(buildInfo,'group_sync_read.h',includeDir);
        addIncludeFiles(buildInfo,'group_bulk_write.h',includeDir);
        addIncludeFiles(buildInfo,'group_bulk_read.h',includeDir);
        addIncludeFiles(buildInfo,'dynamixel_sdk.h',includeDir);
        addIncludeFiles(buildInfo,'port_handler_arduino.h',includeDir);
        addIncludeFiles(buildInfo,'dynamixel_functions.h',includeDir)

        %addLinkFlags(buildInfo,{'-lSource'});
        %addLinkObjects(buildInfo,'sourcelib.a',srcDir);
        %addCompileFlags(buildInfo,{'-D_DEBUG=1'});
        %addDefines(buildInfo,'MY_DEFINE_1')
    end
end
end
end
end

```

MATLAB System Function (Command Current)

```

classdef MoveArm_Torque < matlab.System ...
    & coder.ExternalDependency ...
    & matlab.system.mixin.Propagates ...
    & matlab.system.mixin.CustomIcon

```

```

%
% System object template for a sink block.
%
% This template includes most, but not all, possible properties,
% attributes, and methods that you can implement for a System object in
% Simulink.
%
% NOTE: When renaming the class name Sink, the file name and
% constructor name must be updated to use the class name.
%

% Copyright 2016 The MathWorks, Inc.
%#codegen
%#ok<*EMCA>

properties

end

properties (Nontunable)

end

```

```

properties (Access = private)

end

methods

    % Constructor
    function obj = MoveArm_Torque(varargin)
        % Support name-value pair arguments when constructing the object.
        setProperties(obj,nargin,varargin{:});
    end
end

methods (Access=protected)
    function setupImpl(~)
        if isempty(coder.target)
            % Place simulation setup code here
        else
            % Call C-function implementing device initialization
        end
    end
end

function stepImpl(~,u1,u2,u3)
    if isempty(coder.target)

```

```

        % Place simulation output code here
    else
        % Call C-function implementing device output
        %coder.ceval('sink_output',u);
        coder.cinclude('dynamixel_sdk.h');
        coder.cinclude('dynamixel_functions.h');
        coder.ceval('command_dynamixel_current', u1, u2, u3);
    end
end

function releaseImpl(obj) %#ok<MANU>
    if isempty(coder.target)
        % Place simulation termination code here
    else
        % Call C-function implementing device termination
        %coder.ceval('sink_terminate');
    end
end

end

methods (Access=protected)
    %% Define input properties
    function num = getNumInputsImpl(~)

```



```

        num = 3;
    end

function num = getNumOutputsImpl(~)
    num = 0;
end

function flag = isInputSizeLockedImpl(~,~)
    flag = true;
end

function varargout = isInputFixedSizeImpl(~,~)
    varargout{1} = true;
end

function flag = isInputComplexityLockedImpl(~,~)
    flag = true;
end

function validateInputsImpl(~, u1, u2, u3)
    if isempty(coder.target)
        % Run input validation only in Simulation
        validateattributes(u1,{'double'},{'scalar'},'', 'u1');
    end
end

```

```

        validateattributes(u2,{'double'},{'scalar'},'', 'u2');
        validateattributes(u3,{'double'},{'scalar'},'', 'u3');
    end
end

function icon = getIconImpl(~)
    % Define a string as the icon for the System block in Simulink.
    icon = 'MoveArm_Torque';
end

end

methods (Static, Access=protected)
    function simMode = getSimulateUsingImpl(~)
        simMode = 'Interpreted execution';
    end

    function isVisible = showSimulateUsingImpl
        isVisible = false;
    end

    function header = getHeaderImpl
        header = matlab.system.display.Header('MoveArm_Torque','Title',...

```

```

        'Dynamixel Actuator - Torque Control','Text',...
        ['This simulink block sends the input to the Dynamixel actuators. '...
        'The input must be the desired joint torques in [Nm]. '...
        ' The inputs are joint #1, #2, and #3 going from top to bottom.' newline]);
    end

end

methods (Static)
    function name = getDescriptiveName()
        name = 'MoveArm_Torque';
    end

    function b = isSupportedContext(context)
        b = context.isCodeGenTarget('rtw');
    end

    function updateBuildInfo(buildInfo, context)
        if context.isCodeGenTarget('rtw')
            % Update buildInfo
            srcDir = fullfile(fileparts(mfilename('fullpath')),'src');
            includeDir = fullfile(fileparts(mfilename('fullpath')),'include');
            addIncludePaths(buildInfo,includeDir);
        end
    end
end

```

```

% Add all SOURCE files for compiling robotis software
addSourceFiles(buildInfo,'protocol2_packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'protocol1_packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_windows.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_mac.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_linux.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler_arduino.cpp', srcDir);
addSourceFiles(buildInfo,'port_handler.cpp', srcDir);
addSourceFiles(buildInfo,'packet_handler.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_sync_read.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_write.cpp', srcDir);
addSourceFiles(buildInfo,'group_bulk_read.cpp', srcDir);
addSourceFiles(buildInfo,'dynamixel_functions.cpp', srcDir);

% Add all INCLUDE files for compiling robotis software
addIncludeFiles(buildInfo,'protocol2_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'protocol1_packet_handler.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_windows.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_mac.h',includeDir);
addIncludeFiles(buildInfo,'port_handler_linux.h',includeDir);
addIncludeFiles(buildInfo,'port_handler.h',includeDir);

```

```

        addIncludeFiles(buildInfo,'packet_handler.h',includeDir);
        addIncludeFiles(buildInfo,'group_sync_write.h',includeDir);
        addIncludeFiles(buildInfo,'group_sync_read.h',includeDir);
        addIncludeFiles(buildInfo,'group_bulk_write.h',includeDir);
        addIncludeFiles(buildInfo,'group_bulk_read.h',includeDir);
        addIncludeFiles(buildInfo,'dynamixel_sdk.h',includeDir);
        addIncludeFiles(buildInfo,'port_handler_arduino.h',includeDir);
        addIncludeFiles(buildInfo,'dynamixel_functions.h',includeDir)

        %addLinkFlags(buildInfo,{'-lSource'});
        %addLinkObjects(buildInfo,'sourcelib.a',srcDir);
        %addCompileFlags(buildInfo,{'-D_DEBUG=1'});
        %addDefines(buildInfo,'MY_DEFINE_1')
    end
end
end
end
end

```

Custom Source Files

```

// Get methods and members of PortHandlerLinux or PortHandlerWindows
#include <fcntl.h>

```

```

#include <termios.h>
#include <stdlib.h>
#include <stdio.h>
#include <tgmath.h>

// Include the dynamixel headers
#include "dynamixel_sdk.h"
#include "dynamixel_functions.h"

// Define the device name, baudrate, and protocol
#define DEVICENAME                "/dev/ttyUSB0"
#define BAUDRATE                  1
#define PROTOCOL_VERSION          2.0

// Control table addresses
#define ADDR_MX_GOAL_POSITION      116
#define ADDR_MX_PRESENT_POSITION  132
#define ADDR_MX_GOAL_SPEED        104
#define ADDR_MX_GOAL_CURRENT      102
#define ADDR_MX_PRESENT_SPEED     128
#define ADDR_MX_PRESENT_LOAD      126
#define ADDR_MX_POSITION_P_GAIN   84
#define ADDR_MX_POSITION_I_GAIN   82

```

```

#define ADDR_MX_POSITION_D_GAIN      80
#define ADDR_MX_VELOCITY_P_GAIN      78
#define ADDR_MX_VELOCITY_I_GAIN      76
#define ADDR_MX_TORQUE_ENABLE        64
#define ADDR_MX_MAX_POSITION         48
#define ADDR_MX_MIN_POSITION         52
#define ADDR_MX_VELOCITY_LIMIT       44
#define ADDR_MX_ACCELERATION_LIMIT   40
#define ADDR_MX_CURRENT_LIMIT        38
#define ADDR_MX_RETURN_DELAY         9
#define ADDR_MX_DRIVE_MODE           10
#define ADDR_MX_OPERATING_MODE       11
#define ADDR_MX_PWM_LIMIT            36
#define ADDR_MX_GOAL_PWM             100

```

```

// Define the port handler and packet handler variables

```

```

dynamixel::PortHandler *portHandler = dynamixel::PortHandler::getPortHandler(DEVICENAME);

```

```

dynamixel::PacketHandler *packetHandler = dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);

```

```

// Initialize GroupBulkWrite instance

```

```

dynamixel::GroupBulkWrite groupBulkWrite(portHandler, packetHandler);

```

```

// Initialize GroupBulkRead instance

```

```

dynamixel::GroupBulkRead groupBulkRead(portHandler, packetHandler);

void initialize_dynamixel_position_control(double P_GAIN, double I_GAIN, double D_GAIN, double MAX_POSITION,
                                           double MIN_POSITION, double VELOCITY_LIMIT)
{

    // Define the transmission failure code
    int dxl_comm_result    = COMM_TX_FAIL;

    // Open COM port for serial communication with the actuators
    portHandler->openPort();

    // Initialize the dxl_error variable
    uint8_t dxl_error = 0;

    // Set up the motors for position control
    dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 1,
        ADDR_MX_OPERATING_MODE, 3, &dxl_error);
    dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 1,
        ADDR_MX_POSITION_P_GAIN, P_GAIN, &dxl_error);
    dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 1,
        ADDR_MX_POSITION_I_GAIN, I_GAIN, &dxl_error);
    dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 1,

```



```

ADDR_MX_POSITION_D_GAIN, D_GAIN, &dxl_error);
dxl_comm_result    = packetHandler->write4ByteTxRx(portHandler, 1,
ADDR_MX_MAX_POSITION, MAX_POSITION, &dxl_error);
dxl_comm_result    = packetHandler->write4ByteTxRx(portHandler, 1,
ADDR_MX_MIN_POSITION, MIN_POSITION, &dxl_error);
dxl_comm_result    = packetHandler->write4ByteTxRx(portHandler, 1,
ADDR_MX_VELOCITY_LIMIT, VELOCITY_LIMIT, &dxl_error);
dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 1,
ADDR_MX_TORQUE_ENABLE, 1, &dxl_error);

dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 2,
ADDR_MX_OPERATING_MODE, 3, &dxl_error);
dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 2,
ADDR_MX_POSITION_P_GAIN, P_GAIN, &dxl_error);
dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 2,
ADDR_MX_POSITION_I_GAIN, I_GAIN, &dxl_error);
dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 2,
ADDR_MX_POSITION_D_GAIN, D_GAIN, &dxl_error);
dxl_comm_result    = packetHandler->write4ByteTxRx(portHandler, 2,
ADDR_MX_MAX_POSITION, MAX_POSITION, &dxl_error);
dxl_comm_result    = packetHandler->write4ByteTxRx(portHandler, 2,
ADDR_MX_MIN_POSITION, MIN_POSITION, &dxl_error);
dxl_comm_result    = packetHandler->write4ByteTxRx(portHandler, 2,

```

```

    ADDR_MX_VELOCITY_LIMIT, VELOCITY_LIMIT, &dxl_error);
dxl_comm_result  = packetHandler->write1ByteTxRx(portHandler, 2,
    ADDR_MX_TORQUE_ENABLE, 1, &dxl_error);

dxl_comm_result  = packetHandler->write1ByteTxRx(portHandler, 3,
    ADDR_MX_OPERATING_MODE, 3, &dxl_error);
dxl_comm_result  = packetHandler->write2ByteTxRx(portHandler, 3,
    ADDR_MX_POSITION_P_GAIN, P_GAIN, &dxl_error);
dxl_comm_result  = packetHandler->write2ByteTxRx(portHandler, 3,
    ADDR_MX_POSITION_I_GAIN, I_GAIN, &dxl_error);
dxl_comm_result  = packetHandler->write2ByteTxRx(portHandler, 3,
    ADDR_MX_POSITION_D_GAIN, D_GAIN, &dxl_error);
dxl_comm_result  = packetHandler->write4ByteTxRx(portHandler, 3,
    ADDR_MX_MAX_POSITION, MAX_POSITION, &dxl_error);
dxl_comm_result  = packetHandler->write4ByteTxRx(portHandler, 3,
    ADDR_MX_MIN_POSITION, MIN_POSITION, &dxl_error);
dxl_comm_result  = packetHandler->write4ByteTxRx(portHandler, 3,
    ADDR_MX_VELOCITY_LIMIT, VELOCITY_LIMIT, &dxl_error);
dxl_comm_result  = packetHandler->write1ByteTxRx(portHandler, 3,
    ADDR_MX_TORQUE_ENABLE, 1, &dxl_error);

```

```

}

```

```

void initialize_dynamixel_current_control(double CURRENT_LIMIT)
{

    // Define the transmission failure code
    int dxl_comm_result    = COMM_TX_FAIL;

    // Open COM port for serial communication with the actuators
    portHandler->openPort();

    // Initialize the dxl_error variable
    uint8_t dxl_error = 0;

    // Set up the motors for position control
    dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 1,
        ADDR_MX_OPERATING_MODE, 0, &dxl_error);
    dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 1,
        ADDR_MX_CURRENT_LIMIT, CURRENT_LIMIT, &dxl_error);
    dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 1,
        ADDR_MX_TORQUE_ENABLE, 1, &dxl_error);

    dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 2,
        ADDR_MX_OPERATING_MODE, 0, &dxl_error);
    dxl_comm_result    = packetHandler->write2ByteTxRx(portHandler, 2,

```

```

    ADDR_MX_CURRENT_LIMIT, CURRENT_LIMIT, &dxl_error);
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, 2,
    ADDR_MX_TORQUE_ENABLE, 1, &dxl_error);

dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, 3,
    ADDR_MX_OPERATING_MODE, 0, &dxl_error);
dxl_comm_result = packetHandler->write2ByteTxRx(portHandler, 3,
    ADDR_MX_CURRENT_LIMIT, CURRENT_LIMIT, &dxl_error);
dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, 3,
    ADDR_MX_TORQUE_ENABLE, 1, &dxl_error);

}

void command_dynamixel_position(double JOINT1_POS_RAD, double JOINT2_POS_RAD,
    double JOINT3_POS_RAD )
{

    // Define the transmission failure code
    int dxl_comm_result = COMM_TX_FAIL;
    bool dxl_addparam_result = false;
    uint8_t param_goal_position_1[4];
    uint8_t param_goal_position_2[4];
    uint8_t param_goal_position_3[4];

```

```

// Convert the commanded position from radians to raw bits
double JOINT1_POS_BITS = nearbyint(651.74*JOINT1_POS_RAD + 2048);
double JOINT2_POS_BITS = nearbyint(651.74*JOINT2_POS_RAD + 2048);
double JOINT3_POS_BITS = nearbyint(651.74*JOINT3_POS_RAD + 2048);

// Allocate goal position value into byte array
param_goal_position_1[0] = DXL_LOBYTE(DXL_LOWORD(JOINT1_POS_BITS));
param_goal_position_1[1] = DXL_HIBYTE(DXL_LOWORD(JOINT1_POS_BITS));
param_goal_position_1[2] = DXL_LOBYTE(DXL_HIWORD(JOINT1_POS_BITS));
param_goal_position_1[3] = DXL_HIBYTE(DXL_HIWORD(JOINT1_POS_BITS));

param_goal_position_2[0] = DXL_LOBYTE(DXL_LOWORD(JOINT2_POS_BITS));
param_goal_position_2[1] = DXL_HIBYTE(DXL_LOWORD(JOINT2_POS_BITS));
param_goal_position_2[2] = DXL_LOBYTE(DXL_HIWORD(JOINT2_POS_BITS));
param_goal_position_2[3] = DXL_HIBYTE(DXL_HIWORD(JOINT2_POS_BITS));

param_goal_position_3[0] = DXL_LOBYTE(DXL_LOWORD(JOINT3_POS_BITS));
param_goal_position_3[1] = DXL_HIBYTE(DXL_LOWORD(JOINT3_POS_BITS));
param_goal_position_3[2] = DXL_LOBYTE(DXL_HIWORD(JOINT3_POS_BITS));
param_goal_position_3[3] = DXL_HIBYTE(DXL_HIWORD(JOINT3_POS_BITS));

// Initialize the dxl_error variable

```

```

uint8_t dxl_error = 0;

// Send the raw (0->4096) initial position value.
dxl_addparam_result = groupBulkWrite.addParam(1, ADDR_MX_GOAL_POSITION, 4,
    param_goal_position_1);
dxl_addparam_result = groupBulkWrite.addParam(2, ADDR_MX_GOAL_POSITION, 4,
    param_goal_position_2);
dxl_addparam_result = groupBulkWrite.addParam(3, ADDR_MX_GOAL_POSITION, 4,
    param_goal_position_3);

dxl_comm_result = groupBulkWrite.txPacket();
groupBulkWrite.clearParam();
}

void command_dynamixel_current(int JOINT1_BITS, int JOINT2_BITS,
    int JOINT3_BITS )
{
    // Convert torque to amps using performance curve
    //int JOINT1_BITS;
    //int JOINT2_BITS;
    //int JOINT3_BITS;

    //JOINT1_BITS = nearbyint(JOINT1_TAU/0.0030912);

```

```

//JOINT2_BITS = nearbyint(JOINT2_TAU/0.0030912);
//JOINT3_BITS = nearbyint(JOINT3_TAU/0.0030912);

// Define the transmission failure code
int dxl_comm_result    = COMM_TX_FAIL;
bool dxl_addparam_result = false;
uint8_t param_goal_TAU_1[2];
uint8_t param_goal_TAU_2[2];
uint8_t param_goal_TAU_3[2];

// Allocate goal position value into byte array
param_goal_TAU_1[0] = DXL_LOBYTE((JOINT1_BITS));
param_goal_TAU_1[1] = DXL_HIBYTE((JOINT1_BITS));

param_goal_TAU_2[0] = DXL_LOBYTE((JOINT2_BITS));
param_goal_TAU_2[1] = DXL_HIBYTE((JOINT2_BITS));

param_goal_TAU_3[0] = DXL_LOBYTE((JOINT3_BITS));
param_goal_TAU_3[1] = DXL_HIBYTE((JOINT3_BITS));

uint8_t dxl_error = 0;

// Send the raw (0->4096) initial position value.

```

```

dxl_addparam_result = groupBulkWrite.addParam(1, ADDR_MX_GOAL_CURRENT, 2,
    param_goal_TAU_1);
dxl_addparam_result = groupBulkWrite.addParam(2, ADDR_MX_GOAL_CURRENT, 2,
    param_goal_TAU_2);
dxl_addparam_result = groupBulkWrite.addParam(3, ADDR_MX_GOAL_CURRENT, 2,
    param_goal_TAU_3);

dxl_comm_result = groupBulkWrite.txPacket();
groupBulkWrite.clearParam();

}

void terminate_dynamixel()
{
    // Define the transmission failure code
    int dxl_comm_result    = COMM_TX_FAIL;

    uint8_t dxl_error = 0;

    dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 1, ADDR_MX_TORQUE_ENABLE, 0, &dxl_error);
    dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 2, ADDR_MX_TORQUE_ENABLE, 0, &dxl_error);
    dxl_comm_result    = packetHandler->write1ByteTxRx(portHandler, 3, ADDR_MX_TORQUE_ENABLE, 0, &dxl_error);
    // Open COM port for serial communication with the actuators

```



```

    portHandler->closePort();
}

```

Custom Header Files

```

#include "rtwtypes.h"

void initialize_dynamixel_position_control(double, double, double, double, double, double);
void initialize_dynamixel_current_control(double);
void command_dynamixel_position(double, double, double);
void command_dynamixel_current(int, int, int);
void terminate_dynamixel();

```

A.3.2 PhaseSpace Driver Blocks

In this subsection, the code for the PhaseSpace cameras is presented. To properly compile the code presented here, users must also include the sources and header files provided by PhaseSpace, which can be downloaded from <https://www.dropbox.com/sh/efs3hz6d81fqnpb/AADFf1sZp3qfppSnRijumaipa?dl=0>.

MATLAB System Function (Stream PhaseSpace Data)

```

classdef PhaseSpace_ALL < realtime.internal.SourceSampleTime ...
    & coder.ExternalDependency ...
    & matlab.system.mixin.Propagates ...
    & matlab.system.mixin.CustomIcon

```

```

%
% System object template for a source block.
%
% This template includes most, but not all, possible properties,
% attributes, and methods that you can implement for a System object in
% Simulink.
%
% NOTE: When renaming the class name Source, the file name and
% constructor name must be updated to use the class name.
%

% Copyright 2016 The MathWorks, Inc.
%#codegen
%#ok<*EMCA>

properties
    % Platform Selection
    platformSelection = 3;
    sampleRate = 0.1;
end

properties (Nontunable)
    % Public, non-tunable properties.

```

```

end

properties (Access = private)
    % Pre-computed constants.
end

methods
    % Constructor
    function obj = PhaseSpace_ALL(varargin)
        % Support name-value pair arguments when constructing the object.
        setProperties(obj,nargin,varargin{:});
    end
end

methods (Access=protected)
    function setupImpl(obj)
        if isempty(coder.target)
            % Place simulation setup code here
        else
            % Call C-function implementing device initialization
            coder.cinclude('owl.hpp');
            coder.cinclude('phasespace_headers.h');
        end
    end
end

```

```

        coder.ceval('initialize_phasespace',obj.platformSelection, obj.sampleRate);
    end
end

function y = stepImpl(obj)

    y1 = double(0);
    y2 = double(0);
    y3 = double(0);
    y4 = double(0);
    y5 = double(0);
    y6 = double(0);
    y7 = double(0);
    y8 = double(0);
    y9 = double(0);
    y10 = double(0);
    y11 = double(0);
    y12 = double(0);

    y = zeros(1,12,'double');

    if isempty(coder.target)
        % Place simulation output code here
    end
end

```

```

else

    coder.ceval('stream_phasespace',coder.ref(y1),...
                coder.ref(y2),coder.ref(y3),coder.ref(y4),...
                coder.ref(y5),coder.ref(y6),coder.ref(y7),...
                coder.ref(y8),coder.ref(y9),coder.ref(y10),...
                coder.ref(y11),coder.ref(y12),obj.platformSelection);
    y =[y1/1000, y2/1000, y3, y4/15, y5/1000, y6,...
        y7/1000, y8/1000, y9/1000, y10/1000, y11/1000, y12/1000];

end

end

function releaseImpl(obj) %#ok<MANU>
    if isempty(coder.target)
        % Place simulation termination code here
    else
        % Call C-function implementing device termination
        coder.ceval('terminate_phasespace');
    end
end

end

end

```

```

methods (Access=protected)
    %% Define output properties
    function num = getNumInputsImpl(~)
        num = 0;
    end

    function num = getNumOutputsImpl(~)
        num = 1;
    end

    function flag = isOutputSizeLockedImpl(~,~)
        flag = true;
    end

    function varargout = isOutputFixedSizeImpl(~,~)
        varargout{1} = true;
    end

    function flag = isOutputComplexityLockedImpl(~,~)
        flag = true;
    end

    function varargout = isOutputComplexImpl(~)

```

```

        varargout{1} = false;
    end

    function varargout = getOutputSizeImpl(~)
        varargout{1} = [1,12];
    end

    function varargout = getOutputDataTypeImpl(~)
        varargout{1} = 'double';
    end

    function icon = getIconImpl(~)
        % Define a string as the icon for the System block in Simulink.
        icon = 'Stream_PS_ALL';
    end
end

methods (Static, Access=protected)
    function simMode = getSimulateUsingImpl(~)
        simMode = 'Interpreted execution';
    end

    function isVisible = showSimulateUsingImpl

```

```

        isVisible = false;
    end

function header = getHeaderImpl
    header = matlab.system.display.Header('PhaseSpace_ALL','Title',...
        'Initialize and Stream PhaseSpace','Text',...
        ['This simulink block initializes the PhaseSpace cameras to '...
        'collect position data. The sample time should be '...
        'set to 0.1 seconds. The output of this block is a [1x12] vector '...
        'containing the position in [m] and the attitude in [rad].' newline]);
end

end

methods (Static)
    function name = getDescriptiveName()
        name = 'Stream_PS_ALL';
    end

    function b = isSupportedContext(context)
        b = context.isCodeGenTarget('rtw');
    end
end

```



```

function updateBuildInfo(buildInfo, context)
    if context.isCodeGenTarget('rtw')
        % Update buildInfo
        srcDir = fullfile(fileparts(mfilename('fullpath')),'src');
        includeDir = fullfile(fileparts(mfilename('fullpath')),'include');
        addIncludePaths(buildInfo,includeDir);

        % Add the INCLUDE files for the PhaseSpace camera
        addIncludeFiles(buildInfo,'vector.h',includeDir);
        addIncludeFiles(buildInfo,'thread.h',includeDir);
        addIncludeFiles(buildInfo,'socket.h',includeDir);
        addIncludeFiles(buildInfo,'serialize.h',includeDir);
        addIncludeFiles(buildInfo,'protocol.h',includeDir);
        addIncludeFiles(buildInfo,'packet.h',includeDir);
        addIncludeFiles(buildInfo,'owl_rpd.h',includeDir);
        addIncludeFiles(buildInfo,'owl_math.h',includeDir);
        addIncludeFiles(buildInfo,'owl.h',includeDir);
        addIncludeFiles(buildInfo,'libowl.h',includeDir);
        addIncludeFiles(buildInfo,'owl_rpd.hpp',includeDir);
        addIncludeFiles(buildInfo,'owl.hpp',includeDir);
        addIncludeFiles(buildInfo,'phasespace_headers.h',includeDir);

        % Add the SOURCE files for the PhaseSpace camera

```

```

        addSourceFiles(buildInfo, 'socket.cc', srcDir);
        addSourceFiles(buildInfo, 'owl_rpd.cc', srcDir);
        addSourceFiles(buildInfo, 'owl.cc', srcDir);
        addSourceFiles(buildInfo, 'libowl_socket.cc', srcDir);
        addSourceFiles(buildInfo, 'libowl.cc', srcDir);
        addSourceFiles(buildInfo, 'filters.cc', srcDir);
        addSourceFiles(buildInfo, 'phasespace_functions.cpp', srcDir);

        %addLinkFlags(buildInfo, {'-lSource'});
        %addLinkObjects(buildInfo, 'sourcelib.a', srcDir);
        %addCompileFlags(buildInfo, {'-D_DEBUG=1'});
        %addDefines(buildInfo, 'MY_DEFINE_1')
    end
end
end
end
end

```

Custom Source Files

```

#include <iostream>
#include <cstring>
#include <ctime>
#include <cerrno>

```

```

#include <unistd.h>

#include <math.h>
#include <signal.h>

#include "owl.hpp"

using namespace std;

OWL::Context owl;
OWL::Markers markers;
OWL::Rigids rigids;

/* initialize_phasespace() initializes the options for the phasespace cameras
   and starts streaming data. */
double initialize_phasespace(double platformSelection, double sampleTime)
{
    /* The address is the IP address for the phasespace computer. */
    string address = "192.168.0.109";
    std::string phaseSpaceOptions;
    std::string tracker_id_RED_7_pos_string, tracker_id_RED_1_pos_string;
    std::string tracker_id_RED_3_pos_string, tracker_id_RED_5_pos_string;
    std::string tracker_id_BLACK_15_pos_string, tracker_id_BLACK_9_pos_string;

```

```

std::string tracker_id_BLACK_11_pos_string, tracker_id_BLACK_13_pos_string;

/* The options are sent to the phasespace computer. The most important
   setting here is the frequency, which indicates how fast the phasespace
   sends data up to the groundstation. This frequency must be smaller then
   the samplerate of the function reading the data. If the frequency is
   too high, the buffer will fill up and there will be a growing time delay
   in the data. */
double phaseSpaceFrequency = 0.8*(1/sampleTime);
phaseSpaceOptions = "profile=all120 frequency=15";

/* The ID's indicate the location of each LED relative to the center of
   mass for the platform. */
tracker_id_RED_7_pos_string      = "pos=150.906958,123.344136,0";
    tracker_id_RED_1_pos_string    = "pos=150.906958,-154.655864,0";
tracker_id_RED_3_pos_string      = "pos=-128.593042,-155.155864,0";
tracker_id_RED_5_pos_string      = "pos=-127.593042,123.344136,0";
tracker_id_BLACK_15_pos_string   = "pos=122.62,152.11,0";
    tracker_id_BLACK_9_pos_string   = "pos=121.82,-118.12,0";
tracker_id_BLACK_11_pos_string   = "pos=-147.06,-117.56,0";
tracker_id_BLACK_13_pos_string   = "pos=-147.03,152.18,0";

```

```

    const std::string myoptions = phaseSpaceOptions;

/* Open the TCP/IP port and send the options string. */
if (owl.open(address) <= 0 || owl.initialize(myoptions) <= 0)
{
    return 0;
}

if (platformSelection == 1)
{
    /* Create the rigid tracker for the RED satellite. The "rigid tracker"
       refers to the syntax used by phasespace, and it used when you want to
       track a rigid body. */
    uint32_t tracker_id_RED = 0;
    owl.createTracker(tracker_id_RED, "rigid", "RED_rigid");

    /* Assign markers to the rigid body and indicate their positions
       w.r.t the centre of mass (obtained from calibration text file) */
    owl.assignMarker(tracker_id_RED, 7, "7", tracker_id_RED_7_pos_string); // top left
    owl.assignMarker(tracker_id_RED, 1, "1", tracker_id_RED_1_pos_string); // top right
    owl.assignMarker(tracker_id_RED, 3, "3", tracker_id_RED_3_pos_string); // bottom right
    owl.assignMarker(tracker_id_RED, 5, "5", tracker_id_RED_5_pos_string); // bottom left
}

```

```

if (platformSelection == 2)
{
    /* Create the rigid tracker for the RED satellite. The "rigid tracker"
       refers to the syntax used by phasespace, and it used when you want to
       track a rigid body. */
    uint32_t tracker_id_BLACK = 2;
    owl.createTracker(tracker_id_BLACK, "rigid", "BLACK_rigid");

    /* Assign markers to the rigid body and indicate their positions
       w.r.t the centre of mass (obtained from calibration text file) */
    owl.assignMarker(tracker_id_BLACK, 15, "15", tracker_id_BLACK_15_pos_string); // top left
    owl.assignMarker(tracker_id_BLACK, 9, "9", tracker_id_BLACK_9_pos_string); // top right
    owl.assignMarker(tracker_id_BLACK, 11, "11", tracker_id_BLACK_11_pos_string); // bottom right
    owl.assignMarker(tracker_id_BLACK, 13, "13", tracker_id_BLACK_13_pos_string); // bottom left
}
if (platformSelection == 3)
{
    /* Create the rigid tracker for the RED satellite. The "rigid tracker"
       refers to the syntax used by phasespace, and it used when you want to
       track a rigid body. */
    uint32_t tracker_id_RED = 0;
    owl.createTracker(tracker_id_RED, "rigid", "RED_rigid");
}

```

```

/* Assign markers to the rigid body and indicate their positions
   w.r.t the centre of mass (obtained from calibration text file) */
owl.assignMarker(tracker_id_RED, 7, "7", tracker_id_RED_7_pos_string); // top left
owl.assignMarker(tracker_id_RED, 1, "1", tracker_id_RED_1_pos_string); // top right
owl.assignMarker(tracker_id_RED, 3, "3", tracker_id_RED_3_pos_string); // bottom right
owl.assignMarker(tracker_id_RED, 5, "5", tracker_id_RED_5_pos_string); // bottom left

/* Create the rigid tracker for the RED satellite. The "rigid tracker"
   refers to the syntax used by phasespace, and it used when you want to
   track a rigid body. */
uint32_t tracker_id_BLACK = 2;
owl.createTracker(tracker_id_BLACK, "rigid", "BLACK_rigid");

/* Assign markers to the rigid body and indicate their positions
   w.r.t the centre of mass (obtained from calibration text file) */
owl.assignMarker(tracker_id_BLACK, 15, "15", tracker_id_BLACK_15_pos_string); // top left
owl.assignMarker(tracker_id_BLACK, 9, "9", tracker_id_BLACK_9_pos_string); // top right
owl.assignMarker(tracker_id_BLACK, 11, "11", tracker_id_BLACK_11_pos_string); // bottom right
owl.assignMarker(tracker_id_BLACK, 13, "13", tracker_id_BLACK_13_pos_string); // bottom left
}
if (platformSelection == 4)

```

```

{
    /* Create the rigid tracker for the RED satellite. The "rigid tracker"
       refers to the syntax used by phasespace, and it used when you want to
       track a rigid body. */
    uint32_t tracker_id_RED = 0;
    owl.createTracker(tracker_id_RED, "rigid", "RED_rigid");

    /* Assign markers to the rigid body and indicate their positions
       w.r.t the centre of mass (obtained from calibration text file) */
    owl.assignMarker(tracker_id_RED, 7, "7", tracker_id_RED_7_pos_string); // top left
    owl.assignMarker(tracker_id_RED, 1, "1", tracker_id_RED_1_pos_string); // top right
    owl.assignMarker(tracker_id_RED, 3, "3", tracker_id_RED_3_pos_string); // bottom right
    owl.assignMarker(tracker_id_RED, 5, "5", tracker_id_RED_5_pos_string); // bottom left

    /* Create the rigid tracker for the RED satellite. The "rigid tracker"
       refers to the syntax used by phasespace, and it used when you want to
       track a rigid body. */
    uint32_t tracker_id_BLACK = 2;
    owl.createTracker(tracker_id_BLACK, "rigid", "BLACK_rigid");

    /* Assign markers to the rigid body and indicate their positions
       w.r.t the centre of mass (obtained from calibration text file) */
    owl.assignMarker(tracker_id_BLACK, 15, "15", tracker_id_BLACK_15_pos_string); // top left

```



```

owl.assignMarker(tracker_id_BLACK, 9, "9", tracker_id_BLACK_9_pos_string); // top right
owl.assignMarker(tracker_id_BLACK, 11, "11", tracker_id_BLACK_11_pos_string); // bottom right
owl.assignMarker(tracker_id_BLACK, 13, "13", tracker_id_BLACK_13_pos_string); // bottom left

/* Create the rigid tracker for the ARM. The "rigid tracker"
   refers to the syntax used by phasespace, and it used when you want to
   track a rigid body. */
uint32_t tracker_id_RED_arm = 1;
owl.createTracker(tracker_id_RED_arm, "rigid", "RED_arm_flexible");

owl.assignMarker(tracker_id_RED_arm, 2, "2", "pos=0,0,0"); // elbow
owl.assignMarker(tracker_id_RED_arm, 0, "0", "pos=0,0,0"); // wrist
owl.assignMarker(tracker_id_RED_arm, 6, "6", "pos=0,0,0"); // end-effector

}
if (platformSelection == 5)
{
    /* Create the rigid tracker for the RED satellite. The "rigid tracker"
       refers to the syntax used by phasespace, and it used when you want to
       track a rigid body. */
    uint32_t tracker_id_RED = 0;
    owl.createTracker(tracker_id_RED, "rigid", "RED_rigid");
}

```

```

/* Assign markers to the rigid body and indicate their positions
   w.r.t the centre of mass (obtained from calibration text file) */
owl.assignMarker(tracker_id_RED, 7, "7", tracker_id_RED_7_pos_string); // top left
owl.assignMarker(tracker_id_RED, 1, "1", tracker_id_RED_1_pos_string); // top right
owl.assignMarker(tracker_id_RED, 3, "3", tracker_id_RED_3_pos_string); // bottom right
owl.assignMarker(tracker_id_RED, 5, "5", tracker_id_RED_5_pos_string); // bottom left

/* Create the rigid tracker for the ARM. The "rigid tracker"
   refers to the syntax used by phasespace, and it used when you want to
   track a rigid body. */
uint32_t tracker_id_RED_arm = 1;
owl.createTracker(tracker_id_RED_arm, "rigid", "RED_arm_flexible");

owl.assignMarker(tracker_id_RED_arm, 2, "2", "pos=0,0,0"); // elbow
owl.assignMarker(tracker_id_RED_arm, 0, "0", "pos=0,0,0"); // wrist
owl.assignMarker(tracker_id_RED_arm, 6, "6", "pos=0,0,0"); // end-effector
}

/* Start streaming phasespace data. Sending (1) streams data using TCP/IP,
   sending (2) streams data using UDP, and sending (3) streams data using
   UDP but broadcasts to all IP addresses. */
owl.streaming(2);

}

```

```

/* stream_phasespace() is executed during each loop of the simulink diagram.
   This function checks if data is available; if there is data, it is sent
   back to simulink. */
void stream_phasespace(double* XPOS_red, double* YPOS_red,
    double* ATTI_red, double* XPOS_black, double* YPOS_black, double* ATTI_black,
    double* ELBOX_red, double* ELBOY_red, double* WRISX_red, double* WRISY_red,
    double* ENDEX_red, double* ENDEY_red, double platformSelection)
{
    if (platformSelection == 1)
    {
        /* Initialize the "event" parameter. This parameter indicates if there is
           any data available. If there is no data, a zero is returned. */
        const OWL::Event *event = owl.nextEvent(1000);

        /* If the connection is available and the properties are initialized,
           check if there is data available. If there is data, then check which
           rigid body has been located. Then, loop through the different IDs
           found and store the data. */
        if (owl.isOpen() && owl.property<int>("initialized"))
        {
            if (!event)
            {

```

```

        // Do not do anything! There is no good data.
    }
    else if (event->type_id() == OWL::Type::FRAME)
    {
        if (event->find("rigids", rigids) > 0)
        {
            for (OWL::Rigids::iterator r = rigids.begin(); r != rigids.end(); r++)
            {
                if (r->cond > 0)
                {
                    if (r->id == 0)
                    {
                        *XPOS_red = r->pose[0];
                        *YPOS_red = r->pose[1];
                        *ATTI_red = atan2(2 * r->pose[4] * r->pose[5]
                                         + 2 * r->pose[3] * r->pose[6],
                                         2 * r->pose[3] * r->pose[3] - 1
                                         + 2 * r->pose[4] * r->pose[4]);
                        *XPOS_black = event->time();
                        *YPOS_black = 0;
                        *ATTI_black = 0;
                        *ELBOX_red = 0;
                        *WRISX_red = 0;
                    }
                }
            }
        }
    }

```



```

{
    if (!event)
    {
        // Do not do anything! There is no good data.
    }
    else if (event->type_id() == OWL::Type::FRAME)
    {
        if (event->find("rigids", rigids) > 0)
        {
            for (OWL::Rigids::iterator r = rigids.begin(); r != rigids.end(); r++)
            {
                if (r->cond > 0)
                {
                    if (r->id == 2)
                    {
                        *XPOS_red = 0;
                        *YPOS_red = 0;
                        *ATTI_red = 0;
                        *XPOS_black = r->pose[0];
                        *YPOS_black = r->pose[1];
                        *ATTI_black = atan2(2 * r->pose[4] * r->pose[5]
                                           + 2 * r->pose[3] * r->pose[6],
                                           2 * r->pose[3] * r->pose[3] - 1

```



```

    rigid body has been located. Then, loop through the different IDs
    found and store the data. */
if (owl.isOpen() && owl.property<int>("initialized"))
{
    if (!event)
    {
        // Do not do anything! There is no good data.
    }
    else if (event->type_id() == OWL::Type::FRAME)
    {
        if (event->find("rigids", rigids) > 0)
        {
            for (OWL::Rigids::iterator r = rigids.begin(); r != rigids.end(); r++)
            {
                if (r->cond > 0)
                {
                    if (r->id == 0)
                    {
                        *XPOS_red = r->pose[0];
                        *YPOS_red = r->pose[1];
                        *ATTI_red = atan2(2 * r->pose[4] * r->pose[5]
                                         + 2 * r->pose[3] * r->pose[6],
                                         2 * r->pose[3] * r->pose[3] - 1

```



```

        + 2 * r->pose[4] * r->pose[4]);
*ELBOX_red    = 0;
*WRISX_red    = 0;
*ENDEX_red    = 0;
*ELBOY_red    = 0;
*WRISY_red    = 0;
*ENDEY_red    = 0;
}
if (r->id == 2)
{
    *XPOS_black = r->pose[0];
    *YPOS_black = r->pose[1];
    *ATTI_black = atan2(2 * r->pose[4] * r->pose[5]
        + 2 * r->pose[3] * r->pose[6],
        2 * r->pose[3] * r->pose[3] - 1
        + 2 * r->pose[4] * r->pose[4]);
    *ELBOX_red    = 0;
    *WRISX_red    = 0;
    *ENDEX_red    = 0;
    *ELBOY_red    = 0;
    *WRISY_red    = 0;
    *ENDEY_red    = 0;
    /*current_time = event->time();

```



```

else if (event->type_id() == OWL::Type::FRAME)
{
    if (event->find("rigids", rigids) > 0)
    {
        for (OWL::Rigids::iterator r = rigids.begin(); r != rigids.end(); r++)
        {
            if (r->cond > 0)
            {
                if (r->id == 0)
                {
                    *XPOS_red = r->pose[0];
                    *YPOS_red = r->pose[1];
                    *ATTI_red = atan2(2 * r->pose[4] * r->pose[5]
                                     + 2 * r->pose[3] * r->pose[6],
                                     2 * r->pose[3] * r->pose[3] - 1
                                     + 2 * r->pose[4] * r->pose[4]);
                }
                if (r->id == 2)
                {
                    *XPOS_black = r->pose[0];
                    *YPOS_black = r->pose[1];
                    *ATTI_black = atan2(2 * r->pose[4] * r->pose[5]
                                       + 2 * r->pose[3] * r->pose[6],
                                       2 * r->pose[3] * r->pose[3] - 1
                                       + 2 * r->pose[4] * r->pose[4]);
                }
            }
        }
    }
}

```

```

                2 * r->pose[3] * r->pose[3] - 1
                + 2 * r->pose[4] * r->pose[4]);
            /*current_time = event->time();
        }
    }
}

if (event->find("markers", markers) > 0)
{
    for (OWL::Markers::iterator m = markers.begin(); m != markers.end(); m++)
    {
        if (m->cond > 0)
        {
            if (m->id == 2)
            {
                *ELBOX_red    = m->x;
                *ELBOY_red    = m->y;
            }
            if (m->id == 0)
            {
                *WRISX_red    = m->x;
                *WRISY_red    = m->y;
            }
        }
    }
}

```



```

if (!event)
{
    // Do not do anything! There is no good data.
}
else if (event->type_id() == OWL::Type::FRAME)
{
    if (event->find("rigids", rigids) > 0)
    {
        for (OWL::Rigids::iterator r = rigids.begin(); r != rigids.end(); r++)
        {
            if (r->cond > 0)
            {
                if (r->id == 0)
                {
                    *XPOS_red = r->pose[0];
                    *YPOS_red = r->pose[1];
                    *ATTI_red = atan2(2 * r->pose[4] * r->pose[5]
                                     + 2 * r->pose[3] * r->pose[6],
                                     2 * r->pose[3] * r->pose[3] - 1
                                     + 2 * r->pose[4] * r->pose[4]);
                    *XPOS_black = 0;
                    *YPOS_black = 0;
                    *ATTI_black = 0;
                }
            }
        }
    }
}

```

```

        }
    }
}

if (event->find("markers", markers) > 0)
{
    for (OWL::Markers::iterator m = markers.begin(); m != markers.end(); m++)
    {
        if (m->cond > 0)
        {
            if (m->id == 2)
            {
                *ELBOX_red    = m->x;
                *ELBOY_red    = m->y;
            }
            if (m->id == 0)
            {
                *WRISX_red    = m->x;
                *WRISY_red    = m->y;
            }
            if (m->id == 6)
            {
                *ENDEX_red    = m->x;

```

```

        *ENDEY_red    = m->y;
    }
}
}
}
}
}
}
}

/* terminate_phasespace() stops all cameras from running and closes all
   communication ports/clears all buffers */
void terminate_phasespace()
{
    owl.done();
    owl.close();
}

```

Custom Header Files

```

#include "rtwtypes.h"

double initialize_phasespace(double, double);
void stream_phasespace(double *, double *, double *, double *, double *, double *,

```



```
        double *, double *, double *, double *, double *, double *,
        double);
void terminate_phasespace();
```

Appendix B

Supplementary Documentation - Chapter #3

Here, the script used to symbolically derive the equations of motion will be provided, in addition to the final elements for the dynamics model representing the space robot used in SPOT.

B.1 Derivation of Dynamics Model - MATLAB Script

```
clear;
clc;

%=====
% This script derives the equations of motion for SPOT.
% Author: Alex
% Creation Date: 2016-08-04
% Last Edit:      2018-01-18
%=====

% Define symbolic variables
syms a1 a2 a3 b0 b1 b2 b3 q0 q1 q2 q3 m0 m1 m2 m3 x0_dot y0_dot...
q0_dot...
      q1_dot q2_dot q3_dot I0 I1 I2 I3 x0 y0 x_dot phi...
      fx fy t0 t1 t2 t3

% Define position vectors to each center of mass
r0 = [ x0
      y0 ];
```

```

r1          = [ x0 + (b0)*cos(phi + q0) + (a1)*...
               cos(pi/2 + q0 + q1)
               y0 + (b0)*sin(phi + q0) + (a1)*...
               sin(pi/2 + q0 + q1) ];

r2          = [ x0 + (b0)*cos(phi + q0) + (a1+b1)*...
               cos(pi/2 + q0 + q1) + (a2)*...
               cos(pi/2 + q0 + q1 + q2)
               y0 + (b0)*sin(phi + q0) + (a1+b1)*...
               sin(pi/2 + q0 + q1) + (a2)*...
               sin(pi/2 + q0 + q1 + q2) ];

r3          = [ x0 + (b0)*cos(phi + q0) + (a1+b1)*...
               cos(pi/2 + q0 + q1) + (a2+b2)*...
               cos(pi/2 + q0 + q1 + q2) + (a3)*...
               cos(pi/2 + q0 + q1 + q2 + q3)
               y0 + (b0)*sin(phi + q0) + (a1+b1)*...
               sin(pi/2 + q0 + q1) + (a2+b2)*...
               sin(pi/2 + q0 + q1 + q2) + (a3)*...
               sin(pi/2 + q0 + q1 + q2 + q3)];

% Take the Jacobians
JM_r0       = jacobian(r0,[x0,y0,q0,q1,q2,q3]);
JM_r1       = jacobian(r1,[x0,y0,q0,q1,q2,q3]);
JM_r2       = jacobian(r2,[x0,y0,q0,q1,q2,q3]);
JM_r3       = jacobian(r3,[x0,y0,q0,q1,q2,q3]);

JM_r0_T     = transpose(JM_r0);

```

```

JM_r1_T          = transpose(JM_r1);
JM_r2_T          = transpose(JM_r2);
JM_r3_T          = transpose(JM_r3);

% Calculate linear kinetic energy
Trv              = ((m0*JM_r0_T*JM_r0) + (m1*JM_r1_T*JM_r1) + ...
                    (m2*JM_r2_T*JM_r2) + (m3*JM_r3_T*JM_r3));

wr0              = [0; 0; q0]; wr1 = [0; 0; q0+q1];
wr2              = [0; 0; q0+q1+q2]; wr3 = [0; 0; q0+q1+q2+q3];

JM_wr0           = jacobian(wr0,[x0,y0,q0,q1,q2,q3]);
JM_wr1           = jacobian(wr1,[x0,y0,q0,q1,q2,q3]);
JM_wr2           = jacobian(wr2,[x0,y0,q0,q1,q2,q3]);
JM_wr3           = jacobian(wr3,[x0,y0,q0,q1,q2,q3]);

JM_wr0_T         = transpose(JM_wr0);
JM_wr1_T         = transpose(JM_wr1);
JM_wr2_T         = transpose(JM_wr2);
JM_wr3_T         = transpose(JM_wr3);

% Calculate angular kinetic energy
Trw              = ((I0*JM_wr0_T*JM_wr0) + (I1*JM_wr1_T*JM_wr1) + ...
                    (I2*JM_wr2_T*JM_wr2) + (I3*JM_wr3_T*JM_wr3));

% Calculate inertia matrix
H                = (Trv + Trw);
q                = [x0,y0,q0,q1,q2,q3];
qdot             = [x0_dot, y0_dot, q0_dot, q1_dot, q2_dot, q3_dot];

% Calculate coriolis matrix

```

```

for k = 1:6
    for i = 1:6
        for j = 1:6
            C(i,j,k) = 0.5*(diff(H(k,j),q(i)) + diff(H(k,i),q(j))...
                - diff(H(i,j),q(k)));
        end
    end
end

for k = 1:6
    for j = 1:6
        c(j,k) = C(1,j,k)*qdot(1) + C(2,j,k)*qdot(2) +...
            C(3,j,k)*qdot(3) + C(4,j,k)*qdot(4) + ...
            C(5,j,k)*qdot(5) + C(6,j,k)*qdot(6);
    end
end

Coriolis = transpose(c); Inertia = H;

% Export functions
InertiaS = collect(simplify(Inertia),{'sin' 'cos'});
InertiaFunc = matlabFunction(InertiaS,'File','InertiaFunc3LINK');
CoriolisS = collect(simplify(Coriolis),{'sin' 'cos'});
CoriolisFunc = matlabFunction(CoriolisS,'File','CoriolisFunc3LINK');

```

B.2 Elements of the Space Robot Inertia Matrix

The elements $M_{ij} \forall i = 1...6, j = 1...6$ are defined below:

$$M_{11} = m_0 + m_1 + m_2 + m_3$$

$$M_{12} = 0$$

$$M_{13} = (-a_1 m_1 - m_2 (a_1 + b_1) - m_3 (a_1 + b_1)) \cos(q_0 + q_1) + (-a_2 m_2 - m_3 (a_2 + b_2)) \cos(q_0 + q_1 + q_2) \\ + (-a_3 m_3) \cos(q_0 + q_1 + q_2 + q_3) + (-b_0 m_1 - b_0 m_2 - b_0 m_3) \sin(\phi + q_0)$$

$$M_{14} = (-a_1 m_1 - m_2 (a_1 + b_1) - m_3 (a_1 + b_1)) \cos(q_0 + q_1) + (-a_2 m_2 - m_3 (a_2 + b_2)) \cos(q_0 + q_1 + q_2) \\ + (-a_3 m_3) \cos(q_0 + q_1 + q_2 + q_3)$$

$$M_{15} = (-a_2 m_2 - a_2 m_3 - b_2 m_3) \cos(q_0 + q_1 + q_2) + (-a_3 m_3) \cos(q_0 + q_1 + q_2 + q_3)$$

$$M_{16} = (-a_3 m_3) \cos(q_0 + q_1 + q_2 + q_3)$$

$$M_{21} = 0$$

$$M_{22} = m_0 + m_1 + m_2 + m_3$$

$$M_{23} = (b_0 m_1 + b_0 m_2 + b_0 m_3) \cos(\phi + q_0) + (-a_1 m_1 - m_2 (a_1 + b_1) - m_3 (a_1 + b_1)) \sin(q_0 + q_1) \\ + (-a_2 m_2 - m_3 (a_2 + b_2)) \sin(q_0 + q_1 + q_2) + (-a_3 m_3) \sin(q_0 + q_1 + q_2 + q_3)$$

$$M_{24} = (-a_1 m_1 - m_2 (a_1 + b_1) - m_3 (a_1 + b_1)) \sin(q_0 + q_1) + (-a_2 m_2 - m_3 (a_2 + b_2)) \sin(q_0 + q_1 + q_2) \\ + (-a_3 m_3) \sin(q_0 + q_1 + q_2 + q_3)$$

$$M_{25} = (-a_2 m_2 - a_2 m_3 - b_2 m_3) \sin(q_0 + q_1 + q_2) + (-a_3 m_3) \sin(q_0 + q_1 + q_2 + q_3)$$

$$M_{26} = (-a_3 m_3) \sin(q_0 + q_1 + q_2 + q_3)$$

$$M_{31} = M_{13}$$

$$M_{32} = M_{23}$$

$$\begin{aligned} M_{33} = & (2 a_1 a_2 m_2 + 2 a_1 a_2 m_3 + 2 a_2 b_1 m_2 + 2 a_1 b_2 m_3 + 2 a_2 b_1 m_3 + 2 b_1 b_2 m_3) \cos(q_2) \\ & + (2 a_2 a_3 m_3 + 2 a_3 b_2 m_3) \cos(q_3) + (2 a_1 a_3 m_3 + 2 a_3 b_1 m_3) \cos(q_2 + q_3) \\ & + (2 a_1 b_0 m_1 + 2 a_1 b_0 m_2 + 2 a_1 b_0 m_3 + 2 b_0 b_1 m_2 + 2 b_0 b_1 m_3) \sin(\phi - q_1) \\ & + (-2 a_2 b_0 m_2 - 2 a_2 b_0 m_3 - 2 b_0 b_2 m_3) \sin(q_1 - \phi + q_2) + (-2 a_3 b_0 m_3) \sin(q_1 - \phi + q_2 + q_3) \\ & + I_0 + I_1 + I_2 + I_3 + a_1^2 m_1 + a_1^2 m_2 + a_1^2 m_3 + a_2^2 m_2 + a_2^2 m_3 + a_3^2 m_3 + b_0^2 m_1 + b_0^2 m_2 \\ & + b_0^2 m_3 + b_1^2 m_2 + b_1^2 m_3 + b_2^2 m_3 + 2 a_1 b_1 m_2 + 2 a_1 b_1 m_3 + 2 a_2 b_2 m_3 \end{aligned}$$

$$\begin{aligned} M_{34} = & (2 a_1 a_2 m_2 + 2 a_1 a_2 m_3 + 2 a_2 b_1 m_2 + 2 a_1 b_2 m_3 + 2 a_2 b_1 m_3 + 2 b_1 b_2 m_3) \cos(q_2) \\ & + (2 a_2 a_3 m_3 + 2 a_3 b_2 m_3) \cos(q_3) + (2 a_1 a_3 m_3 + 2 a_3 b_1 m_3) \cos(q_2 + q_3) \\ & + (a_1 b_0 m_1 + a_1 b_0 m_2 + a_1 b_0 m_3 + b_0 b_1 m_2 + b_0 b_1 m_3) \sin(\phi - q_1) \\ & + (-a_2 b_0 m_2 - a_2 b_0 m_3 - b_0 b_2 m_3) \sin(q_1 - \phi + q_2) + (-a_3 b_0 m_3) \sin(q_1 - \phi + q_2 + q_3) \\ & + I_1 + I_2 + I_3 + a_1^2 m_1 + a_1^2 m_2 + a_1^2 m_3 + a_2^2 m_2 + a_2^2 m_3 + a_3^2 m_3 + b_1^2 m_2 + b_1^2 m_3 + b_2^2 m_3 \\ & + 2 a_1 b_1 m_2 + 2 a_1 b_1 m_3 + 2 a_2 b_2 m_3 \end{aligned}$$

$$\begin{aligned} M_{35} = & (a_1 a_2 m_2 + a_1 a_2 m_3 + a_2 b_1 m_2 + a_1 b_2 m_3 + a_2 b_1 m_3 + b_1 b_2 m_3) \cos(q_2) \\ & + (2 a_2 a_3 m_3 + 2 a_3 b_2 m_3) \cos(q_3) + (a_1 a_3 m_3 + a_3 b_1 m_3) \cos(q_2 + q_3) \\ & + (-a_2 b_0 m_2 - a_2 b_0 m_3 - b_0 b_2 m_3) \sin(q_1 - \phi + q_2) + (-a_3 b_0 m_3) \sin(q_1 - \phi + q_2 + q_3) \\ & + I_2 + I_3 + a_2^2 m_2 + a_2^2 m_3 + a_3^2 m_3 + b_2^2 m_3 + 2 a_2 b_2 m_3 \end{aligned}$$

$$\begin{aligned} M_{36} = & (a_2 a_3 m_3 + a_3 b_2 m_3) \cos(q_3) + (a_1 a_3 m_3 + a_3 b_1 m_3) \cos(q_2 + q_3) \\ & + (-a_3 b_0 m_3) \sin(q_1 - \phi + q_2 + q_3) + m_3 a_3^2 + I_3 \end{aligned}$$

$$M_{41} = M_{14}$$

$$M_{42} = M_{24}$$

$$M_{43} = M_{34}$$

$$\begin{aligned} M_{44} = & (2 a_1 a_2 m_2 + 2 a_1 a_2 m_3 + 2 a_2 b_1 m_2 + 2 a_1 b_2 m_3 + 2 a_2 b_1 m_3 + 2 b_1 b_2 m_3) \cos(q_2) \\ & + (2 a_2 a_3 m_3 + 2 a_3 b_2 m_3) \cos(q_3) + (2 a_1 a_3 m_3 + 2 a_3 b_1 m_3) \cos(q_2 + q_3) + I_1 + I_2 + I_3 + a_1^2 m_1 \\ & + a_1^2 m_2 + a_1^2 m_3 + a_2^2 m_2 + a_2^2 m_3 + a_3^2 m_3 + b_1^2 m_2 + b_1^2 m_3 + b_2^2 m_3 + 2 a_1 b_1 m_2 + 2 a_1 b_1 m_3 \\ & + 2 a_2 b_2 m_3 \end{aligned}$$

$$\begin{aligned} M_{45} = & (a_1 a_2 m_2 + a_1 a_2 m_3 + a_2 b_1 m_2 + a_1 b_2 m_3 + a_2 b_1 m_3 + b_1 b_2 m_3) \cos(q_2) \\ & + (2 a_2 a_3 m_3 + 2 a_3 b_2 m_3) \cos(q_3) + (a_1 a_3 m_3 + a_3 b_1 m_3) \cos(q_2 + q_3) + I_2 + I_3 + a_2^2 m_2 + a_2^2 m_3 \\ & + a_3^2 m_3 + b_2^2 m_3 + 2 a_2 b_2 m_3 \end{aligned}$$

$$M_{46} = (a_2 a_3 m_3 + a_3 b_2 m_3) \cos(q_3) + (a_1 a_3 m_3 + a_3 b_1 m_3) \cos(q_2 + q_3) + m_3 a_3^2 + I_3$$

$$M_{51} = M_{15}$$

$$M_{52} = M_{25}$$

$$M_{53} = M_{35}$$

$$M_{54} = M_{45}$$

$$M_{55} = (2 a_2 a_3 m_3 + 2 a_3 b_2 m_3) \cos(q_3) + I_2 + I_3 + a_2^2 m_2 + a_2^2 m_3 + a_3^2 m_3 + b_2^2 m_3 + 2 a_2 b_2 m_3$$

$$M_{56} = (a_2 a_3 m_3 + a_3 b_2 m_3) \cos(q_3) + m_3 a_3^2 + I_3$$

$$M_{61} = M_{16}$$

$$M_{62} = M_{26}$$

$$M_{63} = M_{36}$$

$$M_{64} = M_{46}$$

$$M_{65} = M_{56}$$

$$M_{66} = m_3 a_3^2 + I_3$$

B.3 Elements of the Space Robot Centrifugal/Coriolis Matrix

From these components, as described in the body of the thesis, the components $C_{ij} \forall i = 1...6, j = 1...6$ are derived and presented below:

$$C_{11} = 0$$

$$C_{12} = 0$$

$$\begin{aligned} C_{13} = & (-\dot{q}_0 (b_0 m_1 + b_0 m_2 + b_0 m_3)) \cos(\phi + q_0) \\ & + (-\dot{q}_0 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1)) - \dot{q}_1 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1))) \cos(q_0 + q_1 + \frac{\pi}{2}) \\ & + (-\dot{q}_0 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_1 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_2 (a_2 m_2 + m_3 (a_2 + b_2))) \cos(q_0 + q_1 + q_2 + \frac{\pi}{2}) \\ & + (-a_3 m_3 \dot{q}_0 - a_3 m_3 \dot{q}_1 - a_3 m_3 \dot{q}_2 - a_3 m_3 \dot{q}_3) \cos(q_0 + q_1 + q_2 + q_3 + \frac{\pi}{2}) \end{aligned}$$

$$\begin{aligned} C_{14} = & (-\dot{q}_0 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1)) - \dot{q}_1 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1))) \cos(q_0 + q_1 + \frac{\pi}{2}) \\ & + (-\dot{q}_0 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_1 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_2 (a_2 m_2 + m_3 (a_2 + b_2))) \cos(q_0 + q_1 + q_2 + \frac{\pi}{2}) \\ & + (-a_3 m_3 \dot{q}_0 - a_3 m_3 \dot{q}_1 - a_3 m_3 \dot{q}_2 - a_3 m_3 \dot{q}_3) \cos(q_0 + q_1 + q_2 + q_3 + \frac{\pi}{2}) \end{aligned}$$

$$\begin{aligned} C_{15} = & (-\dot{q}_0 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_1 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_2 (a_2 m_2 + m_3 (a_2 + b_2))) \cos(q_0 + q_1 + q_2 + \frac{\pi}{2}) \\ & + (-a_3 m_3 \dot{q}_0 - a_3 m_3 \dot{q}_1 - a_3 m_3 \dot{q}_2 - a_3 m_3 \dot{q}_3) \cos(q_0 + q_1 + q_2 + q_3 + \frac{\pi}{2}) \end{aligned}$$

$$C_{16} = (a_3 m_3 (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \sin(q_0 + q_1 + q_2 + q_3)$$

$$C_{21} = 0$$

$$C_{22} = 0$$

$$\begin{aligned} C_{23} = & (-\dot{q}_0 (b_0 m_1 + b_0 m_2 + b_0 m_3)) \sin(\phi + q_0) \\ & + (-\dot{q}_0 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1)) - \dot{q}_1 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1))) \sin(q_0 + q_1 + \frac{\pi}{2}) \\ & + (-\dot{q}_0 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_1 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_2 (a_2 m_2 + m_3 (a_2 + b_2))) \sin(q_0 + q_1 + q_2 + \frac{\pi}{2}) \end{aligned}$$

$$\begin{aligned}
& + (-a_3 m_3 \dot{q}_0 - a_3 m_3 \dot{q}_1 - a_3 m_3 \dot{q}_2 - a_3 m_3 \dot{q}_3) \sin(q_0 + q_1 + q_2 + q_3 + \frac{\pi}{2}) \\
C_{24} = & (-\dot{q}_0 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1)) - \dot{q}_1 (a_1 m_1 + m_2 (a_1 + b_1) + m_3 (a_1 + b_1))) \sin(q_0 + q_1 + \frac{\pi}{2}) \\
& + (-\dot{q}_0 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_1 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_2 (a_2 m_2 + m_3 (a_2 + b_2))) \sin(q_0 + q_1 + q_2 + \frac{\pi}{2}) \\
& + (-a_3 m_3 \dot{q}_0 - a_3 m_3 \dot{q}_1 - a_3 m_3 \dot{q}_2 - a_3 m_3 \dot{q}_3) \sin(q_0 + q_1 + q_2 + q_3 + \frac{\pi}{2}) \\
C_{25} = & (-\dot{q}_0 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_1 (a_2 m_2 + m_3 (a_2 + b_2)) - \dot{q}_2 (a_2 m_2 + m_3 (a_2 + b_2))) \sin(q_0 + q_1 + q_2 + \frac{\pi}{2}) \\
& + (-a_3 m_3 \dot{q}_0 - a_3 m_3 \dot{q}_1 - a_3 m_3 \dot{q}_2 - a_3 m_3 \dot{q}_3) \sin(q_0 + q_1 + q_2 + q_3 + \frac{\pi}{2}) \\
C_{26} = & (-a_3 m_3 (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \cos(q_0 + q_1 + q_2 + q_3) \\
C_{31} = & 0 \\
C_{32} = & 0 \\
C_{33} = & (-a_1 b_0 m_1 \dot{q}_1 - a_1 b_0 m_2 \dot{q}_1 - a_1 b_0 m_3 \dot{q}_1 - b_0 b_1 m_2 \dot{q}_1 - b_0 b_1 m_3 \dot{q}_1) \cos(\phi - q_1) \\
& + (-a_2 b_0 m_2 \dot{q}_1 - a_2 b_0 m_2 \dot{q}_2 - a_2 b_0 m_3 \dot{q}_1 - a_2 b_0 m_3 \dot{q}_2 - b_0 b_2 m_3 \dot{q}_1 - b_0 b_2 m_3 \dot{q}_2) \cos(q_1 - \phi + q_2) \\
& + (-a_3 b_0 m_3 \dot{q}_1 - a_3 b_0 m_3 \dot{q}_2 - a_3 b_0 m_3 \dot{q}_3) \cos(q_1 - \phi + q_2 + q_3) \\
& + (-a_2 b_1 m_2 \dot{q}_2 - a_1 b_2 m_3 \dot{q}_2 - a_2 b_1 m_3 \dot{q}_2 - b_1 b_2 m_3 \dot{q}_2 - a_1 a_2 m_2 \dot{q}_2 - a_1 a_2 m_3 \dot{q}_2) \sin(q_2) \\
& + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) \\
& + (-a_3 b_1 m_3 \dot{q}_2 - a_3 b_1 m_3 \dot{q}_3 - a_1 a_3 m_3 \dot{q}_2 - a_1 a_3 m_3 \dot{q}_3) \sin(q_2 + q_3) \\
C_{34} = & (-a_1 b_0 m_1 \dot{q}_0 - a_1 b_0 m_1 \dot{q}_1 - a_1 b_0 m_2 \dot{q}_0 - a_1 b_0 m_2 \dot{q}_1 - a_1 b_0 m_3 \dot{q}_0 - a_1 b_0 m_3 \dot{q}_1 \\
& - b_0 b_1 m_2 \dot{q}_0 - b_0 b_1 m_2 \dot{q}_1 - b_0 b_1 m_3 \dot{q}_0 - b_0 b_1 m_3 \dot{q}_1) \cos(\phi - q_1) + (-a_2 b_0 m_2 \dot{q}_0 - a_2 b_0 m_2 \dot{q}_1 \\
& - a_2 b_0 m_3 \dot{q}_0 - a_2 b_0 m_2 \dot{q}_2 - a_2 b_0 m_3 \dot{q}_1 - a_2 b_0 m_3 \dot{q}_2 - b_0 b_2 m_3 \dot{q}_0 - b_0 b_2 m_3 \dot{q}_1 - b_0 b_2 m_3 \dot{q}_2) \\
& \cos(q_1 - \phi + q_2) + (-a_3 b_0 m_3 \dot{q}_0 - a_3 b_0 m_3 \dot{q}_1 - a_3 b_0 m_3 \dot{q}_2 - a_3 b_0 m_3 \dot{q}_3) \cos(q_1 - \phi + q_2 + q_3) \\
& + (-a_2 b_1 m_2 \dot{q}_2 - a_1 b_2 m_3 \dot{q}_2 - a_2 b_1 m_3 \dot{q}_2 - b_1 b_2 m_3 \dot{q}_2 - a_1 a_2 m_2 \dot{q}_2 - a_1 a_2 m_3 \dot{q}_2) \sin(q_2) \\
& + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) + (-a_3 b_1 m_3 \dot{q}_2 - a_3 b_1 m_3 \dot{q}_3 - a_1 a_3 m_3 \dot{q}_2 - a_1 a_3 m_3 \dot{q}_3)
\end{aligned}$$

$$\begin{aligned}
& \sin(q_2 + q_3) \\
C_{35} = & (-a_2 b_0 m_2 \dot{q}_0 - a_2 b_0 m_2 \dot{q}_1 - a_2 b_0 m_3 \dot{q}_0 - a_2 b_0 m_2 \dot{q}_2 - a_2 b_0 m_3 \dot{q}_1 - a_2 b_0 m_3 \dot{q}_2 \\
& - b_0 b_2 m_3 \dot{q}_0 - b_0 b_2 m_3 \dot{q}_1 - b_0 b_2 m_3 \dot{q}_2) \cos(q_1 - \phi + q_2) + (-a_3 b_0 m_3 \dot{q}_0 - a_3 b_0 m_3 \dot{q}_1 \\
& - a_3 b_0 m_3 \dot{q}_2 - a_3 b_0 m_3 \dot{q}_3) \cos(q_1 - \phi + q_2 + q_3) + (-a_2 b_1 m_2 \dot{q}_0 - a_1 b_2 m_3 \dot{q}_0 \\
& - a_2 b_1 m_2 \dot{q}_1 - a_2 b_1 m_3 \dot{q}_0 - a_1 b_2 m_3 \dot{q}_1 - a_2 b_1 m_2 \dot{q}_2 - a_2 b_1 m_3 \dot{q}_1 \\
& - a_1 b_2 m_3 \dot{q}_2 - a_2 b_1 m_3 \dot{q}_2 - b_1 b_2 m_3 \dot{q}_0) \sin(q_2) + (-b_1 b_2 m_3 \dot{q}_1 - b_1 b_2 m_3 \dot{q}_2 \\
& - a_1 a_2 m_2 \dot{q}_0 - a_1 a_2 m_2 \dot{q}_1 - a_1 a_2 m_3 \dot{q}_0 - a_1 a_2 m_2 \dot{q}_2 - a_1 a_2 m_3 \dot{q}_1 \\
& - a_1 a_2 m_3 \dot{q}_2) \sin(q_2) + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) \\
& + (-a_3 b_1 m_3 \dot{q}_0 - a_3 b_1 m_3 \dot{q}_1 - a_3 b_1 m_3 \dot{q}_2 - a_3 b_1 m_3 \dot{q}_3 - a_1 a_3 m_3 \dot{q}_0 - a_1 a_3 m_3 \dot{q}_1 \\
& - a_1 a_3 m_3 \dot{q}_2 - a_1 a_3 m_3 \dot{q}_3) \sin(q_2 + q_3) \\
C_{36} = & (-a_3 b_0 m_3 (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \cos(q_1 - \phi + q_2 + q_3) + (-a_3 m_3 (a_2 + b_2) (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \sin(q_3) \\
& + (-a_3 m_3 (a_1 + b_1) (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \sin(q_2 + q_3) \\
C_{41} = & 0 \\
C_{42} = & 0 \\
C_{43} = & (a_1 b_0 m_1 \dot{q}_0 + a_1 b_0 m_2 \dot{q}_0 + a_1 b_0 m_3 \dot{q}_0 + b_0 b_1 m_2 \dot{q}_0 + b_0 b_1 m_3 \dot{q}_0) \cos(\phi - q_1) \\
& + (a_2 b_0 m_2 \dot{q}_0 + a_2 b_0 m_3 \dot{q}_0 + b_0 b_2 m_3 \dot{q}_0) \cos(q_1 - \phi + q_2) + (a_3 b_0 m_3 \dot{q}_0) \cos(q_1 - \phi + q_2 + q_3) \\
& + (-a_2 b_1 m_2 \dot{q}_2 - a_1 b_2 m_3 \dot{q}_2 - a_2 b_1 m_3 \dot{q}_2 - b_1 b_2 m_3 \dot{q}_2 \\
& - a_1 a_2 m_2 \dot{q}_2 - a_1 a_2 m_3 \dot{q}_2) \sin(q_2) + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) \\
& + (-a_3 b_1 m_3 \dot{q}_2 - a_3 b_1 m_3 \dot{q}_3 - a_1 a_3 m_3 \dot{q}_2 - a_1 a_3 m_3 \dot{q}_3) \sin(q_2 + q_3) \\
C_{44} = & (-a_2 b_1 m_2 \dot{q}_2 - a_1 b_2 m_3 \dot{q}_2 - a_2 b_1 m_3 \dot{q}_2 - b_1 b_2 m_3 \dot{q}_2 \\
& - a_1 a_2 m_2 \dot{q}_2 - a_1 a_2 m_3 \dot{q}_2) \sin(q_2) + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) + (-a_3 b_1 m_3 \dot{q}_2 - a_3 b_1 m_3 \dot{q}_3)
\end{aligned}$$

$$\begin{aligned}
& -a_1 a_3 m_3 \dot{q}_2 - a_1 a_3 m_3 \dot{q}_3) \sin(q_2 + q_3) \\
C_{45} = & (-a_2 b_1 m_2 \dot{q}_0 - a_1 b_2 m_3 \dot{q}_0 - a_2 b_1 m_2 \dot{q}_1 - a_2 b_1 m_3 \dot{q}_0 - a_1 b_2 m_3 \dot{q}_1 - a_2 b_1 m_2 \dot{q}_2 \\
& - a_2 b_1 m_3 \dot{q}_1 - a_1 b_2 m_3 \dot{q}_2 - a_2 b_1 m_3 \dot{q}_2 - b_1 b_2 m_3 \dot{q}_0) \sin(q_2) \\
& + (-b_1 b_2 m_3 \dot{q}_1 - b_1 b_2 m_3 \dot{q}_2 - a_1 a_2 m_2 \dot{q}_0 - a_1 a_2 m_2 \dot{q}_1 - a_1 a_2 m_3 \dot{q}_0 \\
& - a_1 a_2 m_2 \dot{q}_2 - a_1 a_2 m_3 \dot{q}_1 - a_1 a_2 m_3 \dot{q}_2) \sin(q_2) \\
& + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) + (-a_3 b_1 m_3 \dot{q}_0 - a_3 b_1 m_3 \dot{q}_1 \\
& - a_3 b_1 m_3 \dot{q}_2 - a_3 b_1 m_3 \dot{q}_3 - a_1 a_3 m_3 \dot{q}_0 - a_1 a_3 m_3 \dot{q}_1 - a_1 a_3 m_3 \dot{q}_2 - a_1 a_3 m_3 \dot{q}_3) \sin(q_2 + q_3) \\
C_{46} = & (-a_3 m_3 (a_2 + b_2) (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \sin(q_3) \\
& + (-a_3 m_3 (a_1 + b_1) (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \sin(q_2 + q_3) \\
C_{51} = & 0 \\
C_{52} = & 0 \\
C_{53} = & (a_2 b_0 m_2 \dot{q}_0 + a_2 b_0 m_3 \dot{q}_0 + b_0 b_2 m_3 \dot{q}_0) \cos(q_1 - \phi + q_2) + (a_3 b_0 m_3 \dot{q}_0) \cos(q_1 - \phi + q_2 + q_3) \\
& + (a_2 b_1 m_2 \dot{q}_0 + a_1 b_2 m_3 \dot{q}_0 + a_2 b_1 m_2 \dot{q}_1 + a_2 b_1 m_3 \dot{q}_0 + a_1 b_2 m_3 \dot{q}_1 + a_2 b_1 m_3 \dot{q}_1 \\
& + b_1 b_2 m_3 \dot{q}_0 + b_1 b_2 m_3 \dot{q}_1 + a_1 a_2 m_2 \dot{q}_0 + a_1 a_2 m_2 \dot{q}_1 + a_1 a_2 m_3 \dot{q}_0 + a_1 a_2 m_3 \dot{q}_1) \sin(q_2) \\
& + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) + (a_3 b_1 m_3 \dot{q}_0 + a_3 b_1 m_3 \dot{q}_1 \\
& + a_1 a_3 m_3 \dot{q}_0 + a_1 a_3 m_3 \dot{q}_1) \sin(q_2 + q_3) \\
C_{54} = & (a_2 b_1 m_2 \dot{q}_0 + a_1 b_2 m_3 \dot{q}_0 + a_2 b_1 m_2 \dot{q}_1 + a_2 b_1 m_3 \dot{q}_0 + a_1 b_2 m_3 \dot{q}_1 + a_2 b_1 m_3 \dot{q}_1 \\
& + b_1 b_2 m_3 \dot{q}_0 + b_1 b_2 m_3 \dot{q}_1 + a_1 a_2 m_2 \dot{q}_0 + a_1 a_2 m_2 \dot{q}_1 + a_1 a_2 m_3 \dot{q}_0 + a_1 a_2 m_3 \dot{q}_1) \sin(q_2) \\
& + (-a_3 b_2 m_3 \dot{q}_3 - a_2 a_3 m_3 \dot{q}_3) \sin(q_3) + (a_3 b_1 m_3 \dot{q}_0 + a_3 b_1 m_3 \dot{q}_1 + a_1 a_3 m_3 \dot{q}_0 + a_1 a_3 m_3 \dot{q}_1) \\
& \sin(q_2 + q_3) \\
C_{55} = & (-a_3 m_3 \dot{q}_3 (a_2 + b_2)) \sin(q_3)
\end{aligned}$$

$$C_{56} = (-a_3 m_3 (a_2 + b_2) (\dot{q}_0 + \dot{q}_1 + \dot{q}_2 + \dot{q}_3)) \sin(q_3)$$

$$C_{61} = 0$$

$$C_{62} = 0$$

$$C_{63} = (a_3 b_0 m_3 \dot{q}_0) \cos(q_1 - \phi + q_2 + q_3) + (a_3 m_3 (a_2 \dot{q}_0 + a_2 \dot{q}_1 + a_2 \dot{q}_2 + b_2 \dot{q}_0 + b_2 \dot{q}_1 + b_2 \dot{q}_2)) \sin(q_3) + (a_3 m_3 (a_1 \dot{q}_0 + a_1 \dot{q}_1 + b_1 \dot{q}_0 + b_1 \dot{q}_1)) \sin(q_2 + q_3)$$

$$C_{64} = (a_3 m_3 (a_2 \dot{q}_0 + a_2 \dot{q}_1 + a_2 \dot{q}_2 + b_2 \dot{q}_0 + b_2 \dot{q}_1 + b_2 \dot{q}_2)) \sin(q_3) + (a_3 m_3 (a_1 \dot{q}_0 + a_1 \dot{q}_1 + b_1 \dot{q}_0 + b_1 \dot{q}_1)) \sin(q_2 + q_3)$$

$$C_{65} = (a_3 m_3 (a_2 + b_2) (\dot{q}_0 + \dot{q}_1 + \dot{q}_2)) \sin(q_3)$$

$$C_{66} = 0$$

Appendix C

Supplementary Documentation - Chapter #4

Here, the MATLAB code used to generate the optimal solutions found in this thesis will be provided. Note, however, that without the proper licenses and files, it will not be possible to run the scripts provided in this section.

C.1 MATLAB Code - DIDO Optimization

The DIDO optimization code is segmented into four primary pieces of code: the initialization script, the dynamics function, the event function, and the cost function. In the following subsection, each piece of code will be briefly described to provide readers with a basic understanding of the structure.

C.1.1 Initialization Script

The initialization script for DIDO is required to set up the NLP defined in the body of the thesis. This includes all constants, path constraints, control constraints, bounds on initial and final time, and any initial guesses.

```
%=====
% Names: Alexander Crain
% Student ID: 100848550
% Date: 2017-01-22
% This program optimizes the
% trajectory of a FFR
%=====
```

```
% Step #1
clear all; %#ok<CLALL>
close('all');
clc;

% Conversion constants
d2r = pi/180;
r2d = 1/d2r;

phi = 68.2840*d2r; % rad

% Manipulator characteristics
b0 = 0.2304;          % m
a1 = 0.1933;          % m
b1 = 0.1117;          % m
a2 = 0.1993;          % m
b2 = 0.1057;          % m
a3 = 0.0621;          % m
b3 = 0.0159;          % m

L1 = a1 + b1;          % m
L2 = a2 + b2;          % m

m0 = 16.9478; % kg
m1 = 0.3434; % kg
m2 = 0.3334; % kg
m3 = 0.0111; % kg

I0 = 0.2873675;
```

```

I1 = 0.0037502;
I2 = 0.0034132;
I3 = 0.000056402588;

% Step #2: Supply Boundary Conditions

bounds.states          = [-[1; 1; pi;...
                           90*d2r; 90*d2r;...
                           90*d2r; 1; 1; 1;...
                           0.5; 0.5; 0.5],...
                           [1; 1; pi;...
                           90*d2r; 90*d2r;...
                           90*d2r; 1; 1; 1;...
                           0.5; 0.5; 0.5]];

bounds.controls        = [-[0; 0; 0; 0.05;...
                           0.05; 0.05],[0; 0;...
                           0; 0.05; 0.05; 0.05]];

bounds.initial.time    = [0,0];
bounds.final.time      = [30,30];

% Step #3: Provide a Search Space for DID0

search.states          = bounds.states;

search.controls        = bounds.controls;

% Step #4: Set Initial & Final Events

```



```

initialevents          = [0; 0; 0*d2r;...
                          90*d2r; 90*d2r;...
                          0; 0; 0; 0; 0;...
                          0; 0];
finalevents            = [-0.08; 0.7; -0.08;...
                          0; 0; 0; 0;...
                          0; 0];
bounds.events          = [[initialevents,...
                          initialevents];...
                          [finalevents,...
                          finalevents]];

% Step #5: Define the Problem Using DIDO Expressions

FreeArm.cost           = 'FreeArmCost';
FreeArm.dynamics       = 'FreeArmDynamics';
FreeArm.events         = 'FreeArmEvents';
FreeArm.bounds         = bounds;
FreeArm.search         = search;

% Step #6: Check the Problem Formulation

check(FreeArm);

% Step #8: Setup the Algorithm and Call DIDO

attempt                = 1;

switch attempt
    case 1

```

```

tic;
algorithm.nodes      = 30;
[cost, primal, dual] = dido(FreeArm, algorithm);
runTime              = toc;
fprintf('Run time is: %f seconds.\n', runTime);

save NoGuessFreeFloatingArm

case 2

load NoGuessFreeFloatingArm

m0 = 16.9478; % kg
m1 = 0.3434;  % kg
m2 = 0.3334;  % kg
m3 = 0.0111;  % kg

I0 = 0.2873675; % kgm^2
I1 = 0.0037502; % kgm^2
I2 = 0.0034132; % kgm^2
I3 = 0.000056402588; % kgm^2

% Step #2: Supply Boundary Conditions

bounds.states      = [-[1; 1; pi;...
                        90*d2r; 90*d2r;...
                        90*d2r; 1; 1; 1;...
                        0.5; 0.5; 0.5],...
                        [1; 1; pi;...
                        90*d2r; 90*d2r;...

```

```

                                90*d2r; 1; 1; 1;...
                                0.5; 0.5; 0.5]];

bounds.controls                = [-[0; 0; 0; 0.05;...
                                0.05; 0.05],[0; 0;...
                                0; 0.05; 0.05; 0.05]];

bounds.initial.time            = [0,0];
bounds.final.time              = [30,30];

% Step #3: Provide a Search Space for DIDO

search.states                  = bounds.states;

search.controls                = bounds.controls;

% Step #4: Set Initial & Final Events

initialevents                  = [0; 0; 0*d2r;...
                                90*d2r; 90*d2r;...
                                0; 0; 0; 0; 0;...
                                0; 0];
finalevents                    = [-0.08; 0.7; -0.08;...
                                0; 0; 0; 0;...
                                0; 0];

bounds.events                  = [[initialevents,...
                                initialevents];...
                                [finalevents,...
                                finalevents]];

```

```

% Step #5: Define the Problem Using DIDO Expressions

FreeArm.cost                = 'FreeArmCost';
FreeArm.dynamics            = 'FreeArmDynamics';
FreeArm.events              = 'FreeArmEvents';
FreeArm.bounds              = bounds;
FreeArm.search              = search;

guess.states                = primal.states;
guess.controls              = primal.controls;
guess.time                  = primal.time;
algorithm.guess             = guess;

tic;
algorithm.nodes             = 100;
[cost, primal, dual] = dido(FreeArm, algorithm);
runTimeGuess               = toc;
fprintf('Run time is: %f seconds.\n', runTimeGuess);

save UserGuessFreeFloatingArm

end

```

C.1.2 Dynamics Function

The dynamics function defines the state-space vector which dictates the dynamic constraints on the system. Note the *for*-loop in this function. The state vector has a length equal to the number of nodes, and it is not possible to perform the matrix operations unless the state vector is looped through once per node.

```

function xdot = FreeArmDynamics(primal)

    xdot = zeros(size(primal.states));

    [~, n] = size(primal.states);

    for i = 1:n

        q0          = primal.states(3,i);
        q1          = primal.states(4,i);
        q2          = primal.states(5,i);
        q3          = primal.states(6,i);

        q0_dot      = primal.states(9,i);
        q1_dot      = primal.states(10,i);
        q2_dot      = primal.states(11,i);
        q3_dot      = primal.states(12,i);

        M           = InertiaFunc3LINK(q0,q1,q2,q3);
        C           = CoriolisFunc3LINK(q0,q1,q2,q3,...
            q0_dot,q1_dot,q2_dot,q3_dot);

        xdot(1:6,i) = primal.states(7:12,i);
        xdot(7:12,i) = inv{M}*(primal.controls(:,i) -...
            C*primal.states(7:12,i));

    end

end

```

C.1.3 Event Function

The event function is used to define any major initial or end-point constraints on the NLP.

```
function e = FreeArmEvents(primal)

    % Manipulator characteristics

    % Conversion constants
    d2r = pi/180;

    phi = 68.2840*d2r; % rad

    % Manipulator characteristics
    b0 = 0.2304; % m
    a1 = 0.1933; % m
    b1 = 0.1117; % m
    a2 = 0.1993; % m
    b2 = 0.1057; % m
    a3 = 0.0621; % m
    b3 = 0.0159; % m

    x0 = primal.final.states(1,1);
    y0 = primal.final.states(2,1);
    q0 = primal.final.states(3,1);
    q1 = primal.final.states(4,1);
    q2 = primal.final.states(5,1);
    q3 = primal.final.states(6,1);

    xw = x0+(b0)*cos(phi + q0) + (a1+b1)*cos(pi/2 + q0 + q1)...
```

```

+ (a2+b2)*cos(pi/2 + q0 + q1 + q2);

    xe = x0+(b0)*cos(phi + q0) + (a1+b1)*cos(pi/2 + q0 + q1)...
+ (a2+b2)*cos(pi/2 + q0 + q1 + q2) + (a3+b3)*...
cos(pi/2 + q0 + q1 + q2 + q3);

    ye = y0+(b0)*sin(phi + q0) + (a1+b1)*sin(pi/2 + q0 + q1)...
+ (a2+b2)*sin(pi/2 + q0 + q1 + q2) + (a3+b3)*...
sin(pi/2 + q0 + q1 + q2 + q3);

    e    = [primal.initial.states(:,1); xe; ye; xw;...
            primal.final.states(7:12,1)];

end

```

C.1.4 Cost Function

The cost function defines the function or variable which will be minimized. As discussed in Chap. 4, the running cost (Lagrange) is ignored in favour of minimizing only the change in final attitude.

```

function [Mayer, Lagrange] = FreeArmCost(primal)

    %=====
    % Define Mayer Function

    Mayer    = (primal.final.states(3,1)-...
                primal.initial.states(3,1))^2;

    %=====
    %Define Lagrange Function

```

```

        Lagrange = 0;
end

```

C.2 MATLAB Code - GPOPS-I Optimization

The GPOPS-I optimization code is segmented into three primary pieces of code: the initialization script, the dynamics function, and the cost function. In the following subsection, each piece of code will be briefly described to provide readers with a basic understanding of the structure.

C.2.1 Initialization Script

The initialization script for GPOPS-I is required to set up the NLP defined in the body of the thesis. This includes all constants, path constraints, control constraints, bounds on initial and final time, events, and any initial guesses.

```

%=====
% Names: Alexander Crain
% Student ID: 100848550
% Date: 2017-01-17
%=====

%=====
% Initialize GPOPS
clear;
clc;
close('all')
memory
addpath('snopt7')
gpopsInitialize

```



```

%=====
% Define Global Variables

global b0 a1 b1 a2 b2 a3 b3 L1 L2...
    m0 m1 m2 m3 I0 I1 I2 I3 phi

% Conversion constants
d2r = pi/180;

phi = 68.2840*d2r; % rad

% Manipulator characteristics
b0 = 0.2304;          % m
a1 = 0.1933;          % m
b1 = 0.1117;          % m
a2 = 0.1993;          % m
b2 = 0.1057;          % m
a3 = 0.0621;          % m
b3 = 0.0159;          % m

L1 = a1 + b1;          % m
L2 = a2 + b2;          % m
L3 = 0.0301625;        % m

m0 = 16.9478; % kg
m1 = 0.3434; % kg
m2 = 0.3334; % kg
m3 = 0.0111; % kg

```

```

I0 = 0.2873675;
I1 = 0.0037502;
I2 = 0.0034132;
I3 = 0.000056402588;

%=====
% Setup GPOPS

% Initial States
% Lower Limit
qilow = [0;0;0;pi/2;pi/2;0]; % rad
qdot0low = [0;0;0;0;0;0]; % rad/s

% Upper Limit
qiup = [0;0;0;pi/2;pi/2;0]; % rad
qdot0up = [0;0;0;0;0;0]; % rad/s

% Initial State Matrix
x0low = [qilow;qdot0low];
x0up = [qiup;qdot0up];

% Final States
% Lower Limit
qflow = [-1;-1;0;1.1103; -1.4792;0.3689];
qdotflow = [0;0;0;0;0;0];

% Upper Limit
qfup = [1;1;0;1.1103; -1.4792;0.3689];
qdotfup = [0;0;0;0;0;0];

```



```

limits{imin}{iphase,itime} = [t0low tflow];
limits{imax}{iphase,itime} = [t0up tfup];
limits{imin}{iphase,istate} = [x0low xmin xflow];
limits{imax}{iphase,istate} = [x0up xmax xfup];
limits{imin}{iphase,icontrol} = umin;
limits{imax}{iphase,icontrol} = umax;
limits{imin}{iphase,iparameter} = [];
limits{imax}{iphase,iparameter} = [];
limits{imin}{iphase,ipath} = [];
limits{imax}{iphase,ipath} = [];
limits{imin}{iphase,ievent} = [];
limits{imax}{iphase,ievent} = [];

%=====
% Run GPOPS
attempt = 1;
switch attempt
    case 1
        % Initial Estimates - States
        % load matlab
        %
        solinit{iphase,itime} = [0,30]';
        solinit{iphase,istate}(:,1) = [0,-0.0020];
        solinit{iphase,istate}(:,2) = [0,-0.0121];
        solinit{iphase,istate}(:,3) = [0,0];
        solinit{iphase,istate}(:,4) = [1.5708,1.1103];
        solinit{iphase,istate}(:,5) = [1.5708,-1.4792];
        solinit{iphase,istate}(:,6) = [0,0.3689];
        solinit{iphase,istate}(:,7) = [0,0];

```

```

solinit{iphase,istate}(:,8) = [0,0];
solinit{iphase,istate}(:,9) = [0,0];
solinit{iphase,istate}(:,10) = [0,0];
solinit{iphase,istate}(:,11) = [0,0];
solinit{iphase,istate}(:,12) = [0,0];

%Initial Estimates - Parameters
solinit{iphase,iparameter} = [];

%Initial Estimates - Controls
solinit{iphase,icontrol}(:,1) = [0,0];
solinit{iphase,icontrol}(:,2) = [0,0];
solinit{iphase,icontrol}(:,3) = [0,0];
solinit{iphase,icontrol}(:,4) = [-0.0132,0.050];
solinit{iphase,icontrol}(:,5) = [-0.0222,-0.0375];
solinit{iphase,icontrol}(:,6) = [-0.0052,-0.0037];
setup.name = 'FreeFloatingOptimal';
setup.funcs = {'FreeArmCost','FreeArmDae'};
setup.nodes = 30;
setup.limits = limits;
setup.solinit = solinit;
setup.connections = [];
setup.derivatives = 'numerical';
setup.direction = 'increasing';
setup.autoscale = 'off';

output = gpops(setup);
solution = output.solution;
case 2
    % Load old solution

```

```
%load('90NodeSolution_FreeFloating_GPOPS');
```

```
t = solution{1,1};
```

```
x = solution{1,2};
```

```
u = solution{1,3};
```

```
solinit{iphase,itime} = t;
```

```
solinit{iphase,istate}(:,1) = x(:,1);
```

```
solinit{iphase,istate}(:,2) = x(:,2);
```

```
solinit{iphase,istate}(:,3) = x(:,3);
```

```
solinit{iphase,istate}(:,4) = x(:,4);
```

```
solinit{iphase,istate}(:,5) = x(:,5);
```

```
solinit{iphase,istate}(:,6) = x(:,6);
```

```
solinit{iphase,istate}(:,7) = x(:,7);
```

```
solinit{iphase,istate}(:,8) = x(:,8);
```

```
solinit{iphase,istate}(:,9) = x(:,9);
```

```
solinit{iphase,istate}(:,10) = x(:,10);
```

```
solinit{iphase,istate}(:,11) = x(:,11);
```

```
solinit{iphase,istate}(:,12) = x(:,12);
```

```
% Initial Estimates - Parameters
```

```
solinit{iphase,iparameter} = [];
```

```
%Initial Estimates - Controls
```

```
solinit{iphase,icontrol}(:,1) = u(:,1);
```

```
solinit{iphase,icontrol}(:,2) = u(:,2);
```

```
solinit{iphase,icontrol}(:,3) = u(:,3);
```

```
solinit{iphase,icontrol}(:,4) = u(:,4);
```

```
solinit{iphase,icontrol}(:,5) = u(:,5);
```

```
solinit{iphase,icontrol}(:,6) = u(:,6);
```

```

    setup.name = 'FreeFlyingOptimal';
    setup.funcs = {'FreeArmCost','FreeArmDae'};
    setup.nodes = 100;
    setup.limits = limits;
    setup.solinit = solinit;
    setup.connections = [];
    setup.derivatives = 'numerical';
    setup.direction = 'increasing';
    setup.autoscale = 'off';

    output    = gpops(setup);
    solution = output.solution;
end

```

C.2.2 Dynamics Function

The dynamics function defines the state-space vector which dictates the dynamic constraints on the system. Note the *for*-loop in this function. The state vector has a length equal to the number of nodes, and it is not possible to perform the matrix operations unless the state vector is looped through once per node.

```

function [xdot] = FreeArmDae(soldae,~)

%=====
% Call State Estimates & Control Estimates
x      = soldae{2};
u      = soldae{3};

xdot = zeros(size(x));

```

```

[m,~] = size(x);

for i = 1:m

    q0          = x(i,3);
    q1          = x(i,4);
    q2          = x(i,5);
    q3          = x(i,6);

    q0_dot      = x(i,9);
    q1_dot      = x(i,10);
    q2_dot      = x(i,11);
    q3_dot      = x(i,12);

    xdot(i,1:6) = x(i,7:12);

    M           = InertiaFunc3LINK(q0,q1,q2,q3);
    C           = CoriolisFunc3LINK(q0,q1,q2,q3,q0_dot,...
        q1_dot,q2_dot,q3_dot);

    temp = inv(M)*(u(i,:) - C*x(i,7:12)');
    xdot(i,1:6) = x(i,7:12);
    xdot(i,7:12) = temp';

end

end

```


C.2.3 Cost Function

The cost function defines the function or variable which will be minimized. As discussed in Chap. 4, the running cost (Lagrange) is ignored in favour of minimizing only the change in final attitude.

```
function [Mayer,Lagrange] = FreeArmCost(solcost,~)

x = solcost{2,2}; t = solcost{2,1};

%=====
% Define Mayer Function
Mayer = (x(end,3)-x(1,3))^2;

%=====
% Define Lagrange Function
Lagrange = zeros(size(t));

end
```

C.3 MATLAB Code - TOMLAB Optimization

The TOMLAB optimization code is only a single primary piece of code which defines the constraints, bounds, events, dynamics, and cost function for the NLP being solved.

```
clear;
clc;

%=====
```

```

% Names: Alexander Crain
% Student ID: 100848550
% Date: 2017-10-07
%=====

xee=[-0.08 0.7]';
xew=[-0.08 0.7]';

%=====
% Define Global Variables

% Conversion constants
d2r = pi/180;
r2d = 1/d2r;

phi = 68.2840*d2r; % rad

% Manipulator characteristics
b0 = 0.2304;      % m
a1 = 0.1933;      % m
b1 = 0.1117;      % m
a2 = 0.1993;      % m
b2 = 0.1057;      % m
a3 = 0.0621;      % m
b3 = 0.0159;      % m

L1 = a1 + b1;      % m
L2 = a2 + b2;      % m

m0 = 16.9478; % kg
m1 = 0.3434; % kg

```

```

m2 = 0.3334; % kg
m3 = 0.0111; % kg

I0 = 0.2873675;
I1 = 0.0037502;
I2 = 0.0034132;
I3 = 0.000056402588;

%=====
% Initialize TOMLAB

toms t

% Intial & Final conditions
xi    = [0; 0; 0; pi/2; pi/2; 0; 0; 0; 0; 0; 0; 0];

p      = tomPhase('p', t, 0, 30, 100);
setPhase(p);

tomStates x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12
tomControls u1 u2 u3 u4 u5 u6

%load('100NodeSolution_FreeFloating_DIDO');

% Initial Guess
x_ini = {icollocate({x1 == 0;...
                    x2 == 0;...
                    x3 == 0;...
                    x4 == 0;...
                    x5 == 0;...

```

```

        x6 == 0;...
        x7 == 0;...
        x8 == 0;...
        x9 == 0;...
        x10 == 0;...
        x11 == 0;...
        x12 == 0})

collocate({u1 == 0;...
          u2 == 0;...
          u3 == 0;...
          u4 == 0;...
          u5 == 0;...
          u6 == 0}});

% Box Constraints
cbox = {-1 <= collocate(x1) <= 1
        -1 <= collocate(x2) <= 1
        -pi <= collocate(x3) <= pi
        -pi/2 <= collocate(x4) <= pi/2
        -pi/2 <= collocate(x5) <= pi/2
        -pi/2 <= collocate(x6) <= pi/2
        -1 <= collocate(x7) <= 1
        -1 <= collocate(x8) <= 1
        -1 <= collocate(x9) <= 1
        -0.5 <= collocate(x10) <= 0.5
        -0.5 <= collocate(x11) <= 0.5
        -0.5 <= collocate(x12) <= 0.5
        -0 <= collocate(u1) <= 0

```

```

-0 <= collocate(u2) <= 0
-0 <= collocate(u3) <= 0
-0.05 <= collocate(u4) <= 0.05
-0.05 <= collocate(u5) <= 0.05
-0.05 <= collocate(u6) <= 0.05};

% Boundary Constraints
cbnd = {initial({x1 == 0; x2 == 0; x3 == 0;...
               x4 == pi/2; x5 == pi/2; x6 == 0;...
               x7 == 0; x8 == 0; x9 == 0;...
               x10 == 0; x11 == 0; x12 == 0})
       final({x1+(b0)*cos(phi + x3) + ...
              (a1+b1)*cos(pi/2 + x3 + x4)...
              + (a2+b2)*cos(pi/2 + x3 + x4 + x5) == xew(1);...
              x1+(b0)*cos(phi + x3) + (a1+b1)*cos(pi/2 + x3 + x4)...
              + (a2+b2)*cos(pi/2 + x3 + x4 + x5) +...
              (a3+b3)*cos(pi/2 + x3 + x4 + x5+x6)== xee(1);...
              x2+(b0)*sin(phi + x3) + (a1+b1)*sin(pi/2 + x3 + x4)...
              + (a2+b2)*sin(pi/2 + x3 + x4 + x5) +...
              (a3+b3)*sin(pi/2 + x3 + x4 + x5+x6)== xee(2);...
              x7 == 0; x8 == 0; x9 == 0; x10 == 0;...
              x11 == 0; x12 == 0})});

%=====
% Dynamics Model

M          = InertiaFunc3LINK(x3,x4,x5,x6);
C          = CoriolisFunc3LINK(x3,x4,x5,x6,...
                               x9,x10,x11,x12);

```

```
qdot = [x7; x8; x9; x10; x11; x12];
```

```
Tau = [u1; u2; u3; u4; u5; u6];
```

```
qddot = (inv(M)*(Tau - C*qdot));
```

```
%=====
```

```
% Define DAE
```

```
dx1 = x7;
```

```
dx2 = x8;
```

```
dx3 = x9;
```

```
dx4 = x10;
```

```
dx5 = x11;
```

```
dx6 = x12;
```

```
dx7 = qddot(1);
```

```
dx8 = qddot(2);
```

```
dx9 = qddot(3);
```

```
dx10 = qddot(4);
```

```
dx11 = qddot(5);
```

```
dx12 = qddot(6);
```

```
ceq = collocate({
```

```
    dot(x1) == dx1
```

```
    dot(x2) == dx2
```

```
    dot(x3) == dx3
```

```
    dot(x4) == dx4
```

```
    dot(x5) == dx5
```

```
    dot(x6) == dx6
```

```
    dot(x7) == dx7
```

```
    dot(x8) == dx8
```

```

dot(x9) == dx9
dot(x10) == dx10
dot(x11) == dx11
dot(x12) == dx12});

%=====
% Objective Statement

objective          = (x3(end)-x3(1))^2;
options            = struct;
options.name        = 'FreeFloatingArm';
options.solver      = 'snopt';
[solution, result] = ezsolve(objective,{cbox,...
    cbnd, ceq}, x_ini, options);

```

C.4 MATLAB Code - Baseline Controller

To compare the performance of the optimization tools, a baseline controller based on the transpose Jacobian concept was developed in MATLAB-Simulink. The code is divided into two main components: the initialization script and the Simulink diagram.

C.4.1 Initialization Script

This script defines any constants used in the dynamics and kinematics models, as well as the gains required for the transpose Jacobian controller.

```
clc; clear; close all;
```

[illegible]

```
% Constants and user-defined inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Conversion constants
d2r = pi/180;
r2d = 1/d2r;
```

```
phi = 68.2840*d2r; % rad
```

```
% Manipulator characteristics
```

```
b0 = 0.2304;          % m
a1 = 0.1933;          % m
b1 = 0.1117;          % m
a2 = 0.1993;          % m
b2 = 0.1057;          % m
a3 = 0.0621;          % m
b3 = 0.0159;          % m
```

```
L1 = a1 + b1;         % m
L2 = a2 + b2;         % m
L3 = 0.0301625;       % m
```

```
m0 = 16.9478; % kg
m1 = 0.3434; % kg
m2 = 0.3334; % kg
m3 = 0.0111; % kg
```

```
I0 = 0.2873675;
I1 = 0.0037502;
I2 = 0.0034132;
```



```
set_param('ARM_GJM mdl', 'StopTime', 'tsim')  
disp('Running simulation...')  
sim('ARM_GJM mdl')
```

C.4.2 Top-Level Simulink Diagram

Figure C.1: Simulink diagram for transpose Jacobian simulations.

C.5 Pontryagin's Principle - Symbolic Derivation

```
clear;
close('all');
clc;

syms x0 y0 q0 q1 q2 q3 x0_dot y0_dot q0_dot q1_dot q2_dot q3_dot la_x0 la_y0 la_q0...
la_q1 la_q2 la_q3 la_x0d la_y0d la_q0d la_q1d la_q2d la_q3d mu_x0 mu_y0 mu_q0...
mu_q1 mu_q2 mu_q3 mu_x0d mu_y0d mu_q0d mu_q1d mu_q2d mu_q3d mu_fx mu_fy mu_t0...
mu_t1 mu_t2 mu_t3 fx fy t0 t1 t2 t3 v_x0i v_y0i v_q0i v_q1i v_q2i v_q3i v_x0di...
v_y0di v_q0di v_q1di v_q2di v_q3di v_xEEf v_yEEf v_x0df v_y0df v_q0df v_q1df...
v_q2df v_q3df x0i y0i q0i q1i q2i q3i x0_di y0_di q0_di q1_di q2_di q3_di...
x0_df y0_df q0_df q1_df q2_df q3_df t x0f y0f q0f q1f q2f q3f v_xEWf xewf...
v_q0f

% Conversion constants
d2r = pi/180;
r2d = 1/d2r;

phi = 68.2840*d2r; % rad

% Manipulator characteristics
b0 = 0.2304; % m
```

```

a1 = 0.1933;      % m
b1 = 0.1117;      % m
a2 = 0.1993;      % m
b2 = 0.1057;      % m
a3 = 0.0621;      % m
b3 = 0.0159;      % m

```

```

L1 = a1 + b1;      % m
L2 = a2 + b2;      % m
L3 = 0.0301625;    % m

```

```

m0 = 16.9478; % kg
m1 = 0.3434; % kg
m2 = 0.3334; % kg
m3 = 0.0111; % kg

```

```

I0 = 0.2873675;
I1 = 0.0037502;
I2 = 0.0034132;
I3 = 0.000056402588;

```

```

tau      = [fx, fy, t0, t1, t2, t3];
qdot     = [x0_dot, y0_dot, q0_dot, q1_dot, q2_dot, q3_dot];

```

```

fprintf('Loading Symbolic Model...\n')
load('CoriolisSimple');
load('InverseSymbolicSimple');
qdd = invInertiaS*(tau'-CoriolisS*qdot');

x0_dd = (qdd(1));
y0_dd = (qdd(2));
q0_dd = (qdd(3));
q1_dd = (qdd(4));
q2_dd = (qdd(5));
q3_dd = (qdd(6));

%=====
% Define Hamiltonian
fprintf('Defining Hamiltonian...\n')
H    = la_x0*x0_dot + la_y0*y0_dot + la_q0*q0_dot + la_q1*q1_dot + la_q2*q2_dot + la_q3*q3_dot +...
      la_x0d*x0_dd + la_y0d*y0_dd + la_q0d*q0_dd + la_q1d*q1_dd + la_q2d*q2_dd + la_q3d*q3_dd;

f = matlabFunction(H,'File','Hamiltonian');

%=====
% Define the Lagrangian of the Hamiltonian

```

```

fprintf('Defining Lagrangian of Hamiltonian...\n')
H_b = H + mu_x0*x0 + mu_y0*y0 + mu_q0*q0 + mu_q1*q1 + mu_q2*q2 + mu_q3*q3 +...
      mu_x0d*x0_dot + mu_y0d*y0_dot + mu_q0d*q0_dot + mu_q1d*q1_dot + mu_q2d*q2_dot + mu_q3d*q3_dot +...
      mu_fx*fx + mu_fy*fy + mu_t0*t0 + mu_t1*t1 + mu_t2*t2 + mu_t3*t3;

f = matlabFunction(H_b,'File','LagrangianHamiltonian');

%=====
% Define the Endpoint Lagrangian

xef = (x0f + (b0)*cos(phi + q0f)...
      + (a1+b1)*cos(pi/2+q0f+q1f) + (a2+b2)*cos(pi/2+q0f...
      + q1f + q2f)+(a3+b3)*cos(pi/2+q0f...
      + q1f + q2f + q3f));

yef = (y0f + (b0)*sin(phi + q0f)...
      + (a1+b1)*sin(pi/2+q0f+q1f) + (a2+b2)*sin(pi/2+q0f...
      + q1f + q2f)+ (a3+b3)*sin(pi/2+q0f...
      + q1f + q2f + q3f));

v    = [v_x0i
        v_y0i
        v_q0i

```

```

v_q1i
v_q2i
v_q3i
v_x0di
v_y0di
v_q0di
v_q1di
v_q2di
v_q3di
v_xEEf
v_yEEf
v_xEWf
v_x0df
v_y0df
v_q0df
v_q1df
v_q2df
v_q3df];

```

```

x    = [x0i
        y0i
        q0i
        q1i

```



```
q2i
q3i
x0_di
y0_di
q0_di
q1_di
q2_di
q3_di
xef
yef
xewf
x0_df
y0_df
q0_df
q1_df
q2_df
q3_df];
```

```
fprintf('Deriving the Endpoint Lagrangian...\n')
```

```
E_b = transpose(v)*x;
```

```
fprintf('Taking Partial Derivative of E_b wrt Initial States...\n')
```

```

EndLagI =-[ diff(E_b, x0i)
            diff(E_b, y0i)
            diff(E_b, q0i)
            diff(E_b, q1i)
            diff(E_b, q2i)
            diff(E_b, q3i)
            diff(E_b, x0_di)
            diff(E_b, y0_di)
            diff(E_b, q0_di)
            diff(E_b, q1_di)
            diff(E_b, q2_di)
            diff(E_b, q3_di)];

fprintf('Taking Partial Derivative of E_b wrt Final States...\n')
EndLagF = [ diff(E_b, x0f)
            diff(E_b, y0f)
            diff(E_b, q0f)
            diff(E_b, q1f)
            diff(E_b, q2f)
            diff(E_b, q3f)
            diff(E_b, x0_df)
            diff(E_b, y0_df)
            diff(E_b, q0_df)

```

```

        diff(E_b, q1_df)
        diff(E_b, q2_df)
        diff(E_b, q3_df)];

f = matlabFunction(EndLagI,'File','EndLagI');
f = matlabFunction(EndLagF,'File','EndLagF');

%=====
% Hamiltonian Minimization Condition
fprintf('Taking Partial Derivative of H_b wrt Controls...\n')
HamMin = [ diff(H_b, fx)
           diff(H_b, fy)
           diff(H_b, t0)
           diff(H_b, t1)
           diff(H_b, t2)
           diff(H_b, t3)];

fprintf('Saving HamMin...\n')
f = matlabFunction(HamMin,'File','HamMin');

%=====
% Hamiltonian Evolution Equation
fprintf('Taking Partial Derivative of H_b wrt Time...\n')

```

```

HamEve = diff(H_b,t);

fprintf('Saving HamMin...\n')
f = matlabFunction(HamEve,'File','HamEve');

%=====
% Adjoint Condition
fprintf('Taking Partial Derivative of H_b wrt States...\n')
AdjCon = [ diff(H_b, x0)
           diff(H_b, y0)
           diff(H_b, q0)
           diff(H_b, q1)
           diff(H_b, q2)
           diff(H_b, q3)
           diff(H_b, x0_dot)
           diff(H_b, y0_dot)
           diff(H_b, q0_dot)
           diff(H_b, q1_dot)
           diff(H_b, q2_dot)
           diff(H_b, q3_dot) ];

f = matlabFunction(AdjCon,'File','AdjCon');

```

Appendix D

Supplementary Documentation - Chapter #5

Here, only the code of primary interest will be summarized - i.e. the initialization script, disturbance observer functions, and a top-level view of the Simulink diagrams.

D.1 Baseline - Initializer Script

```
clear;
clc;
close('all')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Simulation - Impedance Control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tsim = 10;

% Conversion constants
d2r = pi/180;
r2d = 1/d2r;

phi = 68.2840*d2r; % rad

% Manipulator characteristics
b0 = 0.2304;      % m
a1 = 0.1933;      % m
b1 = 0.1117;      % m
```

```

a2 = 0.1993;          % m
b2 = 0.1057;          % m
a3 = 0.0621;          % m
b3 = 0.0159;          % m

```

```

m0 = 16.9478; % kg
m1 = 0.3434; % kg
m2 = 0.3334; % kg
m3 = 0.0111; % kg

```

```

I0 = 0.2873675;
I1 = 0.0037502;
I2 = 0.0034132;
I3 = 0.000056402588;

```

```

% Properties of BLACK
CONSTANTS_B(1) = 12.33;
CONSTANTS_B(2) = 0.185;

```

```

L1 = a1+b1;
L2 = a2+b2;
L3 = a3+b3;

```

```

q_ini = [-0.1 0 pi/2 1.05272 -1.35985 0.30713];
q_ini_d= [0 0 0 0 0 0];

```

```

q_dot_ini = [0 0 0 0 0 0];

```

```

Kp = 500*eye(2,2);
Kd = 500*eye(2,2);

```

```

Mt      = 25;
Dt      = 50;
Kt      = 0.01;

        set_param('ImpedanceController_Angel', 'StopTime', 'tsim')
disp('Running simulation (Impedance Controller)...')

tic;
sim('ImpedanceController_Angel')
toc;

disp('Simulation complete! Comparing results...')

PlotResults_Angel;

```

D.2 Baseline - Disturbance Observer Function

```

function TotalForce = BaselineDO(q, q_dot, q_ddot)
TotalForces = zeros(6,1);
%=====
% Define States

q0 = q(3);          q1 = q(4);          q2 = q(5);          q3 = q(6);

q0_dot = q_dot(3);   q1_dot = q_dot(4);
q2_dot = q_dot(5);   q3_dot = q_dot(6);

%=====
% Define Inertia and Coriolis Matrix Components

```

```

M = InertiaFunc3LINK(q0,q1,q2,q3);

C = CoriolisFunc3LINK(q0,q1,q2,q3,q0_dot,q1_dot,...
q2_dot,q3_dot);

%=====
% Disturbance Observer

TotalForce = M*q_ddot + C*q_dot;

end

```


D.3 Baseline - Top-Level Simulink Diagram

Figure D.1: Simulink diagram for baseline DO simulations.

D.4 NDO - Initializer Script

```

clear;
clc;
close('all')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulation - Impedance Control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tsim = 10;

% Conversion constants
d2r = pi/180;
r2d = 1/d2r;

phi = 68.2840*d2r; % rad

% Manipulator characteristics
b0 = 0.2304;      % m
a1 = 0.1933;      % m
b1 = 0.1117;      % m
a2 = 0.1993;      % m
b2 = 0.1057;      % m
a3 = 0.0621;      % m
b3 = 0.0159;      % m

m0 = 16.9478; % kg
m1 = 0.3434; % kg
m2 = 0.3334; % kg
m3 = 0.0111; % kg

```

```

I0 = 0.2873675;
I1 = 0.0037502;
I2 = 0.0034132;
I3 = 0.000056402588;

% Properties of BLACK
CONSTANTS_B(1) = 12.33;
CONSTANTS_B(2) = 0.185;

L1 = a1+b1;
L2 = a2+b2;
L3 = a3+b3;

q_ini = [-0.1 0 pi/2 1.05272 -1.35985 0.30713];
q_ini_d= [0 0 0 0 0 0];

q_dot_ini = [0 0 0 0 0 0];

Kp = 500*eye(2,2);
Kd = 500*eye(2,2);

Mt = 25;
Dt = 50;
Kt = 0.01;

c = 2;

set_param('ImpedanceController_Alex', 'StopTime', 'tsim')
disp('Running simulation (Impedance Controller)...')

```

```
tic;
sim('ImpedanceController_Alex')
toc;

disp('Simulation complete! Comparing results...')
```

```
PlotResults_Alex;
```

D.5 NDO - Disturbance Observer Function

```
function [z_dot, P_gain] = NDO(z, q, q_dot, T, c)
%=====
% Define States

q0 = q(3);          q1 = q(4);          q2 = q(5);          q3 = q(6);

q0_dot = q_dot(3);   q1_dot = q_dot(4);
q2_dot = q_dot(5);   q3_dot = q_dot(6);

%=====
% Define Inertia and Coriolis Matrix Components

M = InertiaFunc3LINK(q0,q1,q2,q3);

C = CoriolisFunc3LINK(q0,q1,q2,q3,q0_dot,q1_dot,...
q2_dot,q3_dot);

%=====
% Nonlinear Disturbance Observer
```

```

P_gain = c*[ q_dot(1)
              q_dot(1)+q_dot(2)
              q_dot(1)+q_dot(2)+q_dot(3)
              q_dot(1)+q_dot(2)+q_dot(3)+q_dot(4)
              q_dot(1)+q_dot(2)+q_dot(3)+q_dot(4)...
              +q_dot(5)
              q_dot(1)+q_dot(2)+q_dot(3)+q_dot(4)...
              +q_dot(5)+q_dot(6)];

L_gain = c*tril(ones(6,6))*M^(-1);

z_dot  = -L_gain*z + L_gain*(C*q_dot - T - P_gain);

end

```

D.6 NDO - Top-Level Simulink Diagram

Figure D.2: Simulink diagram for NDO simulations.

D.7 Generalized Jacobian - Force Estimation

```

function F    = CalculateContactForce(DistT, q, CONSTANTS)
%#codegen

q0    = q(3);
q1    = q(4);
q2    = q(5);
q3    = q(6);

% Mass properties
m0    = CONSTANTS(1);
m1    = CONSTANTS(2);
m2    = CONSTANTS(3);
m3    = CONSTANTS(4);
I0    = CONSTANTS(5);
I1    = CONSTANTS(6);
I2    = CONSTANTS(7);
I3    = CONSTANTS(8);

w      = m0+m1+m2+m3;

% Geometric properties
b0    = CONSTANTS(9);
b1    = CONSTANTS(10);
b2    = CONSTANTS(11);
b3    = CONSTANTS(12);
a0    = CONSTANTS(13);
a1    = CONSTANTS(14);
a2    = CONSTANTS(15);

```

```

a3      = CONSTANTS(16);
phi     = CONSTANTS(17);

l0      = b0;
l1      = a1+b1;
l2      = a2+b2;
l3      = a3+b3;

% Contants
K0      = m0*b0/w;
K1      = (m0*l1 + m1*b1)/w;
K2      = ((m0+m1)*l2 + m2*b2)/w;
K3      = ((m0+m1+m2)*l3 + m3*b3)/w;

% Trigonometric simplifications
s0      = sin(phi+q0);
s1      = sin(pi/2+q0+q1);
s2      = sin(pi/2+q0+q1+q2);
s3      = sin(pi/2+q0+q1+q2+q3);
c0      = cos(phi+q0);
c1      = cos(pi/2+q0+q1);
c2      = cos(pi/2+q0+q1+q2);
c3      = cos(pi/2+q0+q1+q2+q3);

% Jacobian for the base (specific)
Jm      = [ -K1*s1-K2*s2-K3*s3  -K2*s2-K3*s3  -K3*s3
             K1*c1+K2*c2+K3*c3  K2*c2+K3*c3  K3*c3 ];

% Jacobian for the manipulator (specific)
Js      = [ 1 0 -K0*b0*s0-K1*l1*s1-K2*l2*s2-K3*l3*s3

```



```

0 1 K0*b0*c0+K1*l1*c1+K2*l2*c2+K3*l3*c3 ];

% More constants
H      = InertiaFunc3LINK(q0,q1,q2,q3);

Mb      = H(1:3,1:3);
Mbm     = H(1:3,4:6);

% Calculate generalized Jacobian
J      = Jm - Js*(Mb\Mbm);
T      = DistT(4:6);

% Calculate the force
F      = (transpose(J)\T);

end

```