# Inertia Compensation of a Planar Robot for Human Upper Limb Interaction

by

Jessie Thorup

B.S., Tufts University (2012)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018



Author .....
Department of Mechanical Engineering
May 18, 2018

Certified by .....
Neville Hogan
Sun Jae Professor of Mechanical Engineering
Thesis Supervisor

Accepted by .....
Rohan Abeyaratne
Professor of Mechanical Engineering Graduate Officer

# Inertia Compensation of a Planar Robot for Human Upper Limb Interaction

by

Jessie Thorup

Submitted to the Department of Mechanical Engineering
on May 18, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

## Abstract

This thesis documents the development of software and a control system for the InMotion2 planar robot. Software was developed to provide sensor input processing from the robot encoders and force/torque transducer, and output processing for the robot motors. A controller scheme was developed to compensate for the natural robot configuration-based inertia as well as friction and un-modeled dynamics. The inertia compensator was designed using an inertial admittance model and a nonlinear robust adaptive tracking controller based on sliding mode control. A hybrid control mode was developed in which impedance control was used to enforce a virtual constraint and inertia compensation acted along the constraint. The controller proved to be stable throughout testing and provide the desired inertia. The accuracy of the inertia compensation was within human perception limits for modest inertia references.

Thesis Supervisor: Neville Hogan
Title: Sun Jae Professor of Mechanical Engineering

# Acknowledgments

I would like to thank Neville Hogan for accepting me into the Newman Lab and helping me find a project that aligns with my interests. Neville has been an inspiring teacher and a patient advisor. In the Newman Lab I have been able to expand my understanding about control systems beyond aircraft engines and gained new perspective. Joe Davidson and Meghan Huber have both provided invaluable assistance with the inertia compensator project and getting the InMotion2 robot up and running.

I would like to thank Paul LoPiccolo, Tom Martyn, and Paul Patoulidis for giving me the opportunity to attend MIT and allowing me travel to school during work hours each week. In doing so I have gotten hands-on experience with controls hardware and system design that I will be able to use for the rest of my career.

I would also like to thank Chris for going through this experience with me, and Mikey for being there every step of the way – I couldn't have done it without you.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Human beings can move with astounding agility, accuracy, and coordination when considering the physical sensory and motor mechanisms with which we are equipped [1]. The scientific community has a somewhat limited understanding of the control mechanisms that the human neurological system employs, especially when considering constrained motion or interaction. A better understanding of how humans perform this sort of motion would facilitate the development of machines used in human-robot interaction, robots used for contact or manipulation tasks, and strategies for neuromotor rehabilitation.

Insight into human motor control strategies can be gained from studies in which humans perform very specific movement tasks while fixing as many extraneous variables as possible. An interactive robot is a useful tool for such studies, as its response can be programmed to respond to the human in whatever manner is desirable. Constrained motion can be simulated by programming a virtual constraint whereby any movement of the robot's end-effector orthogonal to the constraint results in the robot forcing itself back towards the constraint. This simple control scheme, while effective at enforcing constrained motion, does not fix extraneous variables introduced by the

robot such as configuration-dependent friction or inertia. This thesis aims to eliminate these extraneous variables so that human neuro-motor control studies can be performed effectively.

## 1.1 Background

### 1.1.1 History of the InMotion2

The MIT-MANUS was a robot developed between 1989 and 1991 in the MIT Newman Laboratory for Biomechanics and Human Rehabilitation for the use of robotic physical therapy. The design was documented by Jain Charnnarong [2]. It consisted of two direct-drive brushless DC motors attached to a 5-bar linkage system with a handle at the end-effector. The robot moved in the horizontal plane at a height that was comfortably reached by a sitting person. The robot was designed for patients with motor impairment to grasp the handle and perform reaching exercises with the help of the robot actuation. The back-drivability of the direct-drive motors allowed for the end-effector to easily move around the workspace without the assistance of the motors [2]. The MIT-MANUS was tested in a clinical environment and proved to be an effective tool for neuro-rehabilitation by Hermano Igo Krebs and Neville Hogan [3, 4], so a lower cost beta-prototype was developed by Debo Adebiyi in 1998 which included newer actuators and position sensors [5]. The beta-prototype was characterized in 1999 by Craig Foster and an adaptive tracking controller was developed for the robot which uses position control to move the robotic arm about the workspace [6].

The company Interactive Motion Technologies adapted the MIT-MANUS and developed the InMotion ARM [7]. Interactive Motion Technologies was acquired by Bionik Laboratories in 2016. The company's InMotion2 therapy robot is in currently in use in the Newman Lab at MIT, shown in Figure 1-1. As with the original MIT-


Figure 1-1: InMotion2 Robotic Arm

MANUS design, the InMotion2 robot is powered by two direct-drive brushless DC motors. The robot's position is measured with an encoder mounted on each motor. A force transducer mounted on the end-effector measures the externally applied force, and a handle is mounted on the force transducer which is free to rotate about the vertical axis. If a person were to sit facing the robot, they could comfortably grip the handle and push the robot end-effector about the workspace in the $x$ (left-right) and $y$ (front-back) directions, and the handle will freely rotate in order to allow the person's grip to take on the desired orientation.

### 1.1.2 The Need for Inertia Compensation

The InMotion2 and other similar robotic systems have been widely used for both clinical rehabilitation and studies of human motor control. These robots are typically highly back-drivable so that they can easily move about the workspace. However even though the dynamic effects of the motors themselves are reduced, the joint-linkage system provides differing dynamic characteristics in the Cartesian plane. For instance a 10 cm movement in the x direction of the workspace might cause minimal movement

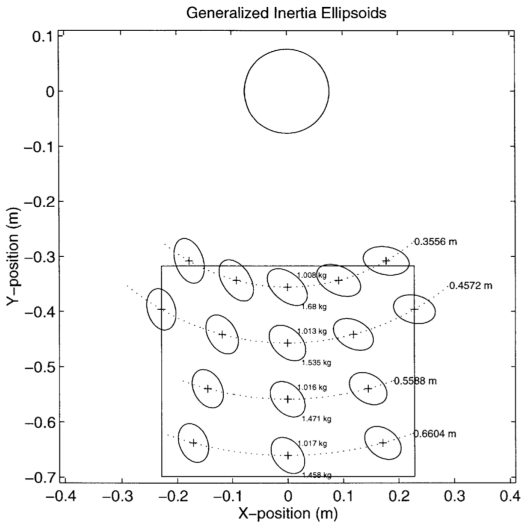



Figure 1-2: Inertial Ellipsoids of MIT-MANUS beta prototype

of the robot joints joints and the robot inertia would feel less heavy. An equivalent 10 cm movement in the y direction of the workspace could cause all of the joints to move through a considerable angle and the robot would feel more heavy. Craig Foster characterized this configuration-dependent inertia for the beta prototype of the MIT-MANUS as ellipsoids in Cartesian space, shown in Figure 1-2 [6].

For many applications the inherent back-drivability of such a robot provides sufficient characteristics for rehabilitation or human study purposes. However a need for a more constant inertia profile exists. In the MIT Newman Lab, Joe Doeringer studied variables that affect intermittency in constrained upper-limb movement for his PhD thesis in 1999 [8]. He performed human crank-turning experiments and examined the position and velocity data in an attempt to isolate the cause of intermittency in human constrained motion. Such an experiment required constant dynamics across the workspace because transiently changing apparent inertia could cause the appearance of intermittency whether or not it was produced by the human. For this reason the experiments were performed on a physical crank which lacks the nonlinear dynamics introduced by the robot. However in his thesis Doeringer points out that the physical

crank limited the flexibility of the experiments, as a robot would have the flexibility to relax away the the constraint, change the constraint shape transiently, or to subtract off the inertia of a subject's limb [8]. However the nonlinear dynamics of the robot rendered these types of experiments impossible, as inertia compensation controllers did not exist at the time. In 2016, Brian Wilcox performed similar studies on the InMotion2 robot using a simple virtual constraint impedance controller to simulate constrained motion on the robot and compared the results to Joe Doeringer's crank turning study. Wilcox found that the velocity profiles from the robot studies, especially for the faster turning experiments, did not line up well with Joe Doeringer's crank turning results. He theorized that this could be due to the nonlinear inertia profile introduced by the robot [9]. In 2017 Ryan Koeppen analyzed constrained motion studies on the InMotion2 robot but confined the experiment to extremely slow movements to simulate quasi-static conditions [10]. Studies of faster constrained motion would have been infeasible in part due to the configuration-dependent dynamics of the robot.

This issue has also been noted outside of the Newman Lab. In 2007 Mahvash and Okamura noted that dynamic properties of a surgical teleoperator including robot inertia and friction impaired the performance of surgical operation tasks by masking haptic forces and reducing the ability of the operator to modulate position. They designed a position-exchange controller that used a feedforward term that attempted to cancel out system dynamics with 70% accuracy [11]. In 2009 Gil et al. devised a linear controller that used force feedback to decrease the apparent inertia of a haptic device and compensate for the dynamics of the operator's arm [12]. In 2011 Aguirre-Ollinger et al. designed a lower limb exoskeleton to aid in walking but noted that the heaviness of the exoskeleton caused fatigue and reduced gait frequency. They designed an inertia compensator that used a feedback loop on acceleration to coun-

teract the inertia of the exoskeleton [13]. In 2012 Colonnese and Okamura designed an inertia compensator to characterize the effect that the cerebellum has on motor control in human subjects with ataxia [14]. In 2017 Erwin et al. performed a study analyzing the effect of exoskeleton robot dynamics on smoothness in wrist pointing tasks. They found that the increased inertia of the flexion-extension degree of freedom facilitated the smoothness of the wrist movements compared with radial/ulnar deviation. Erwin noted that inertia compensators are difficult to implement and can become non-passive [15].

The implementation of a robot control system that compensates for the dynamics of the robot to create the haptic illusion of a point mass at the end-effector would be useful in all of the applications listed above, and be invaluable way to push forward the study of human motor control.

## 1.2 Robot Control Strategies

### 1.2.1 Interaction Control Stability

The InMotion2 robot, when in operation, is a two-part system; the robot which should exhibit predictable behavior within certain bounds of unpredictability, and the human, which for all intents and purposes can be considered an unpredictable part of the environment. A system with mechanical admittance responds to imposed forces with motion; a system with mechanical impedance responds to imposed motion with forces. For two systems to interact with each other, one system must behave with mechanical impedance and the other must behave with mechanical admittance [16]. In 1989 Ed Colgate and Neville Hogan explored criteria for coupled stability of linear time-invariant systems. They determined that for a system to be stable when connected to an arbitrary passive environment, it must have a positive real impedance

or admittance which can be represented by a passive physical equivalent system [17]. If force feedback control is used, programming the apparent mass to less than half of the natural inertia of the system as seen by the end-effector will result in non-passivity [18]. If the admittance of the controller is less than the natural admittance of the end-effector, i.e. the programmed apparent mass of the end-effector is larger than the mass of the end effector, the system can achieve passivity [19]. However Buerger and Hogan argue that passivity is too strict a criteria for coupled stability when the dynamics of the environment are not completely unknown. For robots that are known to interact with humans, the control designer can make assumptions about the bounds of human dynamics and thereby design a controller in which the whole system experiences coupled stability [20].

### 1.2.2 Survey of Inertia Compensation Strategies

There are a number of different ways to control the apparent inertia seen at the end-effector of a robot, and the controller type should depend on the goal of the inertia compensation.

#### Feedforward Force Scaling

In 2009, Gil et al. [12] desired reduced inertia for their robotic systems but did not require that their systems take on a specific inertia. Their robotic system (LHIfAM), shown in Figure 1-3 was modeled as a linear plant $G(s)$. Their system measured the force at the end-effector $F_h$ and subtracted off the dynamics of the operator's arm $Z_h(s)$, then used scalar gain $K_f$ to provide a feed-forward force controller to reduce the apparent inertia of the system.

This approach to controlling inertia with a scalar gain $K_f = 2$ was successful at in-

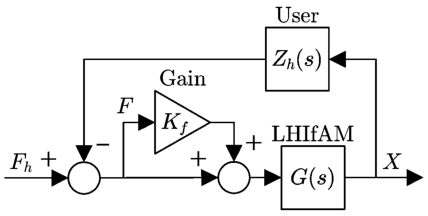



Figure 1-3: Gil et al. System Model [12]

creasing the admittance of the robot by 9.5 dB. This controller was extended to the simulation of a virtual contact in which was modeled by stiffness $K$, shown in Figure 1-4. The filters shown were implemented to limit the noise of the force measurement signal. This strategy, which simply amplified the force applied to the robot, was effec-

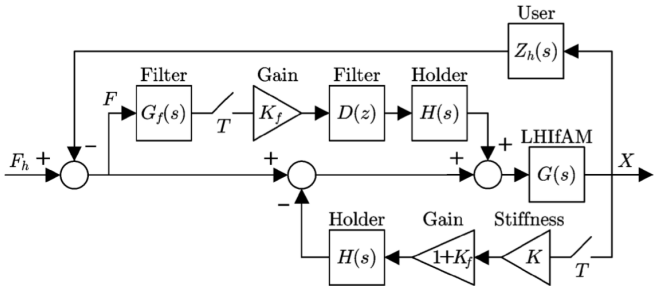



Figure 1-4: Gil et al. System Model with Virtual Contact [12]

tive for reducing the apparent inertia of a linear system for which natural acceleration is naturally proportional applied force. Actuating the system with a scalar multiple of the sensed force transformed the natural response of the system from $F = ma$ to $F(1+K_f) = ma$, scaling down the apparent inertia from $m$ to $\frac{m}{1+K_f}$. However a more sophisticated approach must be taken with a nonlinear system such as the InMotion2 robot, which does not have a natural response of $F = ma$. Simply adding a force scalar without regard to the dynamics of the system could decrease the apparent inertia but would also increase the system's nonlinearities.

### Feedforward Inertia Canceling

In 2007, Mahvash and Okamura [11] created a controller ($f_c$) that modeled the linear friction ($f_f$), gravity ($g$), inertia ($m$), and damping ($b$), of their tele-robotic system based on measured position ($x_m$) at a certain configuration and directly canceled the effect of these forces using a feedforward term. They then used proportional control to achieve desired position ($x_d$) of the controller with proportional gain $k$ [11].

$$f_c = \hat{f}_f + \hat{g} + \hat{m}\ddot{x_m} + \hat{b}\dot{x_m} + k(x_m - x_d) \tag{1.1}$$

This strategy eliminated the need for measuring force feedback but depended on an accurate model and exact knowledge of the system parameters. Additionally, the controller only compensated inertia while the system was moving, as static friction cannot be estimated without a force measurement. It is effective for reducing operator workload but such a strategy cannot completely compensate for the dynamics of the system in the presence of modeling errors, nor does it force the robot to respond with a specified inertia characteristic. Additionally, even though local stability could be guaranteed for a linearized system, that gives no guarantee of global stability.

### Virtual Mass through Impedance Control

In 2012 Colonnese et al. designed an inertia compensator rendering the desired mass as an impedance rather than an admittance [14]. Their system model is shown in Figure 1-5. The top half of the figure denotes how the the human operator was modeled as an impedance controller that output a force $F_h(s)$ according to the difference between a desired trajectory $X_h(s)$ and actual trajectory $X(s)$. The bottom half of the figure denotes the robot controller. The system calculated the acceleration of the end-effector by taking the 2nd derivative of measured position ($X(s)$) and filtering for noise. The acceleration was multiplied by the desired virtual mass $M$ to calculate

desired force. They then used linear stability analysis and numerical simulation to

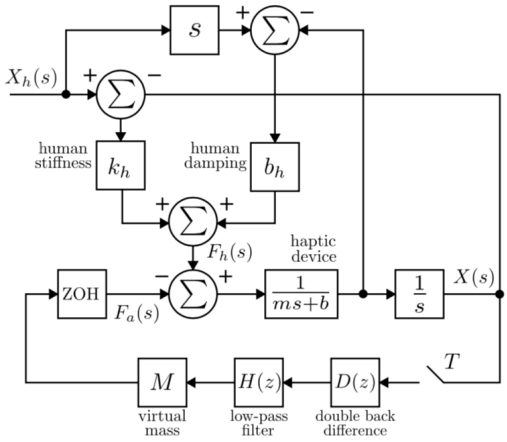



Figure 1-5: Colonnese et al. Virtual Mass Impedance Controller [14]

assess the bands of stability for the system [14]. Because this control strategy used measured feedback to control the system it could have had the potential for more accuracy than the feedforward strategies, but without a feedforward term the controller had to wait for the system reaction, adding extra latency. This latency was extended by the filtering of the acceleration signal which is necessary with a double differentiated signal. This added latency reduced the stability and accuracy of the controller compared with methods that don't rely on pure acceleration feedback.

### Admittance Control with Emulated Inertia

In 2011, Aguirre-Ollinger et al. designed an inertia compensator for a lower limb exoskeleton. The baseline compensator is shown in Figure 1-6. The control system sensed the torque $\tau_s$ which was a combination of the human applied torque $\tau_h$ and the reaction forces from the movement of the leg $I_h$ and exoskeleton $I_{arm}$. This sensed torque signal was then passed through an admittance controller $\bar{Y_e}^d$ which used a simple desired inertia $\bar{I_e}^d$ to command an acceleration which was then integrated to form a desired velocity $\Omega_{ref}$. A simple proportional controller with gain $k_p$ was implemented to modulate motor torque $\tau_m$ to achieve the desired velocity at the motor $w_m$

[13]. This is a reasonable technique for achieving a desired inertia; however they found

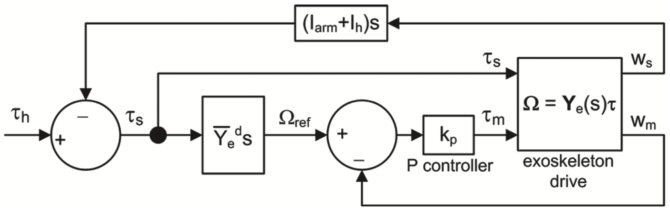



Figure 1-6: Aguirre-Ollinger et al. Admittance Controller [13]

that linear stability analysis limited the desired inertia $\bar{I}_e^{\,d}$ to be at least the value of the actual motor inertia $I_m$. This was undesirable as the goal of the compensation was to give the exoskeleton negative inertia, which would reduce the overall inertia felt by the human's leg. Therefore an additional control component was added to the system in the form of an emulated inertia, shown in the lower path of Figure 1-7. The

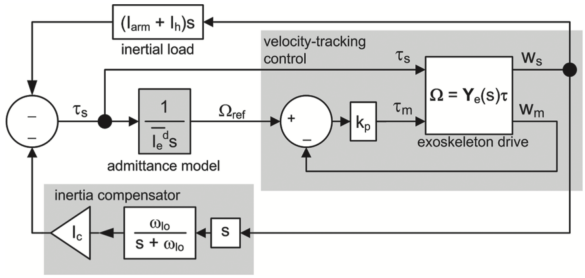



Figure 1-7: Aguirre-Ollinger et al. Emulated Inertia [13]

sensed velocity of the exoskeleton was differentiated and filtered for noise to produce a low-bandwidth acceleration signal. They found that the filtered acceleration measurement could be multiplied by a negative emulated inertia $I_c$ to augment the sensed torque in a positive feedback scheme which amplified the admittance of the system. This created a system which was non-passive, but as Buerger and Hogan pointed out in [20], passivity can be too strict a criteria for coupled stability for human interaction. Experimental testing proved that the system was stable when coupled with a human leg even though it was non-passive, and the controller achieved the desired result of increasing leg swing frequency [13]. However it should be noted that the

goal of this inertia compensator was not to provide constant apparent inertia but to decrease the workload of the human. The acceleration signal needed to be heavily filtered after differentiation which reduced the accuracy of transient compensation.

## 1.3 Overview of Thesis

The objective of this study is to adapt the previous work on inertia compensation presented in Section 1.2.2 using nonlinear control techniques to create an inertia compensation scheme for the InMotion2 Robot. The objective of this controller is stricter than those in Section 1.2.2, which is not just to reduce the apparent inertia of a robot but to make it constant across the workspace in order that the inertia of the apparatus need not be a variable in human studies.

Chapter 2 of this thesis provides an overall description of the InMotion2 Robot including the updates that were made to the sensor and actuator hardware, as well as the derivation of a dynamic model of the InMotion2. Chapter 3 describes the new software developed for the InMotion2 including input and output processing and the simple virtual constraint impedance controller. Chapter 4 derives a robust adaptive tracking inertia compensator and provides proof of nonlinear stability for both the free movement case and the virtual constraint case. Chapter 5 provides an overview of testing and controller validation, and Chapter 6 concludes the thesis with a summary and an exploration of topics that need future work.

# Chapter 2

# The InMotion2 Upper Limb Robot

The InMotion2 Upper Limb rehabilitation robot was developed by Interactive Motion Technologies (now Bionik Laboratories) for rehabilitation of patients recovering from brain injury that results in loss of motor control [7]. The original prototype was designed in the MIT Newman Laboratory as the MIT-MANUS and was adapted by Interactive Motion Technologies for commercial use. The robot used for this study, shown in Figure 1-1, is the InMotion2 with hardware manufactured by Interactive Motion Technologies, but the electrical interface to the robot and controller software have been completely replaced as part of the effort to modernize the control scheme and provide inertia compensation. This chapter will provide an overview of the hardware portion of the new electrical interface as well as a dynamic model derived specifically for the InMotion2 robot. The software portion of the new electrical interface and control system will be described in Chapter 3. A set of step-by-step instructions for using the InMotion2 robot is in Appendix A.

## 2.1 Hardware

The InMotion2 robot assembly consists of four rigid links assembled to move in two degrees of freedom in a horizontal plane. Each of the two degrees of freedom is

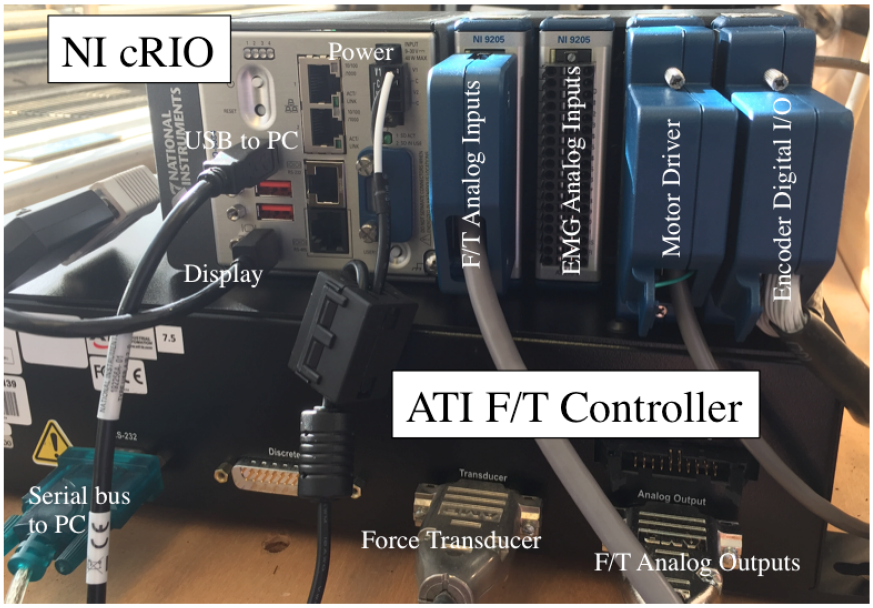



Figure 2-1: National Instruments compactRIO

controlled by a large direct drive motor. The state of the robot is measured by an encoder on each motor as well as a force/torque transducer at the robot end-effector. Each of these sensors and actuators feeds to an input/output module on the National Instruments compactRIO 9034 electronic controller [21], shown in Figure 2-1. The low-level functions are performed on the compactRIO FPGA and the high-level control is performed on the compactRIO real-time processor. The compactRIO optionally connects to a Windows PC for real-time data viewing to provide live input to the controller. A diagram of the system architecture is displayed in Figure 2-2.

### 2.1.1 Motor Servo Drivers

The two motors provide output torque proportional to the current used to drive the motor.

$$\tau = K_t * i \tag{2.1}$$

The motors are controlled using Kollmorgen Servostar CD servo drives which contain high bandwidth closed loop current control [22]. The reference for the control is fed

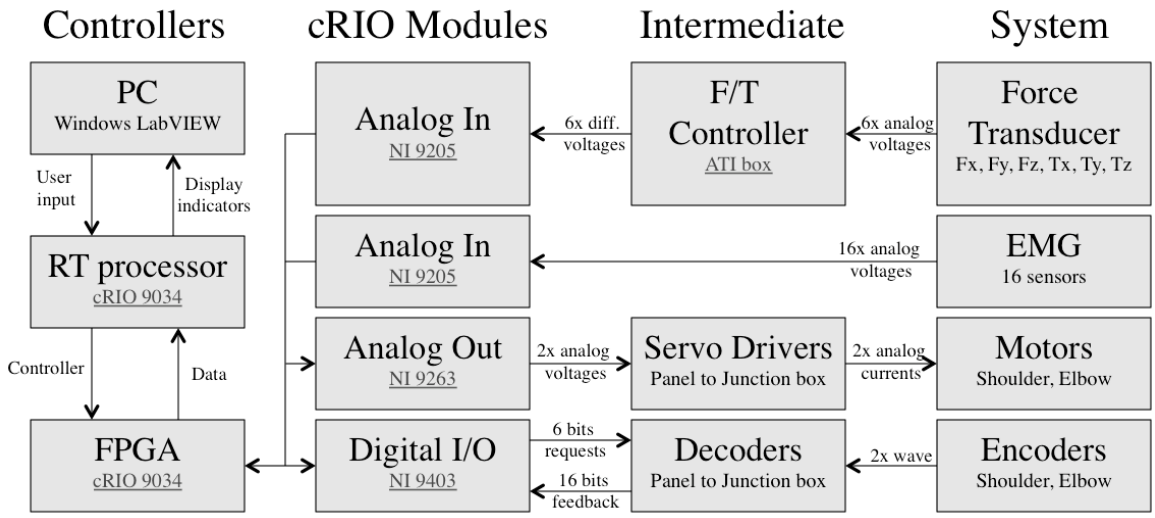



Figure 2-2: System Architecture

into each servo drive as a voltage input, where the voltage is directly proportional to the current provided to each motor. The voltage is controlled using the NI 9263 analog output module on the compactRIO [23], and this voltage is set by the FPGA firmware as directly proportional to the desired motor torque. The motor output pin diagram is shown in Table 2.1.

Table 2.1: Motor Control Pin Diagram

| **Signal** | **Conversion** | **Signal (+)** | | | **Return (-)** | | |
|---|---|---|---|---|---|---|---|
| | | **cRIO Wire Color** | **cRIO Pin** | **Panel Wire Color** | **cRIO Wire Color** | **cRIO Pin** | **Panel Wire Color** |
| Shoulder | 5V=28.8 Nm | White | 2 (AO1) | Blue | Black | 3 (COM) | Light Blue |
| Elbow | 5V=-28.8 Nm | Red | 0 (AO0) | Orange | Brown | 1 (COM) | Pink |

### 2.1.2 Encoders

The position of the robot is sensed by a Gurley Precision Virtual Absolute VB encoder mounted to each motor. Each encoder has a resolution of 16 bits over 360° of revolution. The encoder signal feeds directly to a Gurley Precision Inerpolating Decoder which decodes the analog encoder signal into a digital number [24]. The NI 9403 digital I/O module [25] is used to query the decoder and retrieve the position data. The digital position can be queried using a series of digital signals over two lines, A0 and A1, and returns data over 8 lines, B0-7. Each decoder splits the 16-bit encoder position into three segments: the low bit, the 8 middle bits, and the 7 high bits. The decoder will return a segment of data over B0-7 depending on the status of A0 and A1, detailed in Table 2.2. The decoder pin diagram is detailed in Table 2.3.

Table 2.2: Encoder Request Codes

| **Request** | **A0=0** | **A0=1** |
|---|---|---|
| **A1=0** | Return Nothing | Return Low bit on B7 |
| **A1=1** | Return Middle Bits on B0-7 | Return High bits on B0-6 |

Table 2.3: Decoder Pin Diagram

| **Signal** | **Pin Type** | **GND** | **Request Bits** | | | **Return Bits** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **Hold** | **A0** | **A1** | **B7** | **B6** | **B5** | **B4** | **B3** | **B2** | **B1** | **B0** |
| Shoulder | J4 Ribbon | 33 DGND | 14 DOUT8 | 16 DOUT9 | 18 DOUT10 | 25 DIN15 | 23 DIN14 | 21 DIN13 | 19 DIN12 | 17 DIN11 | 15 DIN10 | 13 DIN9 | 11 DIN8 |
| | NI 9403 pin | 28 COM | 35 DIO29 | 34 DIO28 | 37 DIO31 | 32 DIO26 | 31 DIO25 | 30 DIO24 | 18 DIO15 | 15 DIO12 | 16 DIO13 | 17 DIO14 | 36 DIO30 |
| Elbow | J2 Ribbon | 33 DGND | 14 DOUT0 | 16 DOUT1 | 18 DOUT2 | 25 DIN7 | 23 DIN6 | 21 DIN5 | 19 DIN4 | 17 DIN3 | 15 DIN2 | 13 DIN1 | 11 DIN0 |
| | NI 9403 pin | 9 COM | 26 DIO22 | 1 DIO0 | 2 DIO1 | 24 DIO20 | 23 DIO19 | 22 DIO18 | 21 DIO17 | 20 DIO16 | 7 DIO6 | 8 DIO7 | 27 DIO23 |

### 2.1.3 Force/Torque Transducer

The force and torque applied at the robot end-effector is measured by an ATI Gamma type Force/Torque transducer. This transducer takes six measurements: forces in the

x, y, and z directions, and torques in the roll, pitch, and yaw directions. These signals are processed in the ATI F/T controller which includes a calibration matrix [26]. The F/T controller sends each of the six measurements as a differential voltage to the NI 9205 analog input module of the compactRIO [27]. The F/T controller additionally sends a health bit to the NI 9403 digital I/O module of the compactRIO [25] which indicates whether the signals can be trusted. The force/torque transducer pin diagram is detailed in Table 2.4.

Table 2.4: Force/Torque Pin Diagram

| **Signal** | **Conversion** | **Signal (+)** | | | **Return (-)** | | |
|---|---|---|---|---|---|---|---|
| | | **Color** | **ATI pin** | **cRIO pin** | **Color** | **ATI pin** | **cRIO pin** |
| Fx | 10V=30lbf | Brown | A-9 | 9205-1 (AI0) | Black | A-18 | 9205-19 (AI8) |
| Fy | 10V=30lbf | Red | A-8 | 9205-2 (AI1) | Black | A-17 | 9205-20 (AI9) |
| Fz | 10V=30lbf | Orange | A-7 | 9205-3 (AI2) | Black | A-16 | 9205-21 (AI10) |
| Tx | 10V=100lbf-in | Yellow | A-6 | 9205-4 (AI3) | Black | A-15 | 9205-22 (AI11) |
| Ty | 10V=100lbf-in | Green | A-5 | 9205-5 (AI4) | Black | A-14 | 9205-23 (AI12) |
| Tz | 10V=100lbf-in | Blue | A-4 | 9205-6 (AI5) | Black | A-13 | 9205-24 (AI13) |
| Health | Read AI0: Closed = ON = healthy | Sea green | D-12 | 9403-4 (DIO3) | Pink/black | D-25 | 9403-3 (DIO2) |
| | | | | | 5kΩ resistor | 9403-3 (DI02) | 9403-29 (COM) |

## 2.2 Dynamic Model of InMotion2

### 2.2.1 Inertial Dynamics

The InMotion2 robot has four links, shown in Figure 2-3. The motors are located at the labeled motor axes and provide torque to links 1 and 4. The angle of the "shoulder" joint measured by the encoder in motor 1 is the angle between link 1 and the motor axis. Link 2 is parallel to link 4 so the angle of the "elbow" joint measured by the encoder in motor 2 is the angle between link 4 and the motor axis. The lengths ($l$) and masses ($m$) of each component are detailed in Table 2.5. The variable $r$ is the distance between the center of mass of the link and the joint closest

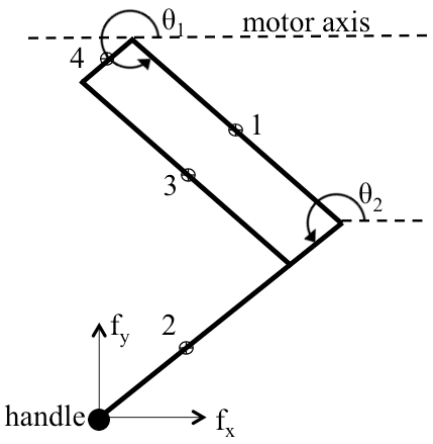



Figure 2-3: Inertial model

Table 2.5: Robot Link Properties

| **Property** | **link 1** | **link 2** | | | **link 3** | **link 4** |
|---|---|---|---|---|---|---|
| | | **link** | **handle** | **total** | | |
| **length** $(m)$ | 0.4064 | 0.5144 | 0.5016 | N/A | 0.4064 | 0.1555 |
| **mass** $(kg)$ | 0.756 | 0.892 | 1.072 | 1.964 | 0.756 | 0.378 |
| $r$ $(m)$ | 0.2032 | 0.2572 | 0.5016 | 0.3906 | 0.2032 | 0.0775 |
| $I_c$ $(kg * m^2)$ | 0.0104 | 0.0197 | 0 | 0.0488 | 0.0104 | 0.0007 |
| $I$ $(kg * m^2)$ | 0.0416 | 0.0787 | 0.2697 | 0.3484 | 0.0416 | 0.0030 |

to the motor axis. The moments of inertia about the center of mass ($I_c$) and the moments of inertia about the joint closest to the motor axis ($I$) for each component are calculated assuming that the links are uniform rods and the handle is a point mass.

If the four links were uncoupled from each other, each of the links would be free to move its in-plane position in the $x, y$, and $\theta$ directions, totaling 12 degrees of freedom for the uncoupled system. However the constraint of each robot joint reduces the system to two generalized degrees of freedom $\theta_1, \theta_2$ which are the spatial joint angles. Any kinematics affecting the uncoupled coordinates can be transformed into the generalized coordinates using transformation matrices. Taking the time derivative of

the transformation yields the full system Jacobian [28].

$$\boldsymbol{x} = \begin{bmatrix} x_{c1} \\ y_{c1} \\ \theta_1 \\ x_{c2} \\ y_{c2} \\ \theta_2 \\ x_{c3} \\ y_{c3} \\ \theta_3 \\ x_{c4} \\ y_{c4} \\ \theta_4 \end{bmatrix} = \begin{bmatrix} r_1\cos(\theta_1) \\ r_1\sin(\theta_1) \\ \theta_1 \\ l_1\cos(\theta_1) + r_2\cos(\theta_2) \\ l_1\sin(\theta_1) + r_2\sin(\theta_2) \\ \theta_2 \\ l_4\cos(\theta_2) + r_3\cos(\theta_1) \\ l_4\sin(\theta_2) + r_3\sin(\theta_1) \\ \theta_1 \\ r_4\cos(\theta_2) \\ r_4\sin(\theta_2) \\ \theta_2 \end{bmatrix} \quad \dot{\boldsymbol{x}} = J(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} = \begin{bmatrix} -r_1\sin(\theta_1) & 0 \\ r_1\cos(\theta_1) & 0 \\ 1 & 0 \\ -l_1\sin(\theta_1) & -r_2\sin(\theta_2) \\ l_1\cos(\theta_1) & r_2\cos(\theta_2) \\ 0 & 1 \\ -r_3\sin(\theta_1) & -l_4\sin(\theta_2) \\ r_3\cos(\theta_1) & l_4\cos(\theta_2) \\ 1 & 0 \\ 0 & -r_4\sin(\theta_2) \\ 0 & r_4\cos(\theta_2) \\ 0 & 1 \end{bmatrix} \dot{\boldsymbol{\theta}} \quad (2.2)$$

The uncoupled inertia matrix is as follows:

$$M = \text{diag}(m_1, m_1, I_{c1}, m_2, m_2, I_{c2}, m_3, m_3, I_{c3}, m_4, m_4, I_{c4}) \quad (2.3)$$

The inertia matrix is defined as the following [28].

$$I(\boldsymbol{\theta}) = J^T(\boldsymbol{\theta}) * M * J(\boldsymbol{\theta}) \quad (2.4)$$

Multiplying the system out and simplifying using trigonometric properties yields the following generalized inertia matrix.

$$I(\boldsymbol{\theta}) = \begin{bmatrix} I_1 + m_2 l_1^2 + I_3 & (m_2 l_1 r_2 + m_3 l_4 r_3)\cos(\theta_1 - \theta_2) \\ (m_2 l_1 r_2 + m_3 l_4 r_3)\cos(\theta_1 - \theta_2) & I_2 + m_3 l_4^2 + I_4 \end{bmatrix} \quad (2.5)$$

The Lagrangian is defined as the kinetic co-energy of the system [29].

$$L(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \frac{1}{2}\dot{\boldsymbol{\theta}}^T I(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{2.6}$$

For this system, the Lagrangian is defined as follows.

$$L(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \frac{1}{2}(I_1+m_2l_1^2+I_3)\dot{\theta_1}^2+(m_2l_1r_2+m_3l_4r_3)\cos(\theta_1-\theta_2)\dot{\theta_1}\dot{\theta_2}+(I_2+m_3l_4^2+I_4)\dot{\theta_2}^2 \tag{2.7}$$

A kinematic model of the generalized system is derived using Lagrange's Equation [29].

$$\boldsymbol{\tau} = \frac{d}{dt}\left(\frac{\delta L(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})}{\delta \dot{\boldsymbol{\theta}}}\right) - \frac{\delta L(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})}{\delta \boldsymbol{\theta}} \tag{2.8}$$

The derivative of the Lagrangian with respect to velocity is as follows.

$$\frac{\delta L(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})}{\delta \dot{\boldsymbol{\theta}}} = \begin{bmatrix} (I_1 + m_2l_1^2 + I_3) & (m_2l_1r_2 + m_3l_4r_3)\cos(\theta_1 - \theta_2) \\ (m_2l_1r_2 + m_3l_4r_3)\cos(\theta_1 - \theta_2) & (I_2 + m_3l_4^2 + I_4) \end{bmatrix} \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} \tag{2.9}$$

The derivative of the Lagrangian with respect to position is as follows.

$$\frac{\delta L(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})}{\delta \boldsymbol{\theta}} = \begin{bmatrix} -(m_2l_1r_2 + m_3l_4r_3)\sin(\theta_1 - \theta_2)\dot{\theta_1}\dot{\theta_2} \\ (m_2l_1r_2 + m_3l_4r_3)\sin(\theta_1 - \theta_2)\dot{\theta_1}\dot{\theta_2} \end{bmatrix} \tag{2.10}$$

Using Lagrange's formula from Equation 2.8 to combine the time derivative of Equation 2.10 with Equation 2.11 yields the following kinematic model.

$$\boldsymbol{\tau} = \begin{bmatrix} (I_1 + m_2l_1^2 + I_3)\ddot{\theta_1} + (m_2l_1r_2 + m_3l_4r_3)(\cos(\theta_1 - \theta_2)\ddot{\theta_2} + \sin(\theta_1 - \theta_2)\dot{\theta_2}^2) \\ (I_2 + m_3l_4^2 + I_4)\ddot{\theta_2} + (m_2l_1r_2 + m_3l_4r_3)(\cos(\theta_1 - \theta_2)\ddot{\theta_1} - \sin(\theta_1 - \theta_2)\dot{\theta_1}^2) \end{bmatrix} \tag{2.11}$$

If the inertial and centrifugal matrices are represented as $I$ and $C$, respectively, Lagrange's formula can take the following form, where external torque $\boldsymbol{\tau}$ is defined. $\boldsymbol{\tau_{in}}$ represents the applied motor torque, $\boldsymbol{\tau_h}$ represents the force applied to the handle by

the human converted to joint coordinates, $\boldsymbol{\tau_f}$ represents the frictional forces on the joints, and $\boldsymbol{d}$ represents time-varying model uncertainty.

$$\boldsymbol{\tau} = I(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} = \boldsymbol{\tau_{in}} + \boldsymbol{\tau_h} - \boldsymbol{\tau_f} + \boldsymbol{d} \tag{2.12}$$

The generalized inertia matrix $I(\boldsymbol{\theta})$ is defined in Equation 2.5 and the centrifugal matrix $C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$ is defined below.

$$C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \begin{bmatrix} 0 & (m_2 l_1 r_2 + m_3 l_4 r_3)\sin(\theta_1 - \theta_2)\dot{\theta}_2 \\ -(m_2 l_1 r_2 + m_3 l_4 r_3)\sin(\theta_1 - \theta_2)\dot{\theta}_1 & 0 \end{bmatrix} \tag{2.13}$$

### 2.2.2 Motor Torque and Applied Force

The applied torque for each motor $\boldsymbol{\tau_{in}}$ can be set directly by the controller and is considered the control input for this design. The force applied to the handle $\boldsymbol{F_h}$ is measured directly by a force-torque transducer at the handle. The handle rotates freely about its axis so it cannot apply torque to the end-effector; only forces $f_x$ and $f_y$ are considered. A new Jacobian $J_{ef}$ is created to relate the end-effector dynamics to joint dynamics using the same principle as Equation 2.2. First a forward kinematics model is created to specify the position of the end-effector.

$$\boldsymbol{x_{ef}} = \begin{bmatrix} l_1\cos(\theta_1) + l_2\cos(\theta_2) \\ l_1\sin(\theta_1) + l_2\sin(\theta_2) \end{bmatrix} \tag{2.14}$$

And the time derivative of this model reveals the Jacobian $J_{ef}$.

$$\dot{\boldsymbol{x_{ef}}} = J_{ef}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} = \begin{bmatrix} -l_1\sin(\theta_1) & -l_2\sin(\theta_2) \\ l_1\cos(\theta_1) & l_2\cos(\theta_2) \end{bmatrix} \dot{\boldsymbol{\theta}} \tag{2.15}$$

The joint torques due to applied handle force can be expressed as the following [28].

$$\boldsymbol{\tau_h} = J_{ef}(\boldsymbol{\theta})^T \boldsymbol{F_h} \tag{2.16}$$

When this model is used to implement a controller on the physical system, imperfections in the servo actuation will cause some difference between the controller commanded torque and actual motor torque $\boldsymbol{\tau_{in}}$. Likewise, imperfections in the handle force sensor will cause some difference between $\boldsymbol{F_h}$ and measured force. It is important to know how inaccurate these values are in order to be able to bound model uncertainty $\boldsymbol{d}$ from Equation 2.12.

The accuracy of the combination of both $\boldsymbol{\tau_{in}}$ and $\boldsymbol{\tau_h}$ can be measured using one simple test. This can be done by clamping the robot handle into a stationary position as in Figure 2-4. In this configuration torque can be commanded to each motor and the handle force can be measured and converted to joint torques. The magnitude of the differences between commanded joint torques and measured joint torques quantify the accuracy of the $\boldsymbol{\tau_{in}}$, $\boldsymbol{\tau_h}$, and static friction, isolated from the effects of the dynamic model or kinetic friction.

The results of this test are shown in Figure 2-5. Examining the difference between commanded and measured joint torque during the static torque test, the torque uncertainty for each joint is bounded by the values in Equation 2.17.

$$\left|\begin{bmatrix} d_{s1} \\ d_{s2} \end{bmatrix}\right| < \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix} Nm, \qquad \left|\begin{bmatrix} d_{s1}/\tau_{in1} \\ d_{s2}/\tau_{in2} \end{bmatrix}\right| < \begin{bmatrix} 0.04 \\ 0.03 \end{bmatrix} \tag{2.17}$$

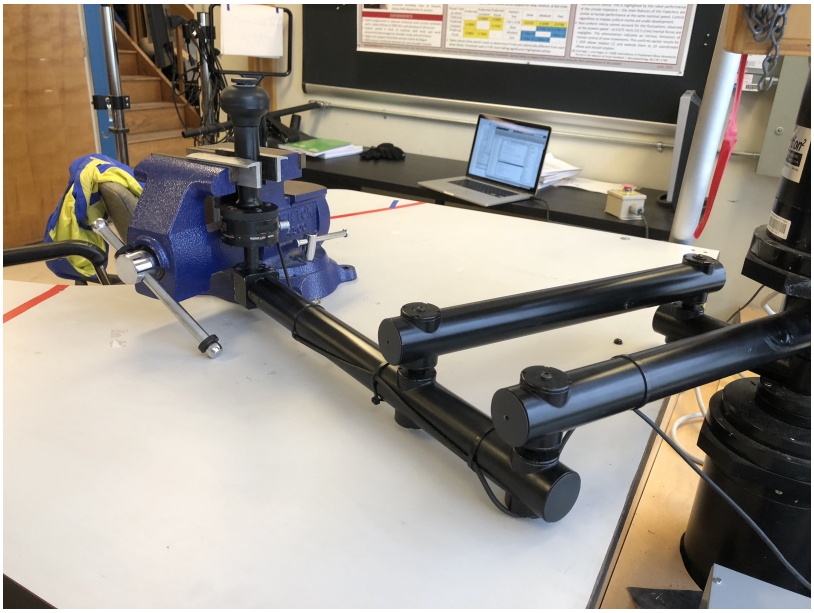
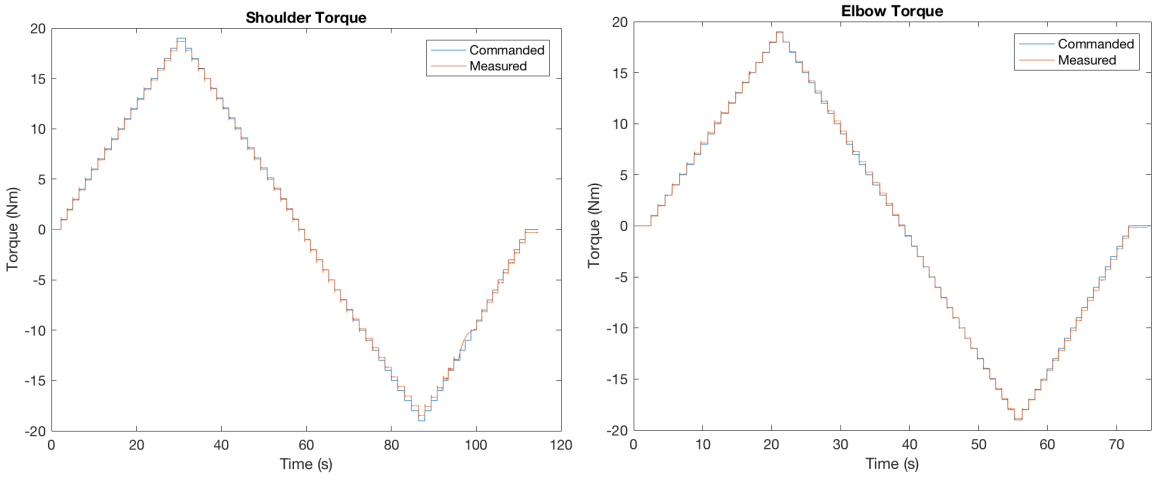


Figure 2-4: The clamp is applied to the end-effector above the force/torque transducer such that all forces generated by the robot motors are read by the force/torque transducer.




(a) Shoulder Torque

(b) Elbow Torque

Figure 2-5: Torques are commanded by the motors (blue) in a stairstep fashion. The force/torque transducer measures the forces and converts them to equivalent motor torques (red).

### 2.2.3 Friction and Model Uncertainty

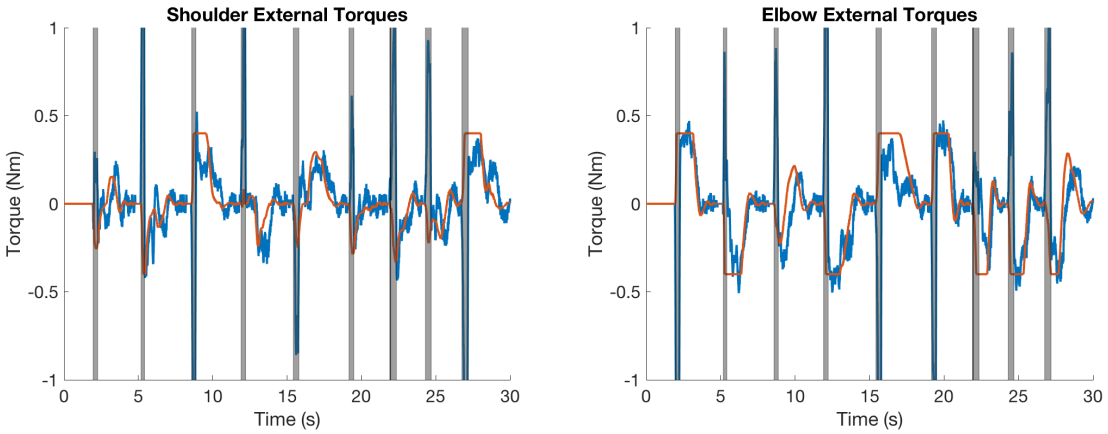
Although the robot has relatively smooth bearings and is extremely back-driveable, the joints of the robot are not perfect and do experience some friction. Friction is very difficult to characterize accurately, so a simple estimation is made as a continuous function combining Coulomb friction and viscous friction [30]. This model was created by examining the dynamic response of the robot using a simple test which quantified both friction and dynamic model accuracy, isolated from the effects of motor torque and applied forces.

When zero commanded torque is applied to the motors, the robot is free to move throughout the workspace. After applying an impulse force to the robot handle to generate an initial velocity, the only external torques on the robot joints are friction and any un-modeled dynamics such as gravity. The net sum of these external torques is known to be the sum of the inertial and centrifugal terms of Equation 2.12. The following friction model for each of the two degrees of freedom is created by examining the net sum of external torques $I(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}}$ shown in Figure 2-6.

$$\tau_{f1}[Nm] = \begin{cases} 0.2, & \dot{\theta_1} \geq 0.3\frac{rad}{s} \\ -0.2, & \dot{\theta_1} \leq -0.3\frac{rad}{s} \\ \frac{2}{3} * \dot{\theta_1}, & |\dot{\theta_1}| < 0.3\frac{rad}{s} \end{cases} + \begin{cases} 0.2, & \dot{\theta_1} - \dot{\theta_2} \geq 0.3\frac{rad}{s} \\ -0.2, & \dot{\theta_1} - \dot{\theta_2} \leq -0.3\frac{rad}{s} \\ \frac{2}{3} * (\dot{\theta_1} - \dot{\theta_2}), & |\dot{\theta_1} - \dot{\theta_2}| < 0.3\frac{rad}{s} \end{cases}$$

$$\tau_{f2}[Nm] = \begin{cases} 0.4, & \dot{\theta_2} \geq 0.3\frac{rad}{s} \\ -0.4, & \dot{\theta_2} \leq -0.3\frac{rad}{s} \\ \frac{4}{3} * \dot{\theta_2}, & |\dot{\theta_2}| < 0.3\frac{rad}{s} \end{cases} \tag{2.18}$$

The uncertainty in this friction model as well as the model dynamics can be quantified by examining the difference between the estimated external torques and the friction




(a) Shoulder Torque (b) Elbow Torque

Figure 2-6: Impulse forces are applied to the end-effector in the blacked-out regions. In the remaining regions the net sum of joint torques (blue) are attributed to Coulomb friction (red) and un-modeled dynamics.

model.

Examining data from the dynamic torque test dictates that this friction error and dynamic model uncertainty for each joint is bounded by the values in Equation 2.19, expressed in units of $Nm$ and as a percentage of the maximum torque seen by the unforced system of $0.4Nm$.

$$\left|\left|\begin{bmatrix} d_{f1} \\ d_{f2} \end{bmatrix}\right|\right| < \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix} Nm = \begin{bmatrix} 75 \\ 100 \end{bmatrix} \% \tag{2.19}$$

Summing the uncertainty quantified by the dynamic torque test with the uncertainty quantified by the static torque test creates a bound $\boldsymbol{D}$ for overall model uncertainty.

$$|\boldsymbol{d}| < \begin{bmatrix} 0.9 \\ 0.7 \end{bmatrix} Nm < D = 1Nm \tag{2.20}$$

# Chapter 3

# The InMotion2 Software

The National Instruments compactRIO contains both a real-time (RT) processor and a Field Programmable Gate Array (FPGA), both of which are used to control the robot. The RT processor has a maximum sampling rate of 1kHz, which slows down with added hardware interfacing and computation. The FPGA can run much faster, approaching 1MHz, but the update rate is limited to 2kHz because of the extensive hardware interface. The FPGA has direct access to the input and output modules and is programmed with firmware using special LabVIEW functions that are optimized for speed. Such optimization is possible because FPGAs use fixed-point notation in which each piece of data has an assigned range and resolution which must be specified during development. Compiling the FPGA code can be complex and time-consuming, especially when compared with the RT processor which requires no compilation. For this reason, the FPGA is used only for low-level functions that should not change often. The RT processor is able to execute traditional LabVIEW and is able to run a wide variety of functions using floating point notation. It is possible to use add-ins such as MathScript to enhance code development on the RT processor, and therefore it is used for high-level functions that may change depending on the task.

The LabVIEW software as well as the Matlab code to process data and robot documentation are managed on GitHub to aid in collaborative development [31]. A detailed description of each software function is included in Appendix B.

## 3.1 Real Time Software

The top-level program that runs on the cRIO real-time processor is contained in run.vi. This program initiates the FPGA low-level code which executes asynchronously at 2kHz, but only executes a timed loop at 1kHz. In this timed loop, the RT processor reads inputs from the FPGA, implements a controller based on those inputs, and sends the outputs of the controller back to the FPGA. While doing so it also continuously writes the data to a file and updates displays on the screen.

## 3.2 User Interface

The LabVIEW user interface to operate the InMotion2 robot is shown in Figure 3-1. The user can start the program by clicking the LabVIEW start arrow which will initiate the loops for both the FPGA and the RT processor. The user can also specify a name for the .tdms file that is automatically created with every run. The large STOP button will exit the program gracefully. The top half of the user input panel specifies parameters for the virtual constraint function described in Section 3.2.1 including a button to turn the virtual constraint on and off. The bottom half of the user input panel specifies parameters for the inertia compensation which will be described in more detail in Chapter 4. Additionally there are boolean indicators to report faults detected in the three sensors (the "shoulder" position encoder, the "elbow" position encoder, and the force/torque transducer). There is a toggle switch to zero out the force/torque sensor, and displays of the force/torque readings and the motor output values. On the right of the screen is a display which indicates where

the end-effector is located in relation to the circular constraint. All of these controls and indicators update live while the robot is running.

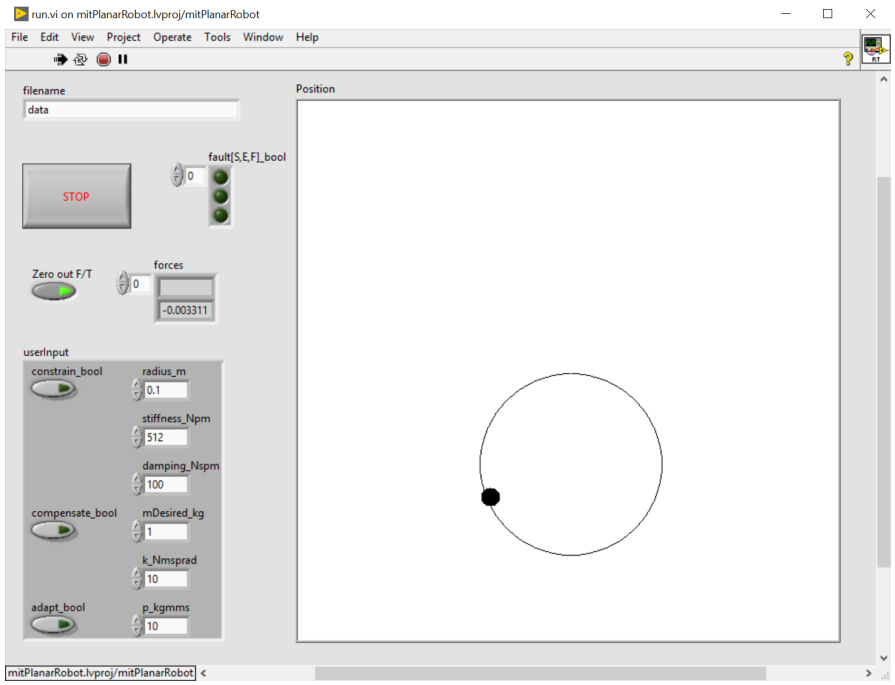



Figure 3-1: LabVIEW User Interface

### 3.2.1 Virtual constraint controller

The controller that runs on the real-time processor can easily be changed using traditional LabVIEW and can be programmed to perform all sorts of tasks. The primary controller creates a circular virtual constraint in order to perform constrained motion studies on humans. This controller forces the robot end-effector onto a circular virtual path by controlling the robot's apparent impedance. The FPGA processed sensor data contains the position and velocity in cartesian x,y coordinates. The virtual constraint controller uses this Cartesian data to determine the radial distance from the end-effector to the edge of the circle $r$ as well as the radial velocity $v_r$. The goal of the virtual constraint is for the end-effector to have the following impedance

characteristic, where $f_{h,r}$ is the radial component of the force on the end-effector [32].

$$ma_r = -k_{vc}r - b_{vc}v_r + f_{h,r} \tag{3.1}$$

**Without Inertia Compensation**

The natural behavior of the robot is to exhibit the following response, where $m$ is the natural apparent inertia of the robot in its specific configuration in the radial direction, and $f_{in}$ is the equivalent force produced by the robot motors in the radial direction.

$$ma_r = f_{h,r} + f_{in} \tag{3.2}$$

By allowing the apparent inertia $m$ in the impedance controller follow the natural inertia of the robot, Equation 3.1 and Equation 3.2 can be combined. This eliminates the need to use end-effector force $f_{h,r}$ or acceleration $a_r$.

$$f_{in} = -k_{vc}r - b_{vc}v_r \tag{3.3}$$

This controller is completely passive for the ideal case with no system delays or filtering lags, and remains passive for a wide range of impedances. The force $f_{in}$ is calculated in the RT processor and sent to the FPGA to be converted to motor torques if the constraint is active and inertia compensation is inactive.

**With Inertia Compensation**

If it is desirable to set the apparent inertia $m$ to a constant value, as is the case with inertia compensation, the controller for the virtual constraint is executed along with the inertia compensation in the FPGA. The implementation of the inertia compensation is detailed in Chapter 4. In order to keep the shape and direction of the virtual constraint as well as desired impedance as flexible as possible, these parameters are

specified in the RT processor. All that the low-level inertia compensator needs to know about the virtual constraint are the desired impedance parameters, the instantaneous direction of the constraint $\gamma$ in radians, and the orthogonal distance between the end-effector and the constraint $r$ in meters, shown in Figure 3-2. The orthogonal

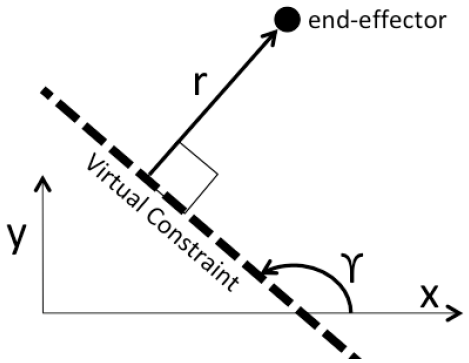



Figure 3-2: Virtual Constraint Parameters

force, velocity, and acceleration used in Equation 3.1 can be determined from the FPGA sensors along with the constraint direction $\gamma$ and orthogonal distance $r$ which are defined in Equation 3.4 for the circular constraint centered about the origin.

$$\gamma = \text{atan2}(y, x) + \frac{\pi}{2} \qquad r = \sqrt{x^2 + y^2} - radius \tag{3.4}$$

Note that the relationship between $\gamma$ and $r$ must stay consistent. The positive direction of $r$ must point in the negative $y$ direction when $\gamma = 0$ and the positive $y$ direction when $\gamma = \pi$. For the circular constraint with $\gamma$ defined in Equation 3.4, the positive direction of $r$ will always be radially outwards from the origin.

### Virtual Constraint Stability

The simple impedance control method defined in Equation 3.3 when controlling a system with perfect torque sources and velocity measurement is inherently passive regardless of the system it is controlling, as it dissipates energy as it moves towards the reference. However, time delays and lags in the system can limit the stability

for certain values of the impedance parameters. The update rate of the impedance controller (without inertia compensation) is 1kHz which is appreciably faster than the most significant natural frequencies of the system. The limiting factor for stability is actually the 30 Hz bandwidth of the velocity calculation, which will be described in greater detail in Section 3.3.1. Many different values of stiffness and damping were tested on the robot hardware to determine the limits of stability. For each of three points (x=0 m, y=-0.1 m, 0m, 0.1m), an impedance controller was implemented with varying levels of damping, then stiffness was increased until the system showed evidence of instability. For the lower damping values, the instability took the form of low frequency oscillations. For the higher damping values, the instability took the form of high frequency motor output which creates a buzzing noise in the motors. Figure 3-3 plots the stiffness limits against damping for the three cases. Within the plotted line the system remains stable in response to perturbation, and any oscillations are damped out within 2-3 seconds. Outside the line the system is unstable and oscillates or buzzes continuously. A software limit (purple) was implemented to ensure that the stiffness and damping remain within the stable region. To optimize virtual constraint performance while still guaranteeing stability, a stiffness of around 5000 N/m and damping of around 40 N/m/s are recommended for use with this controller.

### 3.2.2 Data logging

The real time processor has a handful of other small tasks in addition to the high-level control. Within the 1kHz loop the RT processor collects the low-level FPGA data as well as the high level RT processor data, converts them to floating point representation, and writes them to a FIFO data buffer on the compactRIO. Parallel to this process, data from the FIFO buffer is written to a TDMS file on the compactRIO.

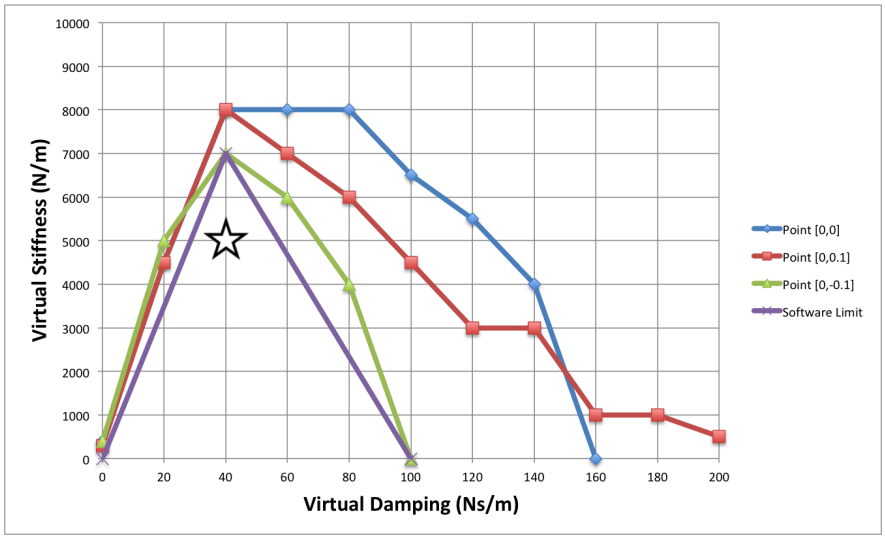



Figure 3-3: Virtual Constraint Stability

Separate Matlab code has been implemented to move the data from the compactRIO to a connected PC and read the TDMS file into Matlab for post-processing. The RT processor also has a special data logging procedure. When the user stops the run.vi program, it also takes the current force bias values described in Section 3.3.2 and writes this value to a file on the compactRIO in storeFTbias.vi. When the program is started back up again, the real-time processor will read this file and set the force bias to these values.

## 3.3 FPGA Firmware

The top-level program that runs the cRIO FPGA code is contained in runFPGA.vi. This program is initiated by the real-time program run.vi, and contains a timed loop at 2kHz. The FPGA code reads and processes the inputs values from the encoders and force/torque transducer, processes the control output commanded by the real-time software and provides an output signal to the robot motors. The FPGA code also includes optional inertia compensation with or without a constraint.

### 3.3.1 Encoder Measurement

Each motor has an encoder mounted on it to measure the position of the motor, which in turn indicates the location and orientation of the robot. The compactRIO uses digital lines to communicate with the Gurley Precision interpolating decoder [24] as described in Section 2.1.2. The FPGA program manipulates the hold, A0, and A1 bits and reads in the low, middle, and high bits of each encoder reading over lines B0-7. After all the readings are in, FPGA code concatenates these bits and converts them to fixed-point integers that represent the position of each motor as a number between 0 and 65535.

#### Position

To begin the position signal processing, a simple fault check is performed on each encoder to ensure that the signal received from the decoder is valid. The Euler method is used to predict the current position based on past measurements shown in Equation 3.5. The variable $\theta_{i-1}$ denotes the position at the previous time step and $\theta_{i-2}$ is the position two time steps previous. This prediction $\hat{\theta}_i$ is then compared to the measured position $\theta_i$. If the error between the measurement and the prediction has a magnitude that exceeds a threshold of 50 bits (out of 65536), an error flag is raised and the predicted value is used instead of the measured value.

$$\hat{\theta}_i = 2\theta_{i-1} - \theta_{i-2} \tag{3.5}$$

Next each selected decoder value is converted to radians in the coordinate frame described by Figure 2-3 by applying a scalar and adder. The forward kinematics model from Equation 2.14 is used to calculate the end-effector position in cartesian coordinates $x$ and $y$. This section of code also computes the Jacobian $J_{ef}$ based on Equation 2.15.

### Velocity

The velocities of the robot are calculated using the backward Euler method.

$$\dot{\theta}_i = \frac{\theta_i - \theta_{i-1}}{\Delta t} \tag{3.6}$$

Because the FPGA code is executed at a rate of 2kHz and each encoder has a resolution of 65536 over one revolution, there are often multiple iterations of FPGA that read the same encoder position even if the robot is moving. An Euler derivative of this discrete signal is quite choppy and unusable. There are many different methods for handling this type of signal such as applying a system observer to predict the robot velocity and using the measurement data to adjust the prediction. These potential methods are discussed in Chapter 6. A 2nd-order Butterworth low-pass filter was used to adjust current data based on past information in order to create a smooth velocity signal. The goal when designing the velocity filter was to remove the choppiness due to the discretized position signal, while still capturing the dynamics of the robot. Figure 3-4 shows velocity data from the robot oscillating at 10 Hz. The raw

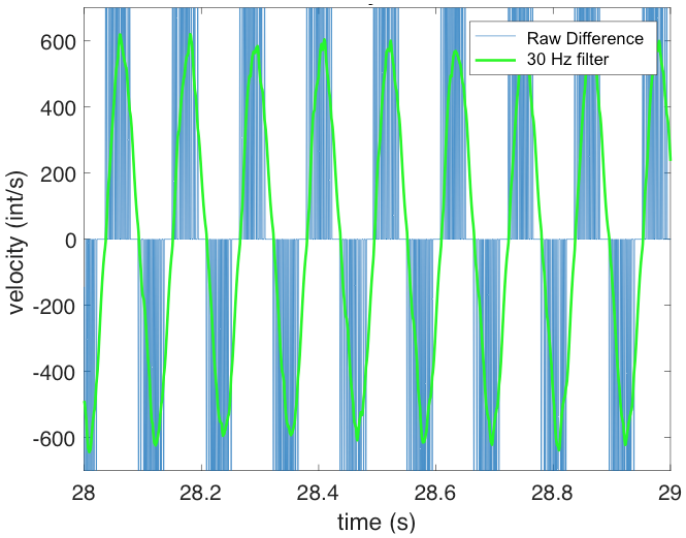



Figure 3-4: Velocity Filter

Euler difference equation yields an unusable velocity signal that jumps between 0 a

very high magnitude number, but accurately depicts the phase of oscillation. There is a direct tradeoff between filter smoothness and phase lag. Responding quickly to the raw signal makes the signal choppy; waiting to respond makes the signal smoother but at the expense of adding phase lag. Filtering the signal at a bandwidth of 30 Hz yields a much smoother signal but at the cost of some phase lag, shown here around 45° or 12.5 ms. Figure 3-4 shows the phase lag as the difference between the phase of the filter input (blue) and the filter output (green). This phase lag affects stability of controllers that use the velocity signal, and must be taken into account when designing controllers for the system. After filtering the radial velocity signals, the Jacobian $J_{ef}$ from Equation 2.15 is used to convert the radial velocity to end-effector velocity in cartesian coordinates.

#### Acceleration

When using position measurements to calculate acceleration, the signal can get quite noisy, especially since the encoders on this robot have limited resolution. It is not advisable to use an acceleration signal in a feedback controller, but it is used to limited effect in the inertia compensation logic. A backwards Euler derivative scheme is used to calculate acceleration of the end-effector in Cartesian coordinates from the calculated velocity. This signal is passed through a 4th-order Butterworth low-pass filter at 50 Hz, which has a sharper high-frequency rolloff than the velocity filter. This is to ensure that the signal is smoothed while adding minimal phase lag to the acceleration signal.

### 3.3.2 Force/Torque Measurement

The force/torque sensor processing in the FPGA is fairly simple, as the F/T controller [26] described in Section 2.1.3 provides most of the processing. Each of the 6 forces and torques are read in via differential analog voltage signals and multiplied

by a scalar to convert the forces to Newtons and the torques to Newton-meters using the conversion factors from Table 2.4. The signals in engineering units are passed through 2nd-order Butterworth low-pass filters in order to mitigate any signal noise. The bandwidth for this filter was chosen to match that of the velocity filter, at 30 Hz.

The force/torque transducer is mounted to the end-effector with tightly fastened screws. These screws along with the gravity of the handle above it provide static loading to the force/torque transducer. In order to get the most accurate force/torque reading as possible, a force/torque bias is applied to the initial reading to zero out the sensor when no external loads are applied. The user is able to re-calibrate the bias at the push of the "Zero out F/T" button on the user interface depicted in Figure 3-1, and this bias is saved to a file in the compactRIO so that the bias calibration can be stored in between runs.

The force/torque sensor measures force and torque along the x, y, z axes of the force/torque transducer itself. As the end-effector is moved about the workspace, the orientation of the transducer changes. The following equations translate the force/torque measurements into the global coordinate system, where $\alpha$ is the angle between the force/torque transducer axis which depends on $\theta_2$ and the global coordinate axis defined in Figure 2-3.

$$\begin{bmatrix} f_{x,global} \\ f_{y,global} \end{bmatrix} = \begin{bmatrix} f_{x,rel}\cos(\alpha) + f_{y,rel}\sin(\alpha) \\ f_{y,rel}\cos(\alpha) - f_{x,rel}\sin(\alpha) \end{bmatrix} \tag{3.7}$$

### 3.3.3 Output processing

The real-time processor contains a high-level controller that sends outputs to the FPGA described in Section 3.1. These outputs can either be motor torques or equivalent end-effector forces along with an optional constraint direction. If the high-level controller has specified end-effector forces and inertia compensation has been turned off, the FPGA converts those forces to joint torques using the following relationship.

$$\boldsymbol{\tau_{in}} = J_{ef}(\boldsymbol{\theta})^T \boldsymbol{F_{in}} \tag{3.8}$$

If inertia compensation is turned on, the FPGA will ignore the force/torque commands from the high level controller and perform compensation internally. The details of the inertia compensation are provided in Chapter 4.

The ability of the robot's motors to control the end-effector in Cartesian coordinates is dependent on the robot's configuration. If the arm is fully extended with $\theta_1$ nearly equal to $\theta_2$, the "elbow" joint can only push the robot in the same direction as the "shoulder" joint, and the robot cannot be controlled in the orthogonal direction. This is also true if the arm is retracted towards the motors such that $\theta_1$ is nearly equal to $-\theta_2$. A good measure of the controllability of a robot is the condition number of the Jacobian matrix transpose $J_{ef}^T$ [33]. All combinations of joint angles $\theta_1$ and $\theta_2$ are tested for condition number, and the configurations with a condition number less than 2 are deemed to be well-conditioned. Figure 3-5 shows a top-down view of the workspace with the well-conditioned end-effector locations are plotted in green. This region can be bounded by two circles about the robot motors of radii 0.45 meters and 0.8 meters, shown in black.

The bounds of the table beneath the robot are shown in red in Figure 3-5. If the

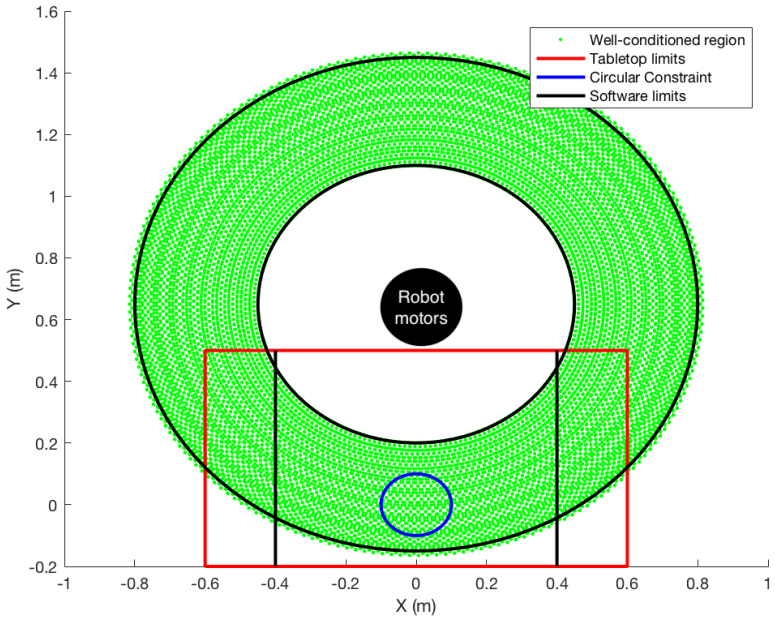



Figure 3-5: InMotion2 well-conditioned region

robot end-effector were to go beyond the bounds of the table, it could potentially be harmful to people or lab equipment. For conservatism, this region is bounded by the black vertical lines at $\pm 0.4$ meters. If the robot end-effector goes outside the linear or circular bounds defined here, no matter what the controller is dictating, the FPGA will implement a damping control to slow the robot down in an attempt to keep the end-effector out of this region.

If the commanded torque for either motor exceeds the capability of the motor, both torque commands are scaled down proportionally to preserve the relative direction of the command. These torque commands are then converted to voltage commands according to the conversion factors listed in Table 2.1 and the voltages are sent to the servo amplifiers which send current to the motors.

# Chapter 4

# Inertia Compensation

The purpose of the inertia compensation is to force the robot to respond to forces on the end-effector with acceleration proportional to the handle force, just as point mass would.

$$f = ma \tag{4.1}$$

When no controller input is applied, the relationship between force and acceleration of the end-effector is not a scalar relationship and varies across the workspace. The arrangement of the links of the robot cause the inertia to be greater in some directions than in others, nonlinearly depending on configuration. The behavior of a point mass is best represented as an admittance [28] so an admittance controller is designed to match the acceleration $\ddot{x}$ of the robot end-effector to a reference that is proportional to the applied force.

There are many different ways to achieve this end, a few of which are detailed in Section 1.2.2. The controller design for this application takes on the form of Figure 4-1. This controller structure has some similarities to the strategies presented in Section 1.2.2. First the measured force exerted by the human ($\boldsymbol{F_h}$) goes through an admittance model. This model differs depending on whether a virtual constraint

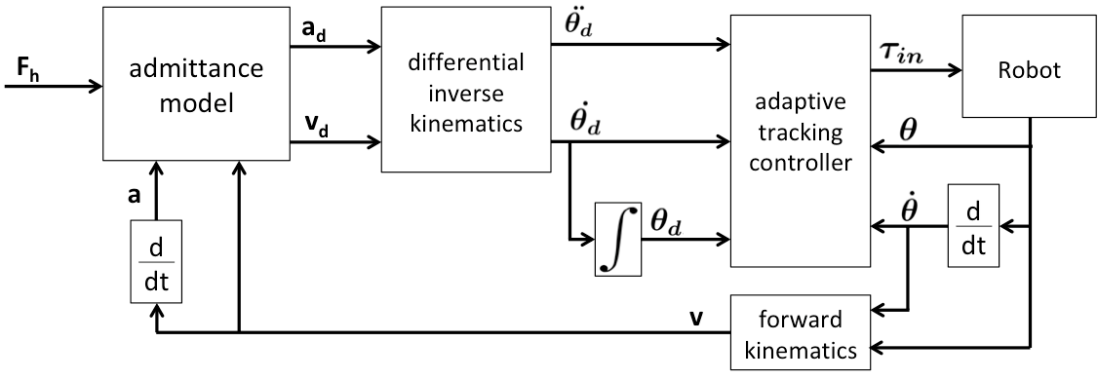



Figure 4-1: Inertia Compensator Block Diagram

is desired or if only inertia compensation is active. The measured acceleration ($\boldsymbol{a}$) is used to subtract off the reaction forces caused by the inertia of the handle above the force-transducer, which resembles the inertial load logic used by [12] and [13]. If a virtual constraint is active, the admittance model includes stiffness and damping similar to the virtual contact in [12] which relies on measured velocity ($\boldsymbol{v}$) in addition to the inputs from the high-level controller. The acceleration and velocity references ($\boldsymbol{a_d}$, $\boldsymbol{v_d}$) in Cartesian coordinates go through differential inverse kinematics to convert to joint coordinates ($\ddot{\boldsymbol{\theta}}_{\boldsymbol{d}}$, $\dot{\boldsymbol{\theta}}_{\boldsymbol{d}}$, $\boldsymbol{\theta_d}$), and an adaptive tracking controller is used to generate joint torques ($\boldsymbol{\tau_{in}}$) that force the robot on the specified trajectory. The adaptive tracking controller contains a feedforward component to cancel modeled forces similar to [11] but also contains a feedback component based on measured joint position and velocity ($\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$) to ensure the robot is staying on track similar to [13].

## 4.1 Admittance Model

The admittance model shown in Figure 4-1 uses the measured force $\boldsymbol{F_h}$ to generate acceleration and velocity references $\boldsymbol{a_d}$ and $\boldsymbol{v_d}$. First some processing of the input force is required, then either a pure inertial admittance is applied if the virtual constraint is not active or a combined inertial, damping, and stiffness admittance model is applied if the virtual constraint is active.

### 4.1.1 Applied Force

The force/torque transducer is located at the end-effector, with a handle mounted on top of it for a human to grip. If the end-effector is accelerating whether or not a human is applying force to the handle, the handle itself exerts inertial forces that are read by the transducer opposite the direction of acceleration. An example of this is demonstrated in Figure 4-2. Here the end-effector is freely rotating around a circular virtual constraint at constant speed without being touched by the human, and the acceleration of the circular motion causes a sinusoidal force profile (blue line, obscured). This measured value does not represent the actual force applied by the human, and can be compensated out by adding the inertial force of the handle (red line) to the measured force. Note that the mass of the handle $m_h$ just represents the part of the handle above the force transducer and not the mass of the entire handle assembly described in Table 2.5. The mass of the handle $m_h$ has been estimated based on acceleration data to be 0.44 kg.

$$\begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} f_{x,global} \\ f_{y,global} \end{bmatrix} + m_h \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} \tag{4.2}$$

This equation does rely on measured acceleration, which noted before can be either exceptionally noisy or lagged by low-pass filtering. However, this usage gives very limited authority to the acceleration measurement, as the inertial forces of the handle are relatively low compared to the potential forces provided by the human. The force signal has a minor phase lag from the 2nd order Butterworth low-pass filter with a cutoff frequency of 30Hz in ftFilterFPGA.vi. The magnitude of this applied force $f_{mag}$ is calculated, as is the direction of the applied force $\beta$.

$$\beta = \arctan\left(\frac{f_y}{f_x}\right) \tag{4.3}$$

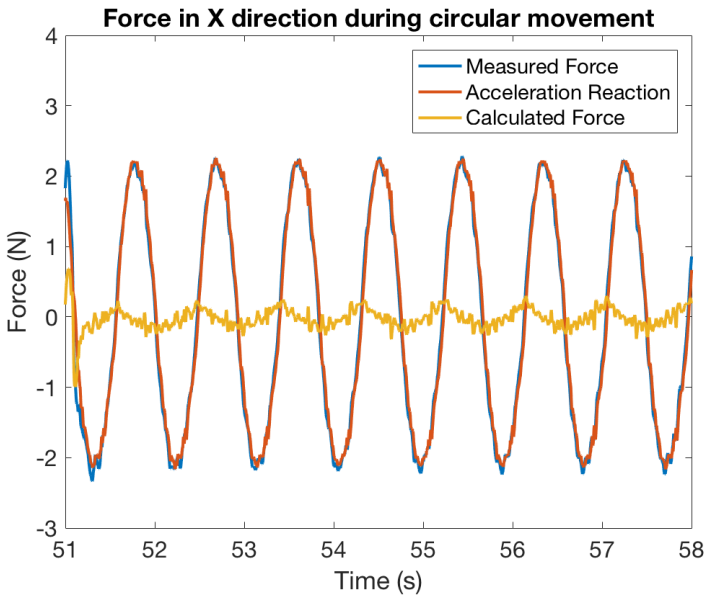



Figure 4-2: Measured Forces when Freely Moving About Circular Constraint

The sum of the measured force and the inertial force of the handle is shown as the yellow line in Figure 4-2. This value is reduced to a very low number but the forces are not completely canceled due to timing differences between the force and acceleration signals. For this reason, a small dead zone of 0.3 N was applied to the force magnitude as shown in Figure 4-3. This conservatively accounted for force and acceleration measurement inaccuracy. The deadband allowed for the measured force to be slightly non-zero, as in Figure 4-2, and not react to it. When actual force is applied to the handle the magnitude is biased downwards, in the conservative direction. The dead zone is implemented as follows:

$$f_h = \begin{cases} f_{mag} - 0.3 & f_{mag} > 0.3 \\ f_{mag} + 0.3 & f_{mag} < -0.3 \\ 0 & |f_{mag}| \leq 0.3 \end{cases} \tag{4.4}$$

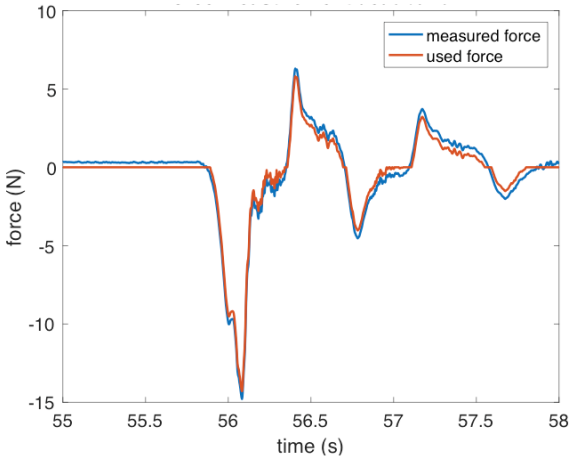



Figure 4-3: Force deadband

### 4.1.2 Unconstrained Reference

If there is no virtual constraint active, the inertia compensator needs to work in both dimensions of the xy-plane. According to Equation 3.2, the acceleration of the end effector should be proportional to the force applied in Cartesian coordinates. By extension, the magnitude of the acceleration should be proportional to the magnitude of the force applied. The mass $m$ is specified as a user input.

$$a_d = \frac{1}{m} f_h \tag{4.5}$$

The desired acceleration in x and y coordinates can be calculated using acceleration magnitude $a_d$ and the direction of the applied force $\beta$.

$$\boldsymbol{a_d} = a_d \begin{bmatrix} \cos(\beta) \\ \sin(\beta) \end{bmatrix} \tag{4.6}$$

Forward Euler integration is used to calculate a velocity reference in x and y coordinates as shown below. $T_s$ represents the time step of the FPGA program, which is

0.0005s.

$$\boldsymbol{v_{d,i}} = \begin{cases} \boldsymbol{v_{d,i-1}} + \boldsymbol{a_d} T_s & \text{if compensation is active} \\ \dot{\boldsymbol{x}} & \text{if compensation is not active} \end{cases} \tag{4.7}$$

### 4.1.3 Constrained Reference

If there is a virtual constraint active, the compensator needs to operate as an inertia compensator in one direction and a virtual constraint in the orthogonal direction. One strategy for achieving this end is a hybrid approach wherein position control compensates the inertia in the direction of the constraint and force control (as described in the high-level controller in Equation 3.3) enforces the constraint in the orthogonal direction, and the two controllers are superimposed [34]. However, this technique only works if each controller is strictly one-dimensional in Cartesian space. The virtual constraint controller presented in Equation 3.3 is indeed one-dimensional and does not output forces in orthogonal direction. The inertia compensator calculates the tracking error and compensates for the robot dynamics in two-dimensional joint space. The compensator relies contains a full two-dimensional system model and relies on the assumption that it is the only controller active. The compensator cannot be superimposed with any other controller for this reason.

Because the two orthogonal objectives cannot be superimposed at controller output, the virtual constraint and inertia compensation are combined into a single trajectory reference using superimposed admittance models to be followed by the single controller. To this end, the high-level controller specifies a constraint direction $\gamma$ as well as an orthogonal distance from the constraint $r$, both defined in Section 3.2.1.

### Constraint Direction

The force $f_h$ can be projected in the direction of the constraint. The component of the force in the direction of $\gamma$ is defined as $f_\gamma$, where $\beta$ is defined in Equation 4.3 and $\gamma$ is defined in Equation 3.4.

$$f_\gamma = f_h \cos(\gamma - \beta) \tag{4.8}$$

The magnitude of desired acceleration is this force value divided by the desired mass $m$.

$$a_{d,\gamma} = \frac{1}{m} f_\gamma \tag{4.9}$$

The acceleration characteristic for this component in $x$ and $y$ coordinates is defined below.

$$\boldsymbol{a_{d,\gamma}} = a_{d,\gamma} \begin{bmatrix} \cos(\gamma) \\ \sin(\gamma) \end{bmatrix} \tag{4.10}$$

The desired acceleration is integrated to find the desired velocity in Cartesian coordinates. Because the constraint is not necessarily linear, the magnitude of desired acceleration should be integrated, ignoring the directional component. For example if a circular constraint is active and no external forces are being applied to the handle, the magnitude of the velocity should stay constant but the direction should move around the circle. Therefore when a constraint is active, the magnitude of acceleration is integrated to yield the magnitude of velocity.

$$v_{d,\gamma,i} = \begin{cases} v_{d,\gamma,i-1} + a_{d,\gamma} T_s & \text{if compensation is active} \\ v_\gamma & \text{if compensation is not active} \end{cases} \tag{4.11}$$

Note that a standard Euler form of integration is used, in which the change in velocity is added to the previous *desired* velocity. This is because inertia compensation relies on the most correct velocity reference possible. After an impulse force is applied to

the robot, the end-effector should maintain constant velocity that does not degrade over time. This velocity magnitude is then mapped back into x and y coordinates in the direction of the constraint.

$$\boldsymbol{v_{d,\gamma}} = v_{d,\gamma} \begin{bmatrix} \cos(\gamma) \\ \sin(\gamma) \end{bmatrix} \tag{4.12}$$

**Orthogonal Direction**

The direction orthogonal to the constraint can be used to create a reference trajectory using a scheme similar to position-based impedance control (PB-IC) [35]. Using this control scheme, a reference acceleration is derived from the input force and desired impedance characteristics, and then integrated to determine the desired velocity and position. First, the component of the applied force in this direction is defined below.

$$f_{\gamma+\pi/2} = f_h \sin(\gamma - \beta) \tag{4.13}$$

The virtual constraint is applied using impedance control, where a specified mass ($m$), stiffness ($k_{vc}$), and damping ($b_{vc}$) are simulated between the robot end-effector and the desired constraint. The following represents the desired characteristic orthogonal to $\gamma$, similar to the relationship described in Equation 3.1.

$$a_{d,\gamma+\pi/2} = \frac{1}{m}(-k_{vc}r - b_{vc}v_{d,\gamma+\pi/2} + f_{\gamma+\pi/2}) \tag{4.14}$$

The velocity used in this equation is the *desired* velocity, or the integral of desired acceleration. The measured velocity lags behind the desired velocity due to both tracking error in the controller and the phase loss in the velocity calculation. Using desired velocity minimizes the overall phase loss in the impedance controller. The distance to the constraint $r$ is defined by the high-level controller and represents the

difference between *measured* position and the constraint. If *desired* position had been used, it might "wander" with the integral of desired velocity. It is important that the position of the constraint stays fixed in space, thus the measured position is used. The acceleration characteristic can be separated into $x$ and $y$ coordinates.

$$\boldsymbol{a_{d,\gamma+\pi/2}} = a_{d,\gamma+\pi/2} \begin{bmatrix} \sin(\gamma) \\ -\cos(\gamma) \end{bmatrix} \tag{4.15}$$

The acceleration is integrated to find velocity in Equation 4.16. Note that this time the change in velocity is added to the *measured* velocity rather than the previous *desired* velocity. This is done in order to emulate instantaneous model-based impedance control [35]. Because a feedback controller is used to reduce the error between measured and desired velocity, the gain in the feedback controller can act as a multiplier on virtual damping. Integrating the desired acceleration based on *measured* velocity removes this gain multiplier and allows the system to achieve the desired impedance characteristic [35]. This allows the feedback controller to have a high gain without increasing the impedance of the virtual constraint.

$$v_{d,\gamma+\pi/2,i} = \begin{cases} v_{\gamma+\pi/2} + a_{d,\gamma+\pi/2} T_s & \text{if compensation is active} \\ v_{\gamma+\pi/2} & \text{if compensation is not active} \end{cases} \tag{4.16}$$

This velocity magnitude is then mapped back into x and y coordinates in the direction of the constraint.

$$\boldsymbol{v_{d,\gamma+\pi/2}} = v_{d,\gamma+\pi/2} \begin{bmatrix} \sin(\gamma) \\ -\cos(\gamma) \end{bmatrix} \tag{4.17}$$

### Superposition

The superimposed desired acceleration is the sum of the desired acceleration in the constraint direction from Equation 4.10 and orthogonal direction from Equation 4.15.

$$\boldsymbol{a_d} = \boldsymbol{a_{d,\gamma}} + \boldsymbol{a_{d,\gamma+\pi/2}} \tag{4.18}$$

Likewise, the superimposed desired velocity is the sum of the desired velocity in the constraint direction from Equation 4.12 and the orthogonal direction from Equation 4.17.

$$\boldsymbol{v_d} = \boldsymbol{v_{d,\gamma}} + \boldsymbol{v_{d,\gamma+\pi/2}} \tag{4.19}$$

### Fading Away the Constraint

If the values of stiffness and damping from Equation 4.14 are reduced to zero, the superimposed acceleration reference from Equation 4.18 is equivalent to the unconstrained acceleration reference of Equation 4.6, acting to compensate inertia in both dimensions of the xy-plane. However the superimposed velocity reference from Equation 4.19 will not necessarily be equal to the unconstrained velocity reference of Equation 4.7. This is because the constrained inertia compensation acts to preserve momentum in the direction of the constraint $\gamma$. Even with no forces acting on the robot (and therefore zero desired acceleration), if the end-effector is moving parallel to the defined constraint direction $\gamma$ and the direction of $\gamma$ rotates, the inertia compensator will rotate the direction of movement with $\gamma$.

In order to successfully fade away the constraint, the high-level controller could ramp down the values of stiffness and damping nearly to zero, at which point it could switch over to the unconstrained inertia compensator, which will maintain linear momentum regardless of constraint direction.

### 4.1.4 Differential Inverse Kinematics

The desired velocity in Cartesian coordinates can be mapped to joint coordinates given the current state of the robot using Equation 2.15. Note that the system Jacobian relies on the *measured* position, and can be calculated without knowing the desired position.

$$\dot{\boldsymbol{\theta}}_{\boldsymbol{d}} = J_{ef}^{-1}(\boldsymbol{\theta})\boldsymbol{v_d} \tag{4.20}$$

The time derivative of Equation 2.15 yields the acceleration conversion.

$$\boldsymbol{a_d} = \dot{J_{ef}}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}_{\boldsymbol{d}} + J_{ef}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}_{\boldsymbol{d}} \tag{4.21}$$

The desired joint acceleration can be found by isolating $\ddot{\boldsymbol{\theta}}_{\boldsymbol{d}}$.

$$\ddot{\boldsymbol{\theta}}_{\boldsymbol{d}} = J_{ef}^{-1}(\boldsymbol{\theta})\left(\boldsymbol{a_d} - \dot{J_{ef}}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}_{\boldsymbol{d}}\right) \tag{4.22}$$

Finally, the desired joint position can be found by integrating $\dot{\boldsymbol{\theta}}_{\boldsymbol{d}}$.

$$\boldsymbol{\theta_{d,i}} = \boldsymbol{\theta_{i-1}} + \dot{\boldsymbol{\theta}}_{\boldsymbol{d}} T_s \tag{4.23}$$

Note once again that the integration is based on *measured* position, this time for all the objectives. This is important for the virtual constraint so that the position controller does not amplify the virtual stiffness [35]. It is important for the inertia compensator as well. If desired position was obtained through a standard integration, the position error of the controller could potentially become very large in the event that the controller could not track the position reference, such as through un-sensed forces on the robot linkage. This could cause the inertia compensator to drive very large torques to get back to the intended virtual trajectory, which is counter-productive for creating the objective of constant inertia.

## 4.2 Adaptive Tracking Controller

The dynamics of the robot derived in Equation 2.12 are very nonlinear, and therefore linear techniques cannot reliably be used to control the system or to analyze system stability. One effective nonlinear technique is based on sliding mode control [36].

### 4.2.1 Sliding Surface

The position demand error $\tilde{\boldsymbol{\theta}}$ and velocity demand error $\dot{\tilde{\boldsymbol{\theta}}}$ are defined as the difference between the measured value and the desired value.

$$\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \boldsymbol{\theta_d} \qquad \dot{\tilde{\boldsymbol{\theta}}} = \dot{\boldsymbol{\theta}} - \dot{\boldsymbol{\theta}}_{\boldsymbol{d}} \tag{4.24}$$

A sliding surface $\boldsymbol{s}$ can be defined within the state-space of the position and velocity errors, using the parameter $\lambda$ to represent control bandwidth.

$$\boldsymbol{s} = \dot{\tilde{\boldsymbol{\theta}}} + \lambda\tilde{\boldsymbol{\theta}} \tag{4.25}$$

The sliding surface $\boldsymbol{s} = \mathbf{0}$ is represented as a line in the state space diagram of Figure 4-4. Sliding mode control reduces the nonlinear dynamics of the robot into the linear system described by Equation 4.25. Such a controller would attract the system from its initial condition towards the $s = 0$ line, within the $s_\Delta = 0$ region, and keep it within this region. Because this line has a negative slope, the properties of the system itself force the system to move towards the origin of the state space diagram where the error terms $\dot{\tilde{\boldsymbol{\theta}}} = \mathbf{0}$ and $\tilde{\boldsymbol{\theta}} = \mathbf{0}$ [36].

A reference parameter is used to aid in the creation of a sliding mode controller [37].

$$\dot{\boldsymbol{\theta}}_{\boldsymbol{r}} = \dot{\boldsymbol{\theta}}_{\boldsymbol{d}} - \lambda\tilde{\boldsymbol{\theta}} \qquad \ddot{\boldsymbol{\theta}}_{\boldsymbol{r}} = \ddot{\boldsymbol{\theta}}_{\boldsymbol{d}} - \lambda\dot{\tilde{\boldsymbol{\theta}}} \tag{4.26}$$

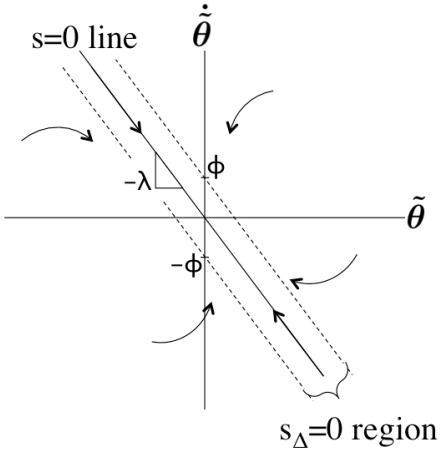



Figure 4-4: State Space Representation

The sliding variable $\boldsymbol{s}$ can be defined in terms of this reference parameter.

$$\boldsymbol{s} = \dot{\boldsymbol{\theta}} - \dot{\boldsymbol{\theta}}_{\boldsymbol{r}} \qquad \dot{\boldsymbol{s}} = \ddot{\boldsymbol{\theta}} - \ddot{\boldsymbol{\theta}}_{\boldsymbol{r}} \tag{4.27}$$

The sliding controller will only be as effective as the model used to create it. Because there is some known bounded model uncertainty $\boldsymbol{d}$ described in Equation 2.20, the system can only be guaranteed to converge to some region $\boldsymbol{s_\Delta}$ around $\boldsymbol{s} = \mathbf{0}$, defined by a dead zone of size $\phi$ [36]. The term $s_\Delta$ for each degree of freedom is defined below [38].

$$s_\Delta = \begin{cases} s - \phi & s > \phi \\ s + \phi & s < -\phi \\ 0 & |s| \leq \phi \end{cases} \tag{4.28}$$

### 4.2.2 System Dynamics

The system dynamics from Equation 2.12 can be re-defined in terms of the new sliding variable $\boldsymbol{s}$ using the definition of $\dot{\boldsymbol{s}}$ from Equation 4.27.

$$I(\boldsymbol{\theta})\dot{\boldsymbol{s}} = I(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} - I(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}_{\boldsymbol{r}} = \boldsymbol{\tau_{in}} - I(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}_{\boldsymbol{r}} - C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} + \boldsymbol{\tau_h} - \boldsymbol{\tau_f} + \boldsymbol{d} \tag{4.29}$$

It is useful to define a new matrix of measured time-varying quantities $Y$ and vector of unknown fixed quantities $\boldsymbol{a}$ such that when multiplied together they yield the following [37].

$$Y\boldsymbol{a} = I(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}_{\boldsymbol{r}} + C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}}_{\boldsymbol{r}} \tag{4.30}$$

The unknown fixed constants used in the $I$ and $C$ matrices can be reduced to just three parameters in the $\boldsymbol{a}$ vector.

$$\boldsymbol{a} = \begin{bmatrix} I_1 + m_2 l_1^2 + I_3 \\ I_2 + m_3 l_4^2 + I_4 \\ m_2 l_1 r_2 + m_3 l_4 r_3 \end{bmatrix} \tag{4.31}$$

The matrix $Y$ that fulfills the requirements of Equation 4.30 with the vector $\boldsymbol{a}$ from Equation 4.31 is defined below.

$$Y(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \dot{\boldsymbol{\theta}}_{\boldsymbol{r}}, \ddot{\boldsymbol{\theta}}_{\boldsymbol{r}}) = \begin{bmatrix} \ddot{\theta_{r_1}} & 0 & \cos(\theta_1 - \theta_2)\ddot{\theta_{r_2}} + \sin(\theta_1 - \theta_2)\dot{\theta}_2\dot{\theta_{r_2}} \\ 0 & \ddot{\theta_{r_2}} & \cos(\theta_1 - \theta_2)\ddot{\theta_{r_1}} - \sin(\theta_1 - \theta_2)\dot{\theta}_1\dot{\theta_{r_1}} \end{bmatrix} \tag{4.32}$$

Integrating Equation 4.30 into Equation 4.29 yields the following. The term $C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} - C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}}_{\boldsymbol{r}}$ is replaced with $C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\boldsymbol{s}$.

$$I(\boldsymbol{\theta})\dot{\boldsymbol{s}} = \boldsymbol{\tau_{in}} - Y(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \dot{\boldsymbol{\theta}}_{\boldsymbol{r}}, \ddot{\boldsymbol{\theta}}_{\boldsymbol{r}})\boldsymbol{a} - C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\boldsymbol{s} + \boldsymbol{\tau_h} - \boldsymbol{\tau_f} + \boldsymbol{d} \tag{4.33}$$

The following is a reasonable controller for the system, with $\hat{\boldsymbol{a}}$ being an approximation for the matrix of unknown parameters $\boldsymbol{a}$.

$$\boldsymbol{\tau_{in}} = Y(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \dot{\boldsymbol{\theta}}_{\boldsymbol{r}}, \ddot{\boldsymbol{\theta}}_{\boldsymbol{r}})\hat{\boldsymbol{a}} - \boldsymbol{\tau_h} + \boldsymbol{\tau_f} - k\boldsymbol{s} \tag{4.34}$$

Replacing $\boldsymbol{\tau_{in}}$ from Equation 4.33 with Equation 4.34 yields the following equation for the term $I(\boldsymbol{\theta})\dot{\boldsymbol{s}}$, where $\tilde{\boldsymbol{a}} = \hat{\boldsymbol{a}} - \boldsymbol{a}$.

$$I(\boldsymbol{\theta})\dot{\boldsymbol{s}} = Y(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \dot{\boldsymbol{\theta}}_{\boldsymbol{r}}, \ddot{\boldsymbol{\theta}}_{\boldsymbol{r}})\tilde{\boldsymbol{a}} - k\boldsymbol{s} - C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\boldsymbol{s} + \boldsymbol{d} \tag{4.35}$$

### 4.2.3 Stability

Barbalat's Lemma is used to assess the stability of the system using Lyapunov function $V(x,t)$. Barablat's Lemma says that the system is stable when the following conditions are met [36]:

1. $V(\boldsymbol{x}, t)$ is lower bounded
2. $\dot{V}(\boldsymbol{x}, t)$ is negative semi-definite
3. $\dot{V}(\boldsymbol{x}, t)$ is uniformly continuous in time

The Lyapunov function used for this system is defined below, where P is a symmetric positive definite constant matrix.

$$V = \frac{1}{2}\boldsymbol{s_\Delta}^T I(\boldsymbol{\theta})\boldsymbol{s_\Delta} + \frac{1}{2}\tilde{\boldsymbol{a}}^T P^{-1}\tilde{\boldsymbol{a}} \tag{4.36}$$

**Stability Condition 1: $V(x,t)$ is lower bounded**

The first stability condition of Barbalat's Lemma is that the Lyapunov function must be lower bounded. Because each term in Equation 4.36 is quadratic and positive, $V$ is positive semi-definite, meaning that it has a lower bound at $V = 0$. Therefore Barbalat's first condition is satisfied.

**Stability Condition 2: $\dot{V}(x,t)$ is negative semi-definite**

The second stability condition of Barbalat's Lemma is that the time derivative of the Lyapunov function must be negative semi-definite. The time derivative of Equation

4.36 is the following.

$$\dot{V} = \boldsymbol{s_\Delta}^T I(\boldsymbol{\theta})\dot{\boldsymbol{s_\Delta}} + \frac{1}{2}\boldsymbol{s_\Delta}^T \dot{I}(\boldsymbol{\theta})\boldsymbol{s_\Delta} + \dot{\tilde{\boldsymbol{a}}}^T P^{-1}\tilde{\boldsymbol{a}} \tag{4.37}$$

The term $I(\boldsymbol{\theta})\dot{\boldsymbol{s_\Delta}}$ can be replaced with Equation 4.35. Because $\boldsymbol{a}$ is a vector of constant dimensions and inertias, $\dot{\tilde{\boldsymbol{a}}} = \dot{\hat{\boldsymbol{a}}}$.

$$\dot{V} = \boldsymbol{s_\Delta}^T \left(Y\tilde{\boldsymbol{a}} - k\boldsymbol{s} - C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\boldsymbol{s_\Delta} + \boldsymbol{d}\right) + \frac{1}{2}\boldsymbol{s_\Delta}^T \dot{I}(\boldsymbol{\theta})\boldsymbol{s_\Delta} + \dot{\hat{\boldsymbol{a}}}^T P^{-1}\tilde{\boldsymbol{a}} \tag{4.38}$$

This equation can be rearranged to consolidate the system into three terms.

$$\dot{V} = \frac{1}{2}\boldsymbol{s_\Delta}^T \left(\dot{I}(\boldsymbol{\theta}) - 2C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\right)\boldsymbol{s_\Delta} + \left(\boldsymbol{s_\Delta}^T Y + \dot{\hat{\boldsymbol{a}}}^T P^{-1}\right)\tilde{\boldsymbol{a}} + \boldsymbol{s_\Delta}^T (\boldsymbol{d} - k\boldsymbol{s}) \tag{4.39}$$

At this point the system can be broken down term by term. Part of the first term is defined below.

$$\dot{I}(\boldsymbol{\theta}) - 2C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \begin{bmatrix} 0 & -a_3 \sin(\theta_1 - \theta_2)(\dot{\theta_1} + \dot{\theta_2}) \\ a_3 \sin(\theta_1 - \theta_2)(\dot{\theta_1} + \dot{\theta_2}) & 0 \end{bmatrix} \tag{4.40}$$

It should be noted that the transpose of this matrix is just the negative of itself, meaning that this matrix is skew-symmetric [37]. Skew symmetric matrices have the unique property that when multiplied by the same vector twice, as this matrix is multiplied by $\boldsymbol{s_\Delta}$ in the first term of Equation 4.39, the product is zero. Hence the first term of Equation 4.39 can be removed.

The second term in Equation 4.39 can zero out as well, as long as the following adaptation law is applied. Note that this will freeze adaptation when the magnitudes of the values within the $\boldsymbol{s}$ vector are within the dead zone $\phi$ so that the time-varying

model uncertainty does not impact adaptation [38].

$$\dot{\hat{\boldsymbol{a}}} = -PY^T \boldsymbol{s_\Delta} \tag{4.41}$$

The initial values for $\hat{\boldsymbol{a}}$ are calculated from Equation 4.31 using the approximate measurements in Table 2.5.

$$\hat{\boldsymbol{a}}_{\mathbf{0}} = \begin{bmatrix} 0.4076 \\ 0.3696 \\ 0.3356 \end{bmatrix} \tag{4.42}$$

It can be assumed that these values are accurate within $\pm 0.05$, and upper and lower bounds can be applied to them without invalidating the stability analysis. For example, if one of the value estimates drifts upwards to $\hat{a}_1 = 0.45$, it can be concluded that $\hat{a}_1 > a_1$, meaning that $\tilde{a}_1 > 0$. If at this point the $\dot{\hat{a}}_1$ term of the adaptation law in Equation 4.41 is positive, a post-adaptation upper limit of $\dot{\hat{a}}_1 = 0$ makes the second term of Equation 4.39 more negative than it would have been with the un-limited $\dot{\hat{a}}_1$, and therefore more stable. The same is true for the reverse case where $\tilde{a}_1 < 0$, and for all the adaptable terms $\hat{a}_1$, $\hat{a}_2$, and $\hat{a}_3$. For this reason, upper and lower bounds can be applied to the adaptation law for all three parameters.

The purpose of the adaptation law is to allow the unknown inertias to adapt based on observation of the dynamics of the system. The lengths of the robot links affect the system Jacobian which does not adapt based on observed dynamics. This is because the link lengths can be measured more accurately than the robot inertias and are assumed to be accurate within the certainty of the model.

Finally, the only term left in $\dot{V}$ is the third term in Equation 4.39 which can be re-written as the following.

$$\dot{V} = \boldsymbol{s_\Delta}^T (\boldsymbol{d} - k\boldsymbol{s}) \tag{4.43}$$

If $\boldsymbol{s_\Delta}$ is zero, then $\dot{V}$ is zero. If $\boldsymbol{s_\Delta}$ is non-zero, the equation can be re-written.

$$\dot{V} = \boldsymbol{s_\Delta}^T (\boldsymbol{d} - k\boldsymbol{\phi} * \text{sign}(\boldsymbol{s_\Delta})) - \boldsymbol{s_\Delta}^T k \boldsymbol{s_\Delta} \tag{4.44}$$

Because the time-varying uncertainty $\boldsymbol{d}$ is bounded, we know that the term inside the parentheses of the first term has the opposite sign to $\boldsymbol{s_\Delta}$ and the product will be negative or zero as long as the following product is greater than or equal to the bounds of the time-varying uncertainty $\boldsymbol{D}$ [38].

$$k\boldsymbol{\phi} \geq \boldsymbol{D} \tag{4.45}$$

The second term in Equation 4.44 will always be negative if $\boldsymbol{s_\Delta}$ is non-zero, making $\dot{V}$ always negative semi-definite and conforms to the second stability condition of Barbalat's Lemma.

**Stability Condition 3: $\dot{V}(x,t)$ is uniformly continuous in time**

The Lyapunov derivative $\dot{V}$, defined in Equation 4.43, combines the terms $\boldsymbol{s}$, $\boldsymbol{s_\Delta}$, and $\boldsymbol{d}$ through multiplication and subtraction. As long as those three terms are all continuous in time, so is the Lyapunov derivative. All of the terms of the derivative of $\boldsymbol{s}$ in Equation 4.35 are finite, meaning that $\boldsymbol{s}$ is continuous in time. Additionally the formula for $\boldsymbol{s_\Delta}$ in Equation 4.28 indicates that $\boldsymbol{s_\Delta}$ is continuous in time if $\boldsymbol{s}$ is continuous in time. Finally, the time-varying uncertainty $\boldsymbol{d}$ is a combination of steady-state error and transient error which is assumed to be continuous. Thus, the system conforms to the third stability condition of Barbalat's Lemma.

The adaptive tracking controller presented meets all three conditions of Barbalat's Lemma. The controller consists of the control law in Equation 4.34, the adaptation law defined in Equation 4.41, and the gain to boundary layer relationship defined in

Equation 4.45.

### 4.2.4 Parameter Tuning

One advantage of this adaptive tracking controller is that it requires minimal tuning. The four constant parameters that need to be specified are control bandwidth $\lambda$, dead-zone $\phi$, gain $k$, and adaptation matrix $P$.

#### Control Bandwidth $\lambda$

The controller bandwidth can be set directly using the parameter $\lambda$, whose units are $\frac{rad}{s}$. This value must be chosen carefully so as to not excite any instabilities in the system. To choose this value requires consideration of everything in the system that could cause a resonance [36].

1. Structural resonant modes: The InMotion2 hardware is very rigid and its resonant modes have high frequencies. The frequency of one of the links can be determined using the following formula, where link length $L = 0.51m$, Young's modulus $E = 68.9 * 10^9 Pa$, 4th polar moment of inertia $I = 5.2 * 10^{-8}$, mass per length $\mu = 2kg/m$.

$$f = \frac{1}{2\pi}\left(\frac{\pi}{L}\right)^2 \sqrt{\frac{E * I}{\mu}} = 256Hz \tag{4.46}$$

   This frequency is higher than the other resonances in the system and therefore will not be the limiting factor for consideration of controller bandwidth.

2. Neglected time delays: The system time delays can be determined by examining the results from static force test described in Section 2.2.2. A zoomed-in version of Figure 2-5, shown in Figure 4-5, shows that the overall time delay is 0.018 seconds. Note that this includes a 30Hz filter on the force measurement, which is the same bandwidth as the velocity filter. Slotine and Li determined that

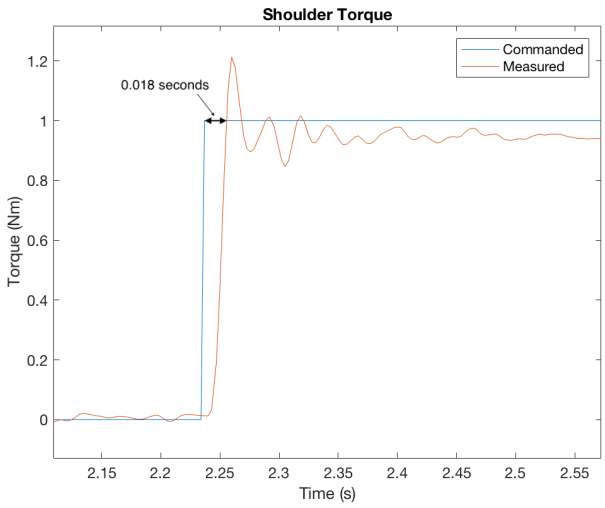



Figure 4-5: Static Torque Test: Zoomed In

the control bandwidth should adhere to the following equation, where $T_A$ is the largest unmodeled time delay [36].

$$\lambda \leq \frac{1}{3T_A} = 18.5\frac{rad}{s} \tag{4.47}$$

3. Sampling rate: The sampling rate of the inertia compensator is 2000 Hz. The condition presented by Slotine and Li is [36]:

$$\lambda \leq \frac{1}{5}\nu_{sampling} = 400\frac{rad}{s} \tag{4.48}$$

To optimize controller performance, the bandwidth was set as high as possible while still staying within the limitations.

$$\lambda = 18\frac{rad}{s} \tag{4.49}$$

**Gain** $k$

The ideal feedback gain $k$ was determined during testing, detailed in Chapter 5. The ideal gain for the system was determined below.

$$k = 20\frac{Nms}{rad} \tag{4.50}$$

**Dead-Zone** $\phi$

The dead-zone $\phi$ is chosen such that the product of the dead-zone and the gain follow Equation 4.45, using the definition of $D$ from Equation 2.20.

$$\phi = \frac{D}{k} = \frac{1Nm * rad}{20Nms} = 0.05\frac{rad}{s} \tag{4.51}$$

**Adaptation Matrix** $P$

The requirements for $P$ are that it must be a symmetric positive definite constant matrix. For simplicity, $P$ is defined as the identity matrix.

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.52}$$

**Desired inertia** $m$

The desired inertia is meant to be a flexible parameter dependent on the objectives of the task at hand. Experimental testing proved that the controller is stable with a desired inertia of $m = 1kg$ but is unstable with a desired inertia of $m = 0.5kg$. This coincides with Colgate's theory that force feedback cannot maintain passivity when attempting to program the apparent inertia to less than half of the natural inertia of the system [18]. The natural inertia of the InMotion2 varies between 1kg

and 2.5kg according to 5-7. The desired inertia should be limited to a minimum of $m = 1kg$. Analysis in Chapter 5 demonstrates that the desired mass should remain below a threshold in order for the accuracy of the inertia compensation to be within a human's perception ability.

$$1kg \leq m \leq 4kg \tag{4.53}$$

# Chapter 5

# Implementation and Testing

The controller described in Chapter 4 along with the software described in Chapter 3 was implemented on the InMotion2 robot. The controller has the flexibility to simulate different levels of inertia $m$, so each test was run with desired masses of 1kg, 2kg, 3kg, 4kg, 5kg, and 6kg. Each of these tests was run with different levels of feedback gain $k$ in order to optimally tune the controller for performance and stability.

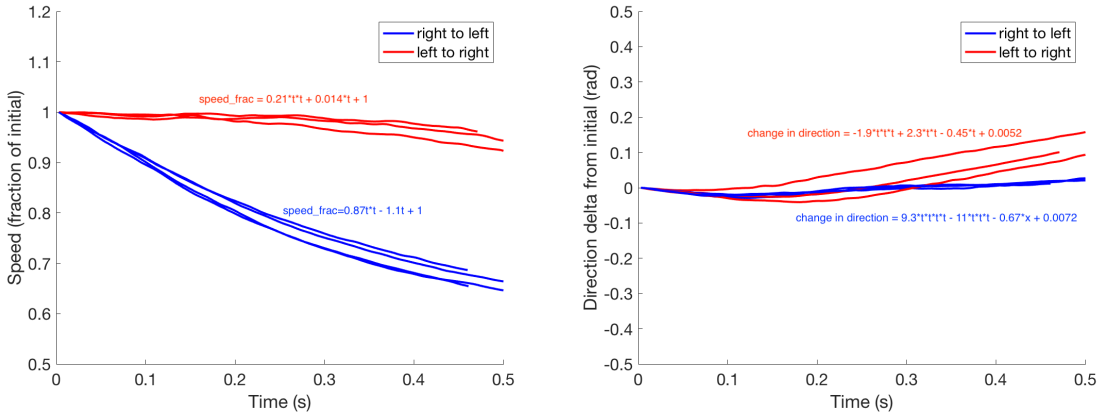
## 5.1 Initial Condition Response

The first test that was performed was an initial condition response test. The end-effector of the robot was lightly tossed from one end of the workspace to the other, and was untouched as it traveled across the workspace. A robot with perfect inertia would behave like a hockey puck on an ice rink, traveling with constant speed and direction when no external forces are applied.

### 5.1.1 Unconstrained Compensation

The following tests were performed using the unconstrained inertia compensation, where the robot was free to move anywhere within the workspace, and inertia com-

pensation was active to ensure that the acceleration of the robot was proportional to the applied force in both magnitude and direction. Figure 5-1 shows the speed (left)

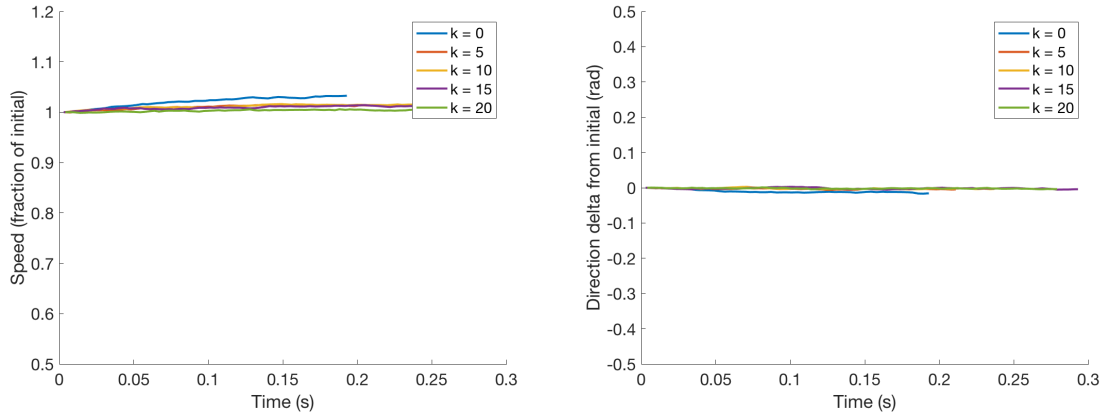



(a) Speed (b) Direction

Figure 5-1: Uncompensated Speed and Direction: A system with constant inertia would travel with constant speed and direction in response to an initial condition. However the natural response of the robot is highly nonlinear, and is best approximated by 2nd, 3rd, and 4th order equations.

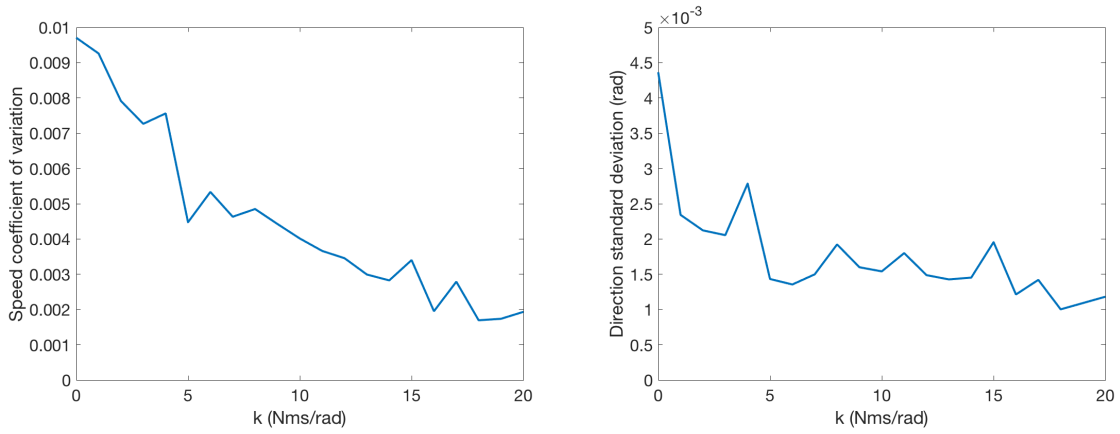
and direction (right) of the uncompensated robot end-effector after initial conditions were applied. Joint friction causes the speed to decrease as the robot moves across the workspace, and the variable natural inertia of the robot causes the trajectory to gradually change direction during the movement. These combined effects cause a trajectory that is best fit by 2nd, 3rd, and 4th order equations. Not only is the apparent inertia not constant, it is very nonlinear.

When the same test was run with inertia compensation turned on, the end-effector velocity stayed much more constant over the transient. Figure 5-2 shows the speed (left) and direction (right) for $m = 2kg$ for varying levels of feedback gain $k$. Because no external forces were applied during this test, the desired acceleration was zero regardless of the desired mass. Even with no feedback compensation with $k = 0\frac{Nms}{rad}$, the feedforward portion of the inertia compensation does an excellent job of maintaining the desired trajectory and keeping the speed and direction of travel constant.




(a) Speed (b) Direction

Figure 5-2: Compensated Speed and Direction, mass = 2kg: When inertia compensation is active the initial condition response moves much closer to constant speed and direction compared with the responses of Figure 5-1.




(a) Speed Error (b) Direction Error

Figure 5-3: Compensator Accuracy, mass=2kg: The coefficient of variation of speed decreases with increasing controller gain $k$, as does the standard deviation of direction of travel.

As the feedback gain $k$ is increased, the response just gets better, shown in Figure 5-3. The coefficient of variation of the speed throughout the response decreases with increasing $k$ (left), and the standard deviation of trajectory direction is minimized with diminishing returns above $k > 5Nms/rad$. Above a gain of $k = 20\frac{Nms}{rad}$ the commanded motor torque began switching fast enough for the motors to respond, which created buzzing noises that are undesirable.

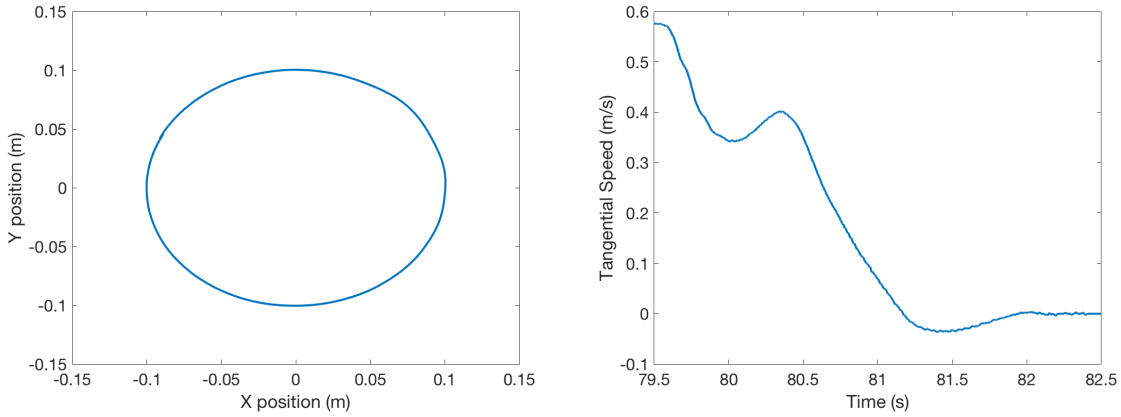
The results for this test show marked improvement over the uncompensated robot dynamics and proves that the adaptive tracking controller is quite effective at maintaining a desired trajectory. However this test does not demonstrate the robot's response in the presence of nonzero desired acceleration.

### 5.1.2 Constrained Compensation

The same impulse tests that were conducted for the unconstrained inertia compensator have also been performed for the constrained inertia compensator. The high-level controller was programmed to simulate the circular virtual constraint described in Section 3.2.1. The recommended stiffness of 5000 N/m was used for the virtual constraint, with the recommended damping of 40 Ns/m.

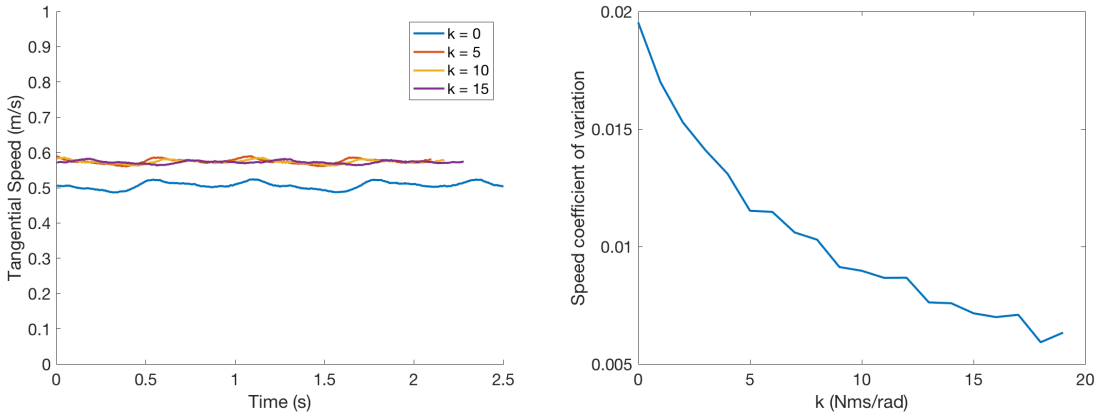
With the virtual constraint active, the robot end-effector was provided with an initial speed and the resulting tangential velocity was recorded as the end-effector moved around the constraint. Figure 5-4 shows the trajectory of the robot end-effector as it moved around the uncompensated virtual constraint (left) and the tangential speed of the end-effector (right). The virtual constraint acted to force the end-effector orthogonal to the constraint but did not act tangentially to the constraint. Joint friction dissipated energy as the robot traveled which caused a decrease in overall velocity. The configuration-dependent apparent inertia decreased and increased throughout

the transient, causing fluctuations in tangential speed that cannot be explained by friction alone. Because the virtual constraint was active, we are only interested in the one dimension of inertia compensation.




(a) Trajectory (b) Tangential Speed

Figure 5-4: Circular Virtual Constraint Without Inertia Compensation: When compensation was not active joint friction caused the end-effector to slow down and varying inertia caused the end-effector to accelerate and decelerate nonlinearly.




(a) Tangential Speed (b) Speed Error

Figure 5-5: Tangential Speed With Inertia Compensation: When compensation was active, tangential speed was maintained (left). The coefficient of variation (right) decreased with increasing feedback gain.

When inertia compensation was active, the virtual constraint tangential speed remained constant and did not degrade, as seen in Figure 5-5. As the controller feedback

gain is increased the oscillations decrease.

## 5.2 Human Interaction

The next test that was performed was the human interaction test. The human tester gripped the robot handle and moved it around the workspace with varying forces and directions to observe how the inertia compensator performed in the presence of changing forces. To maintain consistency, each test involved the robot end-effector being moved in a general circular motion such as the one shown in Figure 5-6, in free space with no virtual constraints.

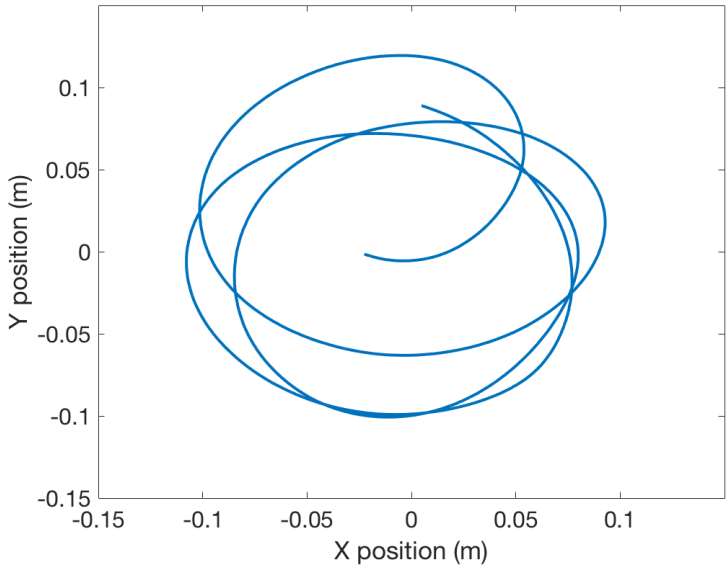
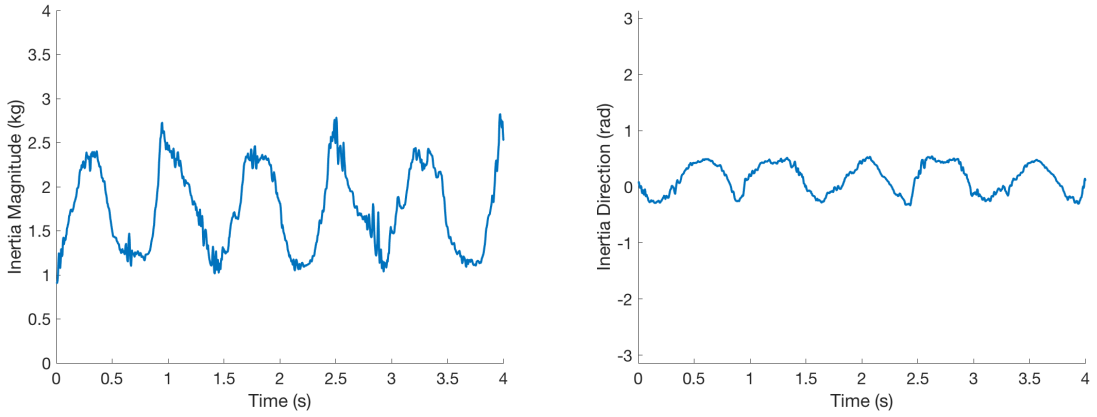



Figure 5-6: Human Interaction Test Trajectory

In this test, the robot end-effector accelerated as it was forced about the workspace. Ideally when a force vector with both magnitude and direction is applied to the robot handle, the end-effector should accelerate with a magnitude proportional to the applied force magnitude, and in the same direction. The apparent inertia magnitude is therefore defined as the ratio of applied force magnitude to acceleration magnitude,

in which the goal is to achieve an apparent inertia magnitude equal to the desired inertia. The apparent inertia direction is defined as the angle between the applied force and the resulting acceleration, in which the goal is for this angle to be zero. The inertia magnitude and direction were measured with the following equations, where $f_h$ is the corrected force magnitude derived in Equation 4.4, and $a_x$ and $a_y$ represent the measured acceleration in x and y directions, respectively.

$$m_{magnitude} = \frac{f_h}{\sqrt{a_x^2 + a_y^2}} \qquad m_{direction} = \text{atan2}(f_y, f_x) - \text{atan2}(a_y, a_x) \tag{5.1}$$




(a) Magnitude (b) Direction

Figure 5-7: Uncompensated Inertia

Without compensation, the inertia magnitude and direction change based on configuration and direction. The inertia for the circular trajectory from Figure 5-6 is displayed in Figure 5-7. As the end-effector moved around the circle the apparent inertia changed between $\sim 1kg$ and $\sim 2.5kg$.

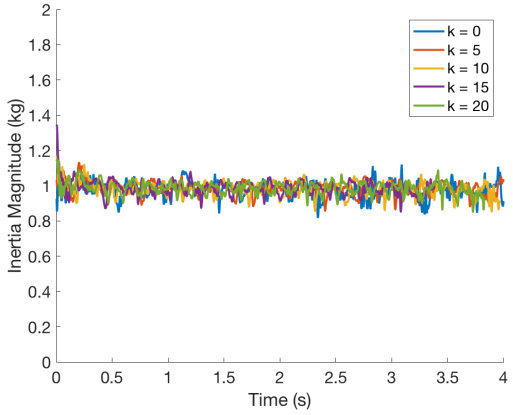
When compensation was active, the apparent inertia was close to constant. Figures 5-8 through 5-13 show the same test performed in Figure 5-7 but with compensation

turned on at desired mass of 1kg through 6kg, for varying levels of feedback gain $k$. The direction of apparent inertia (right) matched the desired direction with minimal deviation. Figure 5-14b shows the standard deviation of direction in units of radians for each value of compensator gain $k$. The magnitude of apparent inertia (left) matched the desired inertia with some deviation from the mean for each case. The coefficient of variation of inertia magnitude error for each gain value of each mass is plotted in Figure 5-14a.

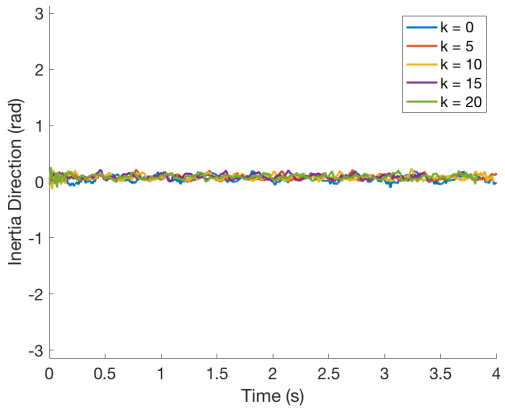
In general the standard deviation of inertia error decreases with increasing gain. However the compensation reaches diminishing returns at $k = 20$, as $k = 25$ begins to excite noise in the system. Therefore the ideal gain for the inertia compensation is $k = 20\frac{Nms}{rad}$.

When a compensator feedback gain of $k = 20$ is selected, Figure 5-14b shows that the error in inertia direction (angle between force and acceleration) is not strongly tied to the desired mass. The magnitude of inertia error is strongly correlated with desired mass, as shown in Figure 5-14a. In order to determine which desired masses are acceptable for human interaction applications, it is important to know what differential values of inertia are detectable by a human. In 2005 Tanaka et al. conducted a small study to estimate a human's ability to perceive changes in haptic inertia [39]. They found that human's inertia perception followed Weber's law, in which a human's perception sensitivity to an incremental increase in a stimulus is proportional to the magnitude of the stimulus [40]. In 1960 Rees and Copeland determined the Weber fraction (delta stimulus divided by magnitude of stimulus) for mass perception [41] which is slightly more conservative than the estimate by Tanaka et al.

The average human perception of differential inertia is plotted in blue in Figure

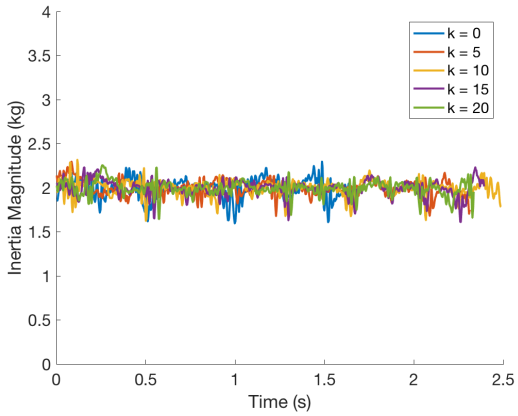



(a) Magnitude

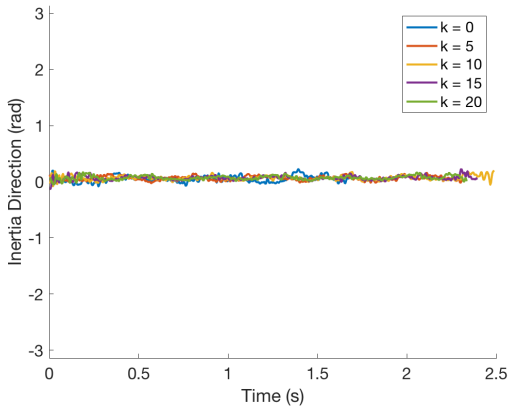



(b) Direction

Figure 5-8: Compensated Inertia, mass=1kg

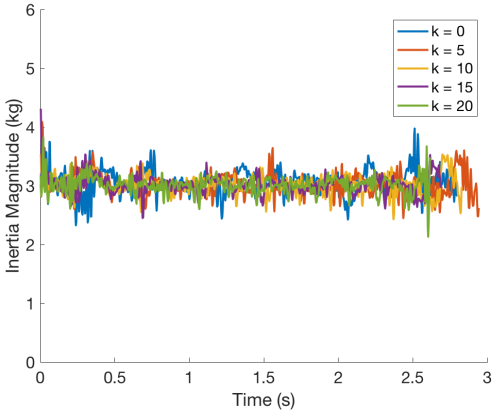



(a) Magnitude

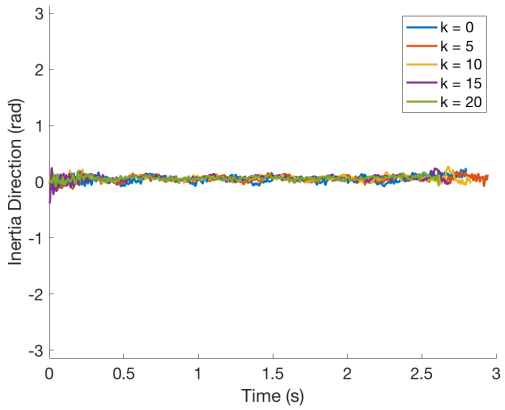



(b) Direction

Figure 5-9: Compensated Inertia, mass=2kg

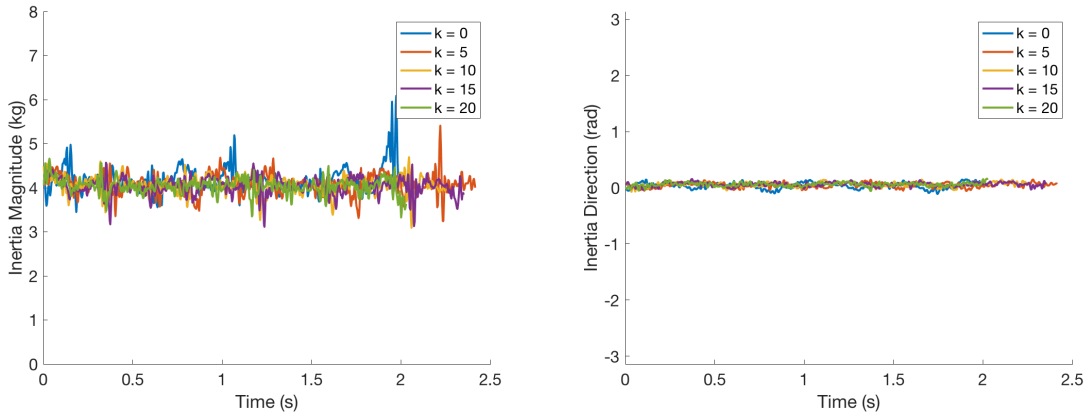



(a) Magnitude



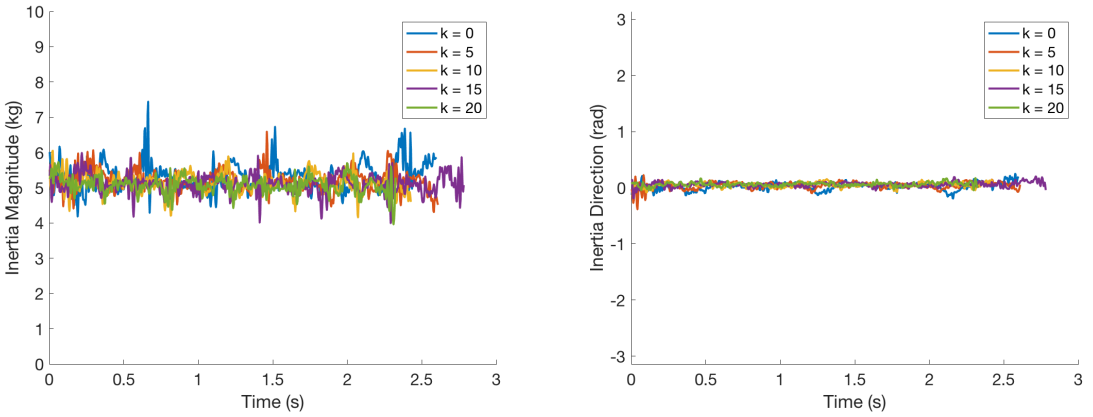

(b) Direction

Figure 5-10: Compensated Inertia, mass=3kg



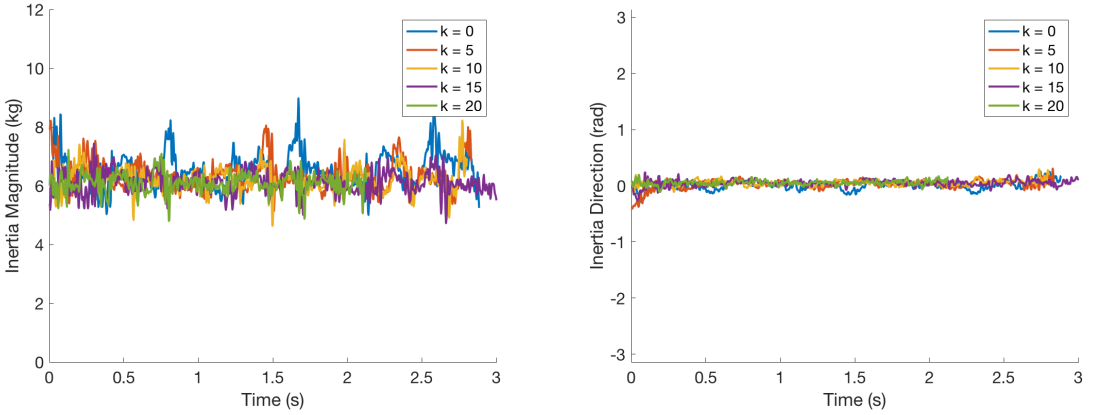

(a) Magnitude (b) Direction

Figure 5-11: Compensated Inertia, mass=4kg



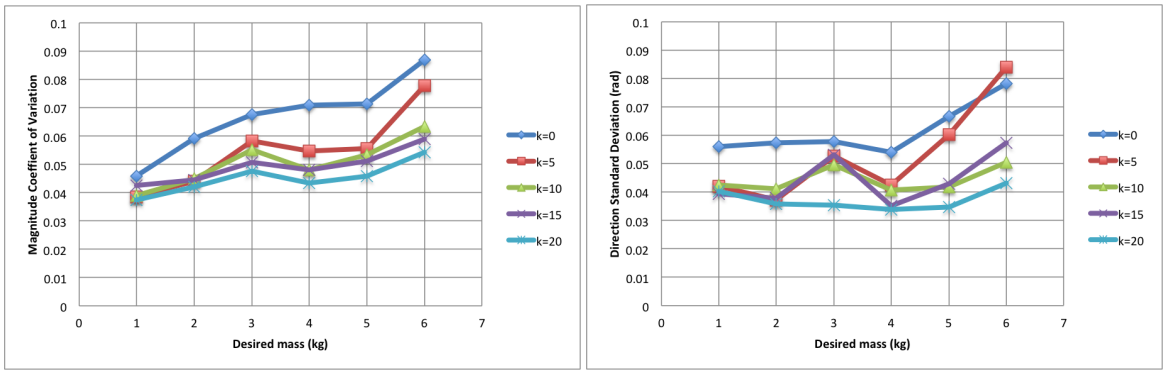

(a) Magnitude (b) Direction

Figure 5-12: Compensated Inertia, mass=5kg




(a) Magnitude (b) Direction

Figure 5-13: Compensated Inertia, mass=6kg




(a) Magnitude Coefficient of Variation (b) Direction Standard Deviation

Figure 5-14: Compensator Error: The inertia magnitude coefficient of variation (left) and the inertia direction standard deviation (right) are reduced with increasing controller gain $k$.

5-15. Plotted in red is the $2\sigma$ inertia compensator error, which represents 95% of the deviation from the given desired inertia. Below a desired inertia of 4kg, the $2\sigma$ inertia compensation error is less than is detectable by a human. Above 4kg, the $2\sigma$ inertia compensation could be detectable by a human. Thus it is recommended that the inertia compensation be used to emulate inertias below 4kg.

The adaptable parameters $a_1$, $a_2$, and $a_3$ are only allowed to change when controller error $s$ exceeds the value of the dead zone $\phi$. Because the dynamic model uncertainty which drives the value of $\phi$ was conservatively chosen, the value of $s$ never exceeded $\phi$ during these trials and the adaptable parameters remained constant.

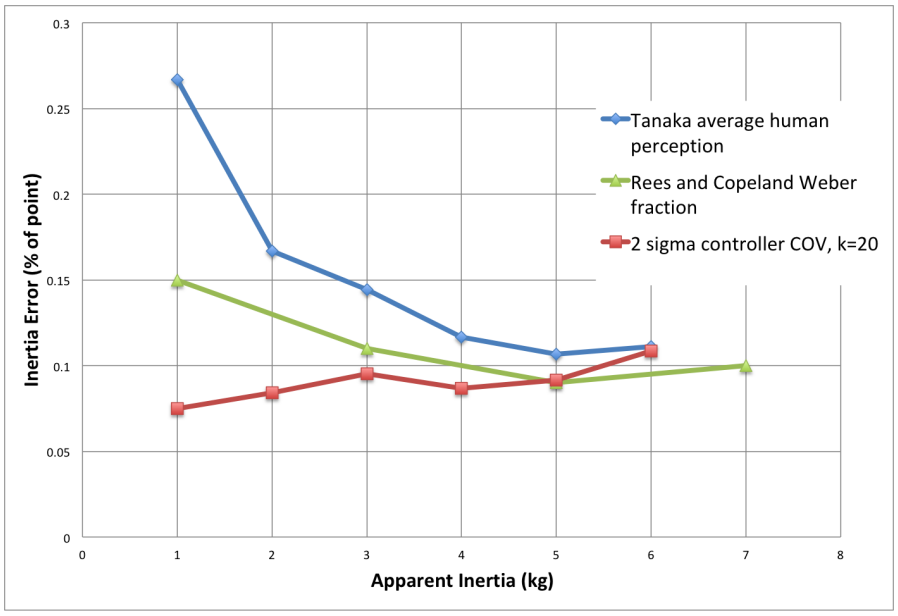



Figure 5-15: Inertia Differential Perception and 2-$\sigma$ Controller Accuracy: The inertia compensator can maintain inertia to the defined (red) error with 95% certainty. The Weber fractions determined by Tanaka et al. (blue) [39] and Rees et al. (green) [41] define the limit for human perception of inertia differences.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

This thesis documented the design and implementation of a unique inertia compensation control scheme implemented on the InMotion2 planar robot. The need for inertia compensation arose from the desire to test the dynamics of human-robot interaction in a manner that isolates the dynamics of the human from the dynamics of the robot. A new suite of LabVIEW-based software was developed to control the robot from a National Instruments compactRIO in a flexible and modern way.

The strategy used for inertia compensation was to model inertia as an admittance, reacting to external forces with a proportional acceleration trajectory. To this end, a position-based controller was designed to force the robot to take on the desired acceleration trajectory. A separate admittance model was created to enforce a virtual constraint in any direction, and this admittance model could be superimposed on the inertia compensation admittance model to create a single desired trajectory for the robot. Sliding mode control was used to design a robust adaptive position tracking controller that achieved the desired admittance.

This controller was successfully implemented on the InMotion2 robot and multiple tests were performed to validate the performance of the inertia compensation as well as tune the feedback gain of the adaptive tracking controller. The inertia compensation allows the robot to move with constant velocity when subjected to an initial condition, both in 2-dimensional space and with the virtual constraint active. Human interaction testing validated the performance of the inertia compensation and the standard deviation of inertia error was found to be below the limit of human perception for modest values of desired inertia.

## 6.2 Future Work

### 6.2.1 Controller Bandwidth

The bandwidth of the inertia compensation is limited by 30Hz 2nd order low-pass filters that are applied to both the measured force and the velocity calculation. The force measurement filter does not require such a low bandwidth; it was set to be consistent with the filters on the velocity and acceleration calculations, described in Section 3.3.1. These filters have such a low bandwidth because of the low resolution on the encoders. There are many methods to deal with calculating velocity from low resolution encoders, many of which are documented in [42]. The 2kHz update time of the controller is fast enough to utilize period measurement rather than frequency measurement, in which the period of time in between increments on the encoder is used to calculate velocity, rather than a fixed time step. This could potentially create a less discretized initial signal which would require less filtering. Other techniques could also be employed to improve the signal such as prediction/estimation. Either a linearized Kalman filter or a nonlinear Luenberger observer could be employed to inform the velocity calculation as well [42]. If any of these techniques are able to

increase the bandwidth of the velocity calculation, the bandwidth of both the inertia compensator and the virtual constraint could be increased to improve accuracy while maintaining system stability.

### 6.2.2 Force Measurement

When processing the measured force at the end-effector for the admittance model, a small dead zone of 0.3 N is applied to the force magnitude, described in Section 4.1.1. The purpose of this dead zone is to ensure that when no actual external force is applied to the end-effector, it commands no acceleration. If there were no dead zone, any slight bias or noise in the force sensor could introduce a nonzero acceleration reference. The inertia compensator also compensates for system friction, so even a minuscule nonzero acceleration reference would cause the robot to slowly move when there are no external forces acting on it. It also adds nonlinear effects to the apparent inertia when forces are low, which may be undesirable. The size of this dead zone was chosen such that the dead zone would zero out the calculated force when accelerating about a virtual constraint, and could potentially be decreased much lower so as to only compensate for forces when the robot is at rest. This would improve the inertia characteristic around zero forces, but would cause nonzero force measurement when accelerating about a virtual constraint.

### 6.2.3 Adaptation

The model uncertainty derived in Section 2.2 sets the size of the adaptation dead-zone $\phi$. It was derived using a combination of steady modeling error from the clamped torque test, and the transient modeling error using the impulse test. The steady error determined in the clamped torque test could be improved by calibration of both the force transducer and the output torque. The transient error determined by comparing the dynamic model with fixed system physical parameters to the dynamic response

of the system. The transient errors are due to some combination of un-modeled dynamics such as gravity and inaccurate friction modeling and physical parameter errors. Because the physical parameter errors are included in the dynamic model uncertainty, it makes the estimation of model uncertainty very conservative. This drives the adaptation dead-zone $\phi$ to be very conservative at a value of 0.05. It was observed in implementation that the magnitude of the sliding error parameter $s$ was always below $\phi$ and therefore the model never was allowed to adapt, shown in Figure 6-1.

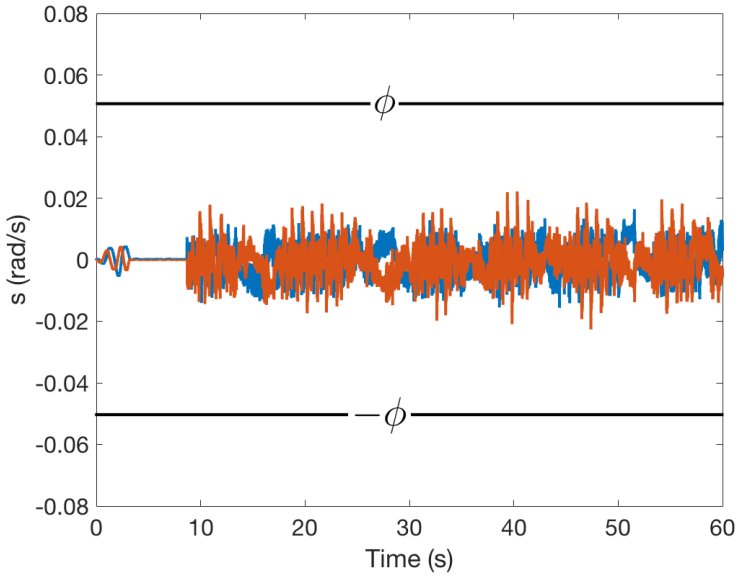



Figure 6-1: Sliding Variable $s$

If the estimation of dynamic modeling error D could be reduced, that would allow the parameter $\phi$ to reduce with it. If the model were to be allowed to adapt to the state of the system it would improve controller accuracy. As it is currently implemented, the adaptation law is not active and therefore the adaptation matrix $P$ has not been tuned. However, to demonstrate that the adaptation law does indeed work, the estimation of model uncertainty was reduced to $D = 0.1Nm$ and adaptation was turned on with a controller gain of $k = 10$. This reduced the adaptation dead-zone

to $\phi = 0.01 rad/s$ which the sliding error sometimes exceeded. The results of this experiment are shown in Figure 6-2. The adaptive parameters moved from their initial values until they hit their respective minimum and maximum limits. This adaptation law is quite aggressive; the adaptation gain matrix $P$ and dead-zone $\phi$ would need to be properly tuned in order for the adaptation to improve error tracking.

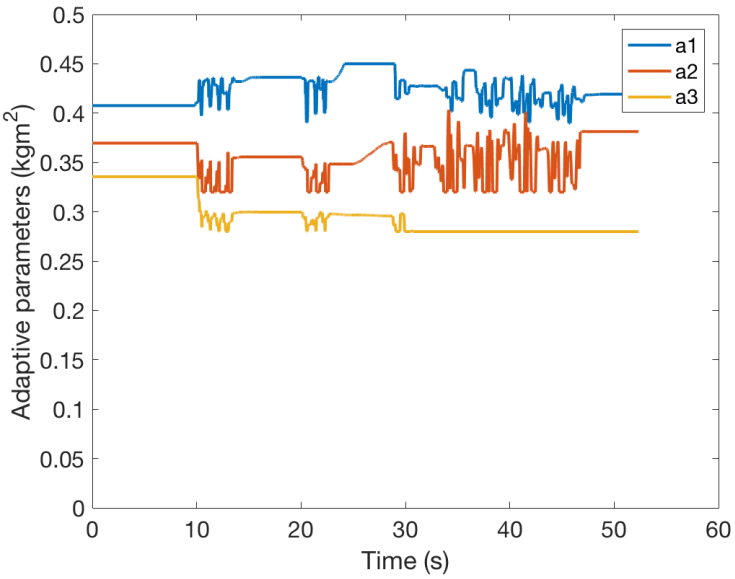



Figure 6-2: Adaptive Parameters

### 6.2.4 Human Testing

Finally, the inertia compensation can be integrated into human-robot interaction testing. Different constraints can be simulated using the flexible $\gamma$ and $r$ parameters described in Section 3.2.1 to change the constraint based on time or position, or even to relax the constraint away. This enables new types of human testing which have never before been possible, and can aid in the quest to gain new insight into human motor control.

# Appendix A

# Operating the InMotion2

## A.1 Running the robot

1. Turn on the ATI Force/Torque Controller by flipping the ON/OFF switch in the back. This step is necessary if you wish to record the end-effector forces or use the inertia compensator.

2. Turn on the compactRIO by pressing the ON/OFF button. The 1st LED will turn green when the cRIO is powered. The 2nd LED will turn yellow while initializing, then turn off when initialization is complete. If this LED blinks 2 or 3 times every few seconds, the cRIO is in safe mode. If this LED is blinking continuously, the software is corrupted and can be fixed by following the instructions in Section A.4.

3. Connect the cRIO to your PC with the USB cable. Look for mitPlanarRobot in NI Measurement & Automation Explorer to verify connection.

4. Switch the red control panel switch from OFF to ON. This powers up the motor servos and encoders and supporting electronics. The "RESET" light should light up red meaning that the motors are shorted out with the break on.

5. Move the robot arm slightly so that both the elbow and shoulder joints move by a few degrees. This initializes the encoders. Before the robot arm is moved, the decoders do not return an accurate position.

6. If changes have been made to the software, run the software with the rotor break on. Take a look at the data to make sure the output torque is acting with the correct magnitude and direction.

7. Press the "RESEST" button so that the red light turns off, indicating that the motors are no longer shorted out without the brake on.

8. Press the "START" button to activate the servos. The green start light should go on. By default the cRIO should command 0 volts when it is on. Do not press "START" while the cRIO is off.

9. Keep the "EMERGENCY STOP" buttons handy. When one of these buttons is pressed, the motor brake is turned on and the servos do not output current to the motors.

10. Open up run.vi in LabVIEW and enter your desired output file name in the "filename" box.

11. Press the run button (white arrow) at the top left of the LabVIEW window. Interact with the robot.

12. Press the STOP button on the LabVIEW front panel to stop the program. This writes the data to a file in the data buffer and sets the motor voltages back to zero.

13. Make sure the cRIO is mapped to the X: drive on the pc. The drive is at `http://172.22.11.2/files/` username: admin, password: imt2.

14. Execute the Matlab file importData.vi to automatically copy the data from the cRIO onto the local computer.

15. Select a file to import into the Matlab workspace when prompted.

16. The data is saved into the Matlabdata folder.

## A.2 Running the inertia compensator

The inertia compensator works by reading the forces applied to the robot end-effector via the force/torque transducers, using an admittance model to convert these forces into a desired trajectory, and implementing an adaptive tracking controller to output motor torques to achieve the desired trajectory.

1. Make sure that the force/torque controller switch is turned on.

2. Start run.vi with the inertia compensation switch turned off.

3. Make sure that the F/T fault (third in sensor fault array) is FALSE. The fault will read TRUE when the force/torque controller is off.

4. Make sure that 0 output torque is being sent to the motors (constraint turned off, compensation turned off).

5. Let the robot sit still for a few seconds with no external forces acting on it.

6. If the ftBias_bool button is on, press the button to turn it off, removing the zero bias on the force/torque signals.

7. Press the ftBias_bool button to turn it on. At the instant this button goes from FALSE to TRUE the zero bias is set to the current force/torque readings.

8. Make sure that the forces in the x and y direction read near zero.

9. Make sure that the inertia compensator gain is reasonable (usually k=10) and mass is reasonable (m between 1kg and 6kg).

10. Grip the robot handle and turn on the inertia compensation by pressing the compensate_bool button. This will turn on unconstrained inertia compensation. If the virtual constraint is desired, make sure the correct impedance parameters are entered and turn on the constraint by pressing the constrain_bool button. This will turn on constrained inertia compensation.

11. If adaptation is desired, press the adapt_bool button.

12. When testing is done, press the "STOP" button on the LabVIEW front panel. This will exit the program gracefully, removing the torque sent to the motors and writing the force zero bias to a file on the cRIO.

## A.3 Modifying LabVIEW code

**RT processor code**

The RT processor is programmed with traditional LabVIEW. The InMotion2 code is separated into a number of functions called subVI's which will be described in detail in Appendix B. SubVI's are useful to isolate functions from each other but can be difficult to change after they are created. The inputs and outputs of a subVI are defined when the subVI is created (highlight code -> Edit -> Create SubVI) and cannot be modified after. If you wish to add or change the inputs or outputs of a subVI you must remake the subVI. Most of the data in the high level code are bundled into clusters of parameters. If anything is added to or removed from one of these clusters, that change has to propagate throughout all the subVI's that use that cluster.

**FPGA code**

The FPGA code is programmed with special LabVIEW functions that get compiled into firmware. These functions use fixed-point representation, in which each number has its data type defined by the following.

1. Signed (+/-) / Unsigned (+): Anything that can go negative should be denoted as Signed.

2. Word length: This is the number of bits used to represent the data. A longer data length enables a higher resolution of data.

3. Integer Word Length: This is the number of bits before the radix point of the data, and defines how the data is scaled. A longer integer word length enables larger numbers to be represented.

The top-level FPGA code "runFPGA.vi" needs to be compiled before running. The recommended method is doing it locally using Xilinx compilation tool. The local compilation takes up a lot of memory (around 2GB) and could take more than an hour. When the compilation is finished it creates a compiled file in the LabVIEW/FPGA Bitfiles folder of the repository. This file can be renamed to whatever is appropriate. To have this FPGA code run from the top-level run.vi, double click the "Configure Open FPGA VI Reference" block on the left of run.vi. Then modify the path of the bitfile and press OK.

## A.4 Fixing a corrupted cRIO

If the status (2nd) LED on the cRIO is blinking continuously, that means that the cRIO software has been corrupted and needs to be reloaded. The instructions to do this are as follows:

1. Name a flash drive NIRECOVERY and format it as FAT32

2. Copy the files from the github NIRECOVERY folder to the flash drive
3. Plug a monitor into the cRIO mini DisplayPort
4. With the cRIO on, plug the flash drive into the cRIO USB slot
5. Press and hold the ON/OFF button to turn the cRIO OFF
6. Push and hold the reset button, keep holding while you press and release the power button, for 5 seconds until the status LED turns on
7. Watch the monitor until it says the installation was successful - if there's an error just start the process over
8. Remove the flash drive and press the reset button. It should go into safe mode in which software can be loaded via the pc.
9. When the cRIO is in safe mode, the status LED should be blinking 2 or 3 times every few seconds.
10. Plug USB cable into your PC and load up NI Measurement & Automation Explorer
11. Install the software using instructions at `https://www.ni.com/getting-started/set-up-hardware/compactrio/controller-software`
12. cRIO username: admin password: imt2
13. Make sure to select "System State Publisher" in addition to the default software.

# Appendix B

# Software Description

In order to run the robot, the user only needs to interact with one file, run.vi. This file runs on the real time (RT) processor and executes all other functions. This appendix contains a list of all logged parameters, as well as a comprehensive description of each function for both the RT processor and FPGA code.

## B.1 Parameter Lists

The data that gets logged is organized into 5 clusters. Within the code, the names of parameters within an array values are separated by brackets. For instance the array "pos[S,E,Q]_raw" translates into three parameters for the data log: posS_raw, posE_raw, and posQ_raw. Table B.1 displays the variables in the "userInput" cluster, set in run.vi. Table B.2 displays the variables in the "control" cluster, set in the high-level control function bundle.vi. Table B.3 displays the variables in the "outputs" cluster, set in outFPGA.vi. Table B.4 displays the variables in the "inputs" cluster, set in inFPGA.vi. Table B.5 displays the variables in the "inertiaVals" cluster, set in outFPGA.vi.

Table B.1: User Inputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| mDesired_kg | Desired mass, kg | refXyFPGA.vi |
| k_Nmsprad | Inertia compensator feedback gain, Nms/rad | finalFPGA.vi, errorFPGA.vi |
| p_kgmms | Adaptation law gain, $kgm^2s$ | a-lawFPGA.vi |
| compensate_bool | Inertia compensation on/off button | outFPGA.vi refXyFPGA.vi |
| adapt_bool | Adaptation on/off button | adaptFPGA.vi |
| damping_Nspm | Virtual constraint damping, Ns/m | adOrthoFPGA.vi |
| stiffness_Npm | Virtual constraint stiffness, N/m | adOrthoFPGA.vi |
| radius_m | Virtual constraint circle radius, meters | circle.vi |
| constraint_bool | Virtual constraint on/off button | refXyFPGA.vi |

Table B.2: Control Inputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| forces_bool | True if forces are specified<br>False if torques are specified | outFPGA.vi |
| controllerOutput1 | X force if forces_bool=ON, N<br>Torque 1 if forces_bool=OFF, Nm | outFPGA.vi |
| controllerOutput2 | Y force if forces_bool=ON, N<br>Torque 2 if forces_bool=OFF, Nm | outFPGA.vi |
| gamma_rad | Virtual constraint angle, rad | vProjFPGA.vi |
| r_meters | Orthogonal distance to constraint, m | adOrthoFPGA.vi |

Table B.3: Output Parameters

| **Parameter** | **Description** | **Source** |
|---|---|---|
| trqOutS_Nm | commanded torque $\tau_{in,1}$, Nm | outFPGA.vi |
| trqOutE_Nm | commanded torque $\tau_{in,2}$, Nm | outFPGA.vi |
| voltsS_V | shoulder output voltage, V | motorFPGA.vi |
| voltsE_V | elbow output voltage, V | motorFPGA.vi |

Table B.4: Input Parameters

| **Parameter** | **Description** | **Source** |
|---|---|---|
| rateFPGA_Hz | FPGA update rate, Hz | timeFPGA.vi |
| posS_raw | joint position $\theta_1$, raw bits | decodeFPGA.vi |
| posE_raw | joint position $\theta_2$, raw bits | decodeFPGA.vi |
| posQ_raw | decoder sequence number, binary | decodeFPGA.vi |
| posS_rad | measured position of $\theta_1$, radians | radFPGA.vi |
| posE_rad | measured position of $\theta_2$, radians | radFPGA.vi |
| posX_m | end-effector X position, meters | meterFPGA.vi |
| posY_m | end-effector Y position, meters | meterFPGA.vi |
| jacobianl1sinS_m | Jacobian matrix term: $l_1 \sin\theta_1$, meters | meterFPGA.vi |
| jacobianl1cosS_m | Jacobian matrix term: $l_1 \cos\theta_1$, meters | meterFPGA.vi |
| jacobianl2sinE_m | Jacobian matrix term: $l_2 \sin\theta_2$, meters | meterFPGA.vi |
| jacobianl2cosE_m | Jacobian matrix term: $l_2 \cos\theta_2$, meters | meterFPGA.vi |
| velX_mps | end-effector X velocity, m/s | velFPGA.vi |
| velY_mps | end-effector Y velocity, m/s | velFPGA.vi |
| velS_rps | joint velocity $\dot{\theta_1}$, rad/s | velFPGA.vi |
| velE_rps | joint velocity $\dot{\theta_2}$, rad/s | velFPGA.vi |
| accX_mpss | end-effector X acceleration, $m/s^2$ | velFPGA.vi |
| accY_mpss | end-effector Y acceleration, $m/s^2$ | velFPGA.vi |
| faultS_bool | Encoder fault on $\theta_1$, binary | faultsFPGA.vi |
| faultE_bool | Encoder fault on $\theta_2$, binary | faultsFPGA.vi |
| faultF_bool | Force/Torque reading fault, binary | faultsFPGA.vi |
| ftRawX_N | Raw force in X direction, N | ftBiasFPGA.vi |
| ftRawY_N | Raw force in Y direction, N | ftBiasFPGA.vi |
| ftRawZ_N | Raw force in Z direction, N | ftBiasFPGA.vi |
| ftRawX_Nm | Raw torque about X axis, Nm | ftBiasFPGA.vi |
| ftRawY_Nm | Raw torque about X axis, Nm | ftBiasFPGA.vi |
| ftRawZ_Nm | Raw torque about X axis, Nm | ftBiasFPGA.vi |
| ftX_N | Force in absolute X direction, N | ftFPGA.vi |
| ftY_N | Force in absolute Y direction, N | ftFPGA.vi |
| ftZ_N | Force in absolute Z direction, N | ftFPGA.vi |
| ftX_Nm | Torque about absolute X axis, Nm | ftFPGA.vi |
| ftY_Nm | Torque about absolute X axis, Nm | ftFPGA.vi |
| ftZ_Nm | Torque about absolute X axis, Nm | ftFPGA.vi |
| ftBiasX_N | Force bias in X direction, N | ftBiasFPGA.vi |
| ftBiasY_N | Force bias in Y direction, N | ftBiasFPGA.vi |
| ftBiasZ_N | Force bias in Z direction, N | ftBiasFPGA.vi |
| ftBiasX_Nm | Torque bias about X axis, Nm | ftBiasFPGA.vi |
| ftBiasY_Nm | Torque bias about X axis, Nm | ftBiasFPGA.vi |
| ftBiasZ_Nm | Torque bias about X axis, Nm | ftBiasFPGA.vi |

Table B.5: Inertia Compensation Values

| **Parameter** | **Description** | **Source** |
|---|---|---|
| inertiaTorqueS_Nm | inertia compensator torque $\tau_{in,1}$, Nm | finalFPGA.vi |
| inertiaTorqueE_Nm | inertia compensator torque $\tau_{in,2}$, Nm | finalFPGA.vi |
| sS_rps | Sliding error $s_1$, rad/s | errorFPGA.vi |
| sE_rps | Sliding error $s_2$, rad/s | errorFPGA.vi |
| thetaDesiredDDotS_rpss | Desired joint acceleration $\ddot{\theta}_{d,1}$, $rad/s^2$ | refAngleFPGA.vi |
| thetaDesiredDDotE_rpss | Desired joint acceleration $\ddot{\theta}_{d,2}$, $rad/s^2$ | refAngleFPGA.vi |
| thetaDesiredDotS_rps | Desired joint velocity $\dot{\theta}_{d,1}$, $rad/s$ | refAngleFPGA.vi |
| thetaDesiredDotE_rps | Desired joint velocity $\dot{\theta}_{d,2}$, $rad/s$ | refAngleFPGA.vi |
| thetaDesiredS_rad | Desired joint position $\theta_{d,1}$, rad | refAngleFPGA.vi |
| thetaDesiredE_rad | Desired joint position $\theta_{d,2}$, rad | refAngleFPGA.vi |
| thetaRDDotS_rpss | Reference joint acceleration $\ddot{\theta}_{r,1}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDDotE_rpss | Reference joint acceleration $\ddot{\theta}_{r,2}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDotS_rps | Reference joint velocity $\dot{\theta}_{r,1}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDotE_rps | Reference joint velocity $\dot{\theta}_{r,2}$, $rad/s^2$ | errorFPGA.vi |
| YaS_Nm | Shoulder Y*a, Nm | YaFPGA.vi |
| YaE_Nm | Elbow Y*a, Nm | YaFPGA.vi |
| a1_kgmm | Adaptive parameter $a_1$, $kgm^2$ | adaptFPGA.vi |
| a2_kgmm | Adaptive parameter $a_2$, $kgm^2$ | adaptFPGA.vi |
| a3_kgmm | Adaptive parameter $a_3$, $kgm^2$ | adaptFPGA.vi |
| humanTrqS_Nm | Human force in joint torques $\tau_{h,1}$, Nm | humanFPGA.vi |
| humanTrqE_Nm | Human force in joint torques $\tau_{h,2}$, Nm | humanFPGA.vi |
| xyDesiredDDotX_mpss | Desired acceleration $\ddot{\theta}_{d,1}$, $m/s^2$ | refXyFPGA.vi |
| xyDesiredDDotY_mpss | Desired acceleration $\ddot{\theta}_{d,2}$, $m/s^2$ | refXyFPGA.vi |
| xyDesiredDotX_mps | Desired velocity $\dot{\theta}_{d,1}$, $m/s$ | refXyFPGA.vi |
| xyDesiredDotY_mps | Desired velocity $\dot{\theta}_{d,2}$, $m/s$ | refXyFPGA.vi |
| humanForceX_N | Human force $f_x$, N | humanFPGA.vi |
| humanForceY_N | Human force $f_y$, N | humanFPGA.vi |
| frictionS_Nm | Estimated joint friction $F_{f,1}$, Nm | frictionFPGA.vi |
| frictionE_Nm | Estimated joint friction $F_{f,2}$, Nm | frictionFPGA.vi |
| fh_N | Magnitude of human force, N | humanFPGA.vi |
| beta_pirad | Angle of human force, $\pi$ rad | humanFPGA.vi |
| adGamma_mpss | Desired acc. along constraint, $m/s^2$ | refXyFPGA.vi |
| adOrtho_mpss | Desired acc. ortho. to constraint, $m/s^2$ | adOrthoFPGA.vi |
| vdGamma_mps | Desired vel. along constraint, m/s | refXyFPGA.vi |
| vdOrtho_mps | Desired vel. ortho. to constraint, m/s | refXyFPGA.vi |

## B.2 RT processor code

**run.vi**

The function run.vi contains a flat sequence structure consisting of 3 stages to be run in order. The first stage opens the compiled FPGA code, uses fetchFTbias.vi to read the FT bias from a cRIO file, initializes the FT bias in the FPGA code, and runs the FPGA code. It also draws a circle on the front panel indicator and executes buffer.vi which forms the parameters for the data buffer. The second stage implements a timed loop at 1kHz until the user presses the STOP button. This timed loop reads the inputs from the FPGA, runs limit.vi on the user inputs, runs circle.vi, and sets the high-level control parameters in the FPGA. All 5 data clusters are sent to logData.vi to be written to a FIFO data buffer. In parallel to the timed loop, a while loop executes display.vi which puts the location of the end-effector on the front panel. In parallel to this, another while loop writes the parameters from the FIFO data buffer into a TDMS file on the cRIO. The third stage closes the FPGA program, runs storeFTbias.vi to save the FT bias to a file on the cRIO, and closes the TDMS file.

Table B.6: run.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| filename | Name of output file, without extension | Front Panel |
| STOP | Gracefully exits the control | Front Panel |
| ftBias_bool | Apply F/T bias, taken at rising edge | Front Panel |
| userInput | Described in Table B.1 | Front Panel |

Table B.7: run.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| faultS_bool | Encoder 1 reading fault | Front Panel |
| faultE_bool | Encoder 2 reading fault | Front Panel |
| faultF_bool | Force/torque reading fault | Front Panel |

### fetchFTbias.vi

The function fetchFTbias.vi reads the file on the cRIO at "/c/bias/bias.txt" and parses the 6 values to be used as a zero bias for the force/torque transducer reading.

Table B.8: fetchFTbias.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| initBias | F/T zero bias from file on cRIO | ftBiasFPGA.vi |

### buffer.vi

The function buffer.vi instantiates the size of the data logging buffer as well as the parameter names that are logged by looping through each element of each data cluster and reading the data labels. Note that the input clusters do not need to be hooked up, as the function only depends on the data type, and not the actual data. If any of the data clusters change, this function must be changed.

Table B.9: buffer.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| inputs | Described in Table B.4 | N/A |
| outputs | Described in Table B.3 | N/A |
| control | Described in Table B.2 | N/A |
| userInput | Described in Table B.1 | N/A |
| inertiaVals | Described in Table B.5 | N/A |
| EMG | 16 differential EMG voltages | N/A |

### limit.vi

The function limit.vi will limit the values of stiffness and damping to be within the stability region. It also will slowly ramp the stiffness from 0 to the full value when the virtual constraint is turned on. This keeps allows the robot end-effector to slowly move towards the virtual constraint rather than initializing with a very large error.

Table B.10: limit.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| userInput | Described in Table B.1 | Front Panel |

Table B.11: limit.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| limitedInput | userInput with limited values | circle.vi<br>adOrthoFPGA.vi |

**circle.vi**

When inertia compensation is off, the output processor uses the output force and/or torque commands from the "control" cluster that are set in this function. Based off the stiffness, damping, and radius defined in the "userInput" cluster, this function calculates the required forces in the x and y direction to simulate a circular virtual constraint, as defined in Equation 3.3. When inertia compensation is turned on, the output processor uses the angle of the virtual constraint $\gamma$ and the orthogonal distance to the constraint $r$ defined in Section 3.2.1. This function calculates $\gamma$ and $r$ based on Equation 3.4 for the circular constraint. The function contains divide by zero protection and finally feeds the high-level control variables $F_x$, $F_y$, $\gamma$, and $r$ into bundle.vi to be converted to a cluster of fixed-point variables.

Table B.12: circle.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| userInput | Described in Table B.1 | Front Panel |
| inputs | Described in Table B.4 | inFPGA.vi |

Table B.13: circle.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| control | Described in Table B.2 | outFPGA.vi |

### bundle.vi

The function bundle.vi takes the outputs from the high-level controller, converts them to fixed-point representation for the FPGA, and bundles them into the control bundle.

Table B.14: bundle.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| forces_bool | True if forces are specified<br>False if torques are specified | circle.vi |
| controllerOutput1 | X force if forces_bool=ON, N<br>Torque 1 if forces_bool=OFF, Nm | circle.vi |
| controllerOutput2 | Y force if forces_bool=ON, N<br>Torque 2 if forces_bool=OFF, Nm | circle.vi |
| gamma_rad | Virtual constraint angle, rad | circle.vi |
| r_meters | Orthogonal distance to constraint, m | circle.vi |

Table B.15: bundle.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| control | Described in Table B.2 | outFPGA.vi |

### logData.vi

The function logData.vi parses the data clusters listed in Table B.16, converts the data to floating point doubles, assembles them into a large array, and writes them to a FIFO data buffer. If any of these clusters change, this function as well as buffer.vi must be updated with the new data types.

### display.vi

The function display.vi reads in a picture pic_in, and draws a circle on that picture that corresponds to the location of the end-effector in x and y coordinates.

Table B.16: logData.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |
| outputs | Described in Table B.3 | outFPGA.vi |
| control | Described in Table B.2 | bundle.vi |
| userInput | Described in Table B.1 | run.vi |
| inertiaVals | Described in Table B.5 | outFPGA.vi |
| EMG | 16 differential EMG voltages | EMG.vi |

Table B.17: display.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |
| pic_in | Picture on which to draw point | run.vi |

Table B.18: display.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| pic_out | Picture with point drawn on it | run.vi |

**storeFTbias.vi**

The function storeFTbias.vi reads the bias required zero out the force transducer reading in all 6 coordinates (linear x, y, z, angular x, y, z) and writes them to a file on the cRIO at "/c/bias/bias.txt".

Table B.19: storeFTbias.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |

## B.3 FPGA code

The following functions must be compiled when any changes are made. Compiling runFPGA.vi will create an FPGA bitfile which can be called by run.vi.

**runFPGA.vi**

The function runFPGA.vi contains a flat sequence structure consisting of 3 stages to be run in order. The first stage runs initFPGA.vi to excite the force/torque health reading. The second stage contains a while loop that runs on a $500\mu s$ timer which reads in the inputs from run.vi as well as a local controller dictating which force/torque signals to read and executes both the input processing program and then the output processing program. Parallel to this process, timeFPGA.vi is timing the execution of the while loop and a stop button controlled from run.vi can terminate the loop. The third stage sends a signal of 0 volts to the motors in motorFPGA.vi, so that the motors stop exerting torque when the controller is stopped.

Table B.20: runFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| control | Described in Table B.2 | bundle.vi |
| ftBias_bool | Apply F/T bias, taken at rising edge | run.vi |
| whichForcesFX_bool | Whether to read the FX sensor | Front Panel |
| whichForcesFY_bool | Whether to read the FY sensor | Front Panel |
| whichForcesFZ_bool | Whether to read the FZ sensor | Front Panel |
| whichForcesTX_bool | Whether to read the TX sensor | Front Panel |
| whichForcesTY_bool | Whether to read the TY sensor | Front Panel |
| whichForcesTZ_bool | Whether to read the TZ sensor | Front Panel |
| initBias | F/T zero bias from file on cRIO | fetchFTbias.vi |

Table B.21: runFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| inputs | Described in Table B.4 | logData.vi<br>circle.vi<br>display.vi<br>storeFTbias.vi |
| outputs | Described in Table B.3 | logData |
| inertiaVals | Described in Table B.5 | logData |

### initFPGA.vi

The function initFPGA.vi contains any outputs that need to be initialized before the start of the FPGA loop. The only function that needs to be initialized is the force/torque transducer health bit. The F/T controller processes the measured force/torque sensor values by converting them to digital representation, multiplying them by a calibration matrix, and converting back to analog. If there is an error in this conversion the F/T controller detects this and opens the health circuit [26]. The health circuit also opens when the F/T controller is off. In order to detect whether this circuit is open or closed, the cRIO continuously energizes the high side of the ATI switch (D-12) via the discrete output 9403-4 (DIO3) in this function. The cRIO reads the open/closed status via the voltage across a resistor in faultsFPGA.vi.

### inFGPA.vi

The function inFPGA.vi reads the sensor inputs and performs all necessary processing before bundling the results into the input cluster. To process the encoder measurement, it runs decodeFPGA.vi, radFPGA.vi, meterFPGA.vi, and velFPGA.vi. To process the force/torque measurement it runs ftFPGA.vi. To process the encoder and force/torque faults it runs faultsFPGA.vi, and to measure the time step it runs timeFPGA.vi.

Table B.22: inFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| whichForcesFX_bool | TRUE = measure X force | runFGPA.vi |
| whichForcesFY_bool | TRUE = measure Y force | runFGPA.vi |
| whichForcesFZ_bool | TRUE = measure Z force | runFGPA.vi |
| whichForcesTX_bool | TRUE = measure X torque | runFGPA.vi |
| whichForcesTY_bool | TRUE = measure Y torque | runFGPA.vi |
| whichForcesTZ_bool | TRUE = measure Z torque | runFGPA.vi |
| ftBias_bool | Apply F/T bias, taken at rising edge | run.vi |
| initBias | F/T zero bias from file on cRIO | fetchFTbias.vi |

Table B.23: inFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| inputs | Described in Table B.4 | logData.vi |

**decodeFGPA.vi**

The function decodeFPGA.vi reads the encoder position by communicating with the Gurley Precision interpolating decoder via discrete queries as described in Sections 2.1.2 and 3.3.1. The sequence of queries alternates each time step in order to minimize the number of operations each time step. The location of encoder 1 (the shoulder motor) is queried by hold bit (DOUT0), and query bits A0 (DOUT1) and A1 (DOUT2). The location of encoder 2 (the elbow motor) is queried by hold bit (DOUT8), and query bits A0 (DOUT9) and A1 (DOUT10). The 16-bit encoder position is communicated through three separate readings of the low, middle, and high bits over 8 discrete lines. When the hold bit is activated at the beginning of the query, the decoder holds the encoder reading. For the first of the alternating sequences, the low bits are read first, then the high bits, then the middle. For the other set of alternating sequences, the middle bits are read first, then the high bits, then the low. After the sequence is over, the low, middle, and high bits are concatenated into the 16-bit word and converted to numeric positions.

Table B.24: decodeFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| posS_raw | Position of encoder 1 (shoulder motor) | radFPGA.vi |
| posE_raw | Position of encoder 2 (elbow motor) | radFPGA.vi |
| posQ_raw | Sequence number (0 or 1) | radFPGA.vi |

**radFPGA.vi**

The function radFPGA.vi takes in the encoder positions and runs a prediction scheme in predictFPGA.vi to detect faults in the encoder readings. When the system is turned

on the encoders each read 0 until the robot is moved and the encoders initialize. The linkages on the robot prevent the angles $\theta_1$ and $\theta_2$ from equaling each other, so encoder faults are also flagged if the measured values equal. Then scalars and adders are applied to the raw decoder values to convert them to radians, and normFPGA.vi normalizes the encoder position and convert to pi radians, which is a special unit used by LabVIEW FPGA code.

Table B.25: radFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| posS_raw | Position of encoder 1 (shoulder motor), raw | radFPGA.vi |
| posE_raw | Position of encoder 2 (elbow motor), raw | radFPGA.vi |
| posQ_raw | Sequence number (0 or 1) | radFPGA.vi |

Table B.26: radFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| faultS_bool | Fault detected on encoder 1 | faultsFPGA.vi |
| faultE_bool | Fault detected on encoder 2 | faultsFPGA.vi |
| posS_rad | Position of encoder 1 (shoulder motor), rad | velFPGA.vi |
| posE_rad | Position of encoder 2 (elbow motor), rad | velFPGA.vi |
| posS_pirad | Position of encoder 1 (shoulder motor), pi rad | velFPGA.vi |
| posE_pirad | Position of encoder 2 (elbow motor), pi rad | velFPGA.vi |

### predictFPGA.vi

The function predictFPGA.vi takes in the raw encoder measurement and predicts the current position by adding the previous encoder position to the delta between the previous two encoder positions, as described in Equation 3.5. If this predicted position differs from the measured position by more than 50 bits, it sets the encoder fault and uses the projected position rather than the measured position.

Table B.27: predictFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| decoder | Measured position from encoder, bits | decodeFPGA.vi |

Table B.28: predictFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| fault | Fault detected on encoder | faultsFPGA.vi |
| position | Corrected position from encoder, bits | velFPGA.vi |

### deltaFPGA.vi

The function deltaFPGA.vi takes the difference between two encoder measurements, taking into account the rollover of encoder measurements between bits 65535 and 0.

Table B.29: deltaFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| current | Position from encoder, bits | predictFPGA.vi |
| past | Position from encoder, bits | predictFPGA.vi |

Table B.30: deltaFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| delta | Difference between encoder measurements, bits | predictFPGA.vi |

### addFPGA.vi

The function addFPGA.vi adds two encoder measurements together, rolling over after bit 65535.

Table B.31: addFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| decoderVal1 | Position from encoder, bits | predictFPGA.vi |
| decoderVal2 | Position from encoder, bits | predictFPGA.vi |

Table B.32: addFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| x+y | Sum of encoder measurements, bits | predictFPGA.vi |

### normFPGA.vi

The function normFPGA.vi makes sure that the encoder measurement is between 0 and 65535. It also converts to pi radians, which are radians divided by pi for use in the LabVIEW FPGA trigonometric functions.

Table B.33: normFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| radians | Encoder position, radians | radFPGA.vi |

Table B.34: normFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| normalized radians | Encoder position, radians | radFPGA.vi |
| pi radians | Encoder position, pi radians | radFPGA.vi |

### meterFPGA.vi

The function meterFPGA.vi takes in the encoder positions in radians and uses the link lengths $l_1$ and $l_2$ to calculate the end-effector position in Cartesian coordinates as well as the end-effector Jacobian according to Equation 2.15.

Table B.35: meterFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| posS_pirad | Encoder 1 position, pi radians | radFPGA.vi |
| posE_pirad | Encoder 2 position, pi radians | radFPGA.vi |

Table B.36: meterFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| posX_m | End-effector X position, meters | logFPGA.vi |
| posY_m | End-effector Y position, meters | logFPGA.vi |
| jacobianl1sinS_m | Jacobian term $l_1 \sin(\theta_1)$, m | outFPGA.vi |
| jacobianl1cosS_m | Jacobian term $l_1 \cos(\theta_1)$, m | outFPGA.vi |
| jacobianl2sinE_m | Jacobian term $l_2 \sin(\theta_2)$, m | outFPGA.vi |
| jacobianl2cosE_m | Jacobian term $l_2 \cos(\theta_2)$, m | outFPGA.vi |

### velFPGA.vi

The function velFPGA.vi takes the Euler derivative of the measured encoder positions in radians using derivFPGA.vi, then filters the result with a 2nd order Butterworth filter at a bandwidth of 30Hz to calculate radial velocity. Using the radial velocity and the end-effector Jacobian, the velocity of the end-effector is calculated in Cartesian coordinates using Equation 2.15. Finally, the Euler derivative of the end-effector velocity is calculated and filtered with a 4th order Butterworth low-pass filter at a bandwidth of 50Hz to calculate end-effector acceleration.

Table B.37: velFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| posS_rad | Encoder 1 position, radians | radFPGA.vi |
| posE_rad | Encoder 2 position, radians | radFPGA.vi |
| jacobianl1sinS_m | Jacobian term $l_1 \sin(\theta_1)$, m | meterFPGA.vi |
| jacobianl1cosS_m | Jacobian term $l_1 \cos(\theta_1)$, m | meterFPGA.vi |
| jacobianl2sinE_m | Jacobian term $l_2 \sin(\theta_2)$, m | meterFPGA.vi |
| jacobianl2cosE_m | Jacobian term $l_2 \cos(\theta_2)$, m | meterFPGA.vi |
| rateFPGA_Hz | FPGA time step, Hz | timeFPGA.vi |

Table B.38: velFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| velS_rps | Encoder 1 velocity, rad/s | inertiaFPGA.vi |
| velE_rps | Encoder 2 velocity, rad/s | inertiaFPGA.vi |
| velX_mps | End effector X velocity, m/s | refXyFPGA.vi |
| velY_mps | End effector Y velocity, m/s | refXyFPGA.vi |
| accX_mpss | End effector X acceleration, $m/s^2$ | humanFPGA.vi |
| accY_mpss | End effector Y acceleration, $m/s^2$ | humanFPGA.vi |

### derivativeFPGA.vi

The function derivativeFPGA.vi calculates the difference between two position measurements for use in a derivative calculation. Because the joint positions are expressed as a radial angle between 0 and $2\pi$ there is the potential for a joint to move from $0.99\pi$

to 0. A simple subtraction would yield $-0.99\pi$ where this function takes the overflow into account by changing the direction of subtraction depending on the quadrants of each parameter.

Table B.39: derivativeFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| current | First encoder position, rad | velFPGA.vi |
| prev | Previous encoder position, rad | velFPGA.vi |

Table B.40: derivativeFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| deriv | Difference, rad | velFPGA.vi |

### ftFPGA.vi

The function ftFPGA.vi reads and processes the inputs from the force/torque transducer. The function readFPGA.vi reads the appropriate force/torque inputs, ftFilterFPGA.vi filters the signals, ftBiasFPGA.vi applies a bias to the signals to zero them out when no forces are applied, and coordFPGA.vi converts the forces and torques relative to the axis of the transducer to the global coordinate system.

Table B.41: ftFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| whichForcesFX_bool | Whether to read the FX sensor | runFPGA.vi |
| whichForcesFY_bool | Whether to read the FY sensor | runFPGA.vi |
| whichForcesFZ_bool | Whether to read the FZ sensor | runFPGA.vi |
| whichForcesTX_bool | Whether to read the TX sensor | runFPGA.vi |
| whichForcesTY_bool | Whether to read the TY sensor | runFPGA.vi |
| whichForcesTZ_bool | Whether to read the TZ sensor | runFPGA.vi |
| ftBias_bool | Apply F/T bias, taken at rising edge | run.vi |
| initBias | F/T zero bias from file on cRIO | fetchFTbias.vi |
| posS_rad | Encoder 1 position, radians | radFPGA.vi |
| posE_rad | Encoder 2 position, radians | radFPGA.vi |

Table B.42: ftFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| ftRawX_N | Raw force transducer X force, N | filterFGPA.vi |
| ftRawY_N | Raw force transducer Y force, N | filterFGPA.vi |
| ftRawZ_N | Raw force transducer Z force, N | filterFGPA.vi |
| ftRawX_Nm | Raw force transducer X torque, Nm | filterFGPA.vi |
| ftRawY_Nm | Raw force transducer Y torque, Nm | filterFGPA.vi |
| ftRawZ_Nm | Raw force transducer Z torque, Nm | filterFGPA.vi |
| ftBiasX_N | Force bias in X direction, N | logData.vi |
| ftBiasY_N | Force bias in Y direction, N | logData.vi |
| ftBiasZ_N | Force bias in Z direction, N | logData.vi |
| ftBiasX_Nm | Torque bias about X axis, Nm | logData.vi |
| ftBiasY_Nm | Torque bias about X axis, Nm | logData.vi |
| ftBiasZ_Nm | Torque bias about X axis, Nm | logData.vi |
| ftX_N | Force in absolute X direction, N | humanFPGA.vi |
| ftY_N | Force in absolute Y direction, N | humanFPGA.vi |
| ftZ_N | Force in absolute Z direction, N | humanFPGA.vi |
| ftX_Nm | Torque about absolute X axis, Nm | humanFPGA.vi |
| ftY_Nm | Torque about absolute X axis, Nm | humanFPGA.vi |
| ftZ_Nm | Torque about absolute X axis, Nm | humanFPGA.vi |

**readFPGA.vi**

The function readFPGA.vi reads the force/torque transducer signal inputs according to which bits in the whichForces array are active. Then the forces and torques are multiplied by a simple scalar to convert the signals to Newtons and Newton-meters, respectively according to the conversion factor in Table 2.4. The forces and torques are then multiplied by scalars to convert to SI units.

Table B.43: readFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| whichForcesFX_bool | Whether to read the FX sensor | runFPGA.vi |
| whichForcesFY_bool | Whether to read the FY sensor | runFPGA.vi |
| whichForcesFZ_bool | Whether to read the FZ sensor | runFPGA.vi |
| whichForcesTX_bool | Whether to read the TX sensor | runFPGA.vi |
| whichForcesTY_bool | Whether to read the TY sensor | runFPGA.vi |
| whichForcesTZ_bool | Whether to read the TZ sensor | runFPGA.vi |

Table B.44: readFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| ftRawX_N | Raw force transducer X force, N | filterFGPA.vi |
| ftRawY_N | Raw force transducer Y force, N | filterFGPA.vi |
| ftRawZ_N | Raw force transducer Z force, N | filterFGPA.vi |
| ftRawX_Nm | Raw force transducer X torque, Nm | filterFGPA.vi |
| ftRawY_Nm | Raw force transducer Y torque, Nm | filterFGPA.vi |
| ftRawZ_Nm | Raw force transducer Z torque, Nm | filterFGPA.vi |

**filterFGPA.vi**

The function ftFilterFPGA.vi applies a 2nd order Butterworth low-pass filter to the raw force signals with a cutoff frequency of 30Hz.

Table B.45: ftFilterFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| ftRawX_N | Raw force transducer X force, N | readFPGA.vi |
| ftRawY_N | Raw force transducer Y force, N | readFPGA.vi |
| ftRawZ_N | Raw force transducer Z force, N | readFPGA.vi |
| ftRawX_Nm | Raw force transducer X torque, Nm | readFPGA.vi |
| ftRawY_Nm | Raw force transducer Y torque, Nm | readFPGA.vi |
| ftRawZ_Nm | Raw force transducer Z torque, Nm | readFPGA.vi |

Table B.46: filterFPGA Outputs

| Parameter | Description | Destination |
|---|---|---|
| forceFiltered | Filtered force transducer array, N(m) | ftBiasFGPA.vi |

**ftBiasFPGA.vi**

The function ftBiasFPGA.vi applies a bias to the force/torque reading to zero it out when there are no external forces or torques applied. If the ftBias boolean is false, no bias is applied to the signals. When the ftBias boolean is true, the ftBias values are subtracted from the filtered force reading. When the ftBias boolean is turned from off to on, the ftBias values are set to the measured force/torques measured at that

instant. The ftBias values are initialized to the values stored in the cRIO file via the initBias parameter.

Table B.47: ftBiasFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| forceFiltered | Filtered force transducer array, N, Nm | filterFGPA.vi |
| initBias | F/T zero bias from file on cRIO | fetchFTbias.vi |
| ftBias_bool | Apply F/T bias, taken at rising edge | run.vi |

Table B.48: ftBiasFPGA Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| forceRel_N(m) | Zeroed force relative to F/T transducer, N(m) | coordFPGA.vi |

### coordFPGA.vi

The function coordFPGA.vi converts the forces and torques from coordinates relative to the force/torque transducer to global coordinates. The force/torque transducer is mounted at the end of link 2, which is at an angle $\theta_2$ relative to the global coordinate frame. Therefore the angle between the relative and global coordinate systems is the angle $\theta_2$ plus a constant offset. This angle is used to project the forces and torques from relative coordinate system to the global coordinate system using Equation 2.16.

Table B.49: coordFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| posS_rad | Encoder 1 (shoulder motor) position, rad | radFGPA.vi |
| posE_rad | Encoder 2 (elbow motor) position, rad | radFGPA.vi |
| forceRel_N(m) | Zeroed force relative to F/T transducer, N(m) | ftBiasFPGA.vi |

### faultsFPGA.vi

The function faultsFPGA.vi combines the encoder fault information from predictFPGA.vi with the force/torque controller health information derived from reading

Table B.50: coordFPGA Outputs

| Parameter | Description | Destination |
|---|---|---|
| ftX_N | Force in absolute X direction, N | humanFPGA.vi |
| ftY_N | Force in absolute Y direction, N | humanFPGA.vi |
| ftZ_N | Force in absolute Z direction, N | humanFPGA.vi |
| ftX_Nm | Torque about absolute X axis, Nm | humanFPGA.vi |
| ftY_Nm | Torque about absolute X axis, Nm | humanFPGA.vi |
| ftZ_Nm | Torque about absolute X axis, Nm | humanFPGA.vi |

DIO2. This signal monitors the open/closed status of the force/torque controller health bit by reading the voltage across a $5k\Omega$ resistor. The process to energize this circuit is described in the initFPGA.vi section.

Table B.51: faultsFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| faultsS_bool | Encoder 1 (shoulder motor) fault, bool | predictFGPA.vi |
| faultsE_bool | Encoder 2 (elbow motor) fault, bool | predictFGPA.vi |

Table B.52: faultsFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| faultsS_bool | Encoder 1 (shoulder motor) fault, bool | outFGPA.vi |
| faultsE_bool | Encoder 2 (elbow motor) fault, bool | outFGPA.vi |
| faultsF_bool | Force/torque controller fault, bool | outFGPA.vi |

**timeFPGA.vi**

The function timeFPGA.vi records the time elapsed once during each FPGA frame and converts the measured update rate to Hz to be used in integrals and differentials.

Table B.53: timeFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| rateFPGA_Hz | FPGA time step, seconds | outFGPA.vi |

### EMG.vi

The function EMG.vi reads the selected differential EMG signals in from module 2 of the compactRIO. No processing is performed on this data.

Table B.54: EMG.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| whichEMG(0-15) | Array of booleans to indicate which voltages to sample | run.vi |

Table B.55: EMG.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| EMG | Array of EMG voltages | logData.vi |

### outFPGA.vi

The function outFPGA.vi performs the output processing for both the shoulder and elbow motors. First the function runs safetyFPGA.vi to determine whether the position and velocity of the end-effector are within the safe range. The safety bit is combined with the sensor fault bits before being sent to inertiaFPGA.vi which performs optional inertia compensation. After this runs, a set of nested case structures is used to determine the correct output torques to be commanded by motorFPGA.vi. The first case structure depends on the encoder detected faults. If faults have been detected on either encoder, an output of 0 Nm is sent to the motors. If the encoders are fault-free, the second nested case structure uses the safety check run in safetyFPGA.vi. If the robot has an unsafe position or velocity, a damping control is implemented on the robot based on end-effector velocity and using the function force2torqueFPGA.vi to convert output forces to torques. If the robot is safe, the third nested case structure is executed. If the user has turned on inertia compensation, the fourth nested case structure checks to see if there is a force/torque transducer fault. If there is, a torque of 0 Nm is sent to the motors. If there is no fault, the torque

commanded by the inertia compensation is sent to the motors. If inertia compensation is turned off, the output specified by the high-level controller is commanded. If the output in in the form of end-effector forces, the function force2torqueFPGA.vi is used to convert the forces to torques. Otherwise the torques are sent directly to the motors.

Table B.56: outFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| inputs | Described in Table B.4 | inFPGA.vi |
| control | Described in Table B.2 | bundle.vi |

Table B.57: outFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| inertiaVals | Described in Table B.5 | logData.vi |
| outputs | Described in Table B.3 | logData.vi |

### safetyFPGA.vi

The function safetyFPGA.vi determines whether the position and velocity are within a safe range. The range of safe position is determined by the controllability of the robot. It is determined that end-effector kinematic Jacobian has a condition number less than 2 when the end-effector is inside a band of two concentric circles of radius 0.45 m and 0.8 m around the motor axis, shown in Figure 3-5. If the end-effector is outside this range or is moving at a velocity faster than 0.4 m/s, it is deemed unsafe and a damping controller is used to keep the robot outside of this zone.

Table B.58: safetyFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |

Table B.59: safetyFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| unsafe_bool | The end-effector is at an unsafe position or velocity | outFPGA.vi |

### inertiaFPGA.vi

The function inertiaFPGA.vi provides inertia compensation consisting of an admittance model to calculate desired acceleration based on input force and an adaptive tracking controller that outputs a torque that forces the robot to achieve the desired trajectory. The input force is processed by humanFPGA.vi and the admittance model which calculates the desired trajectory in Cartesian coordinates is contained in refXyFPGA.vi. The desired trajectory is converted to joint coordinates in refAngleFPGA.vi. The controller error is calculated in errorFPGA.vi, and the feedforward term is calculated in YaFPGA.vi. An adaptation law in a-lawFPGA.vi calculates the desired rate of change of the adaptable parameters, and the parameters are adapted in adaptFPGA.vi. An estimate of kinetic friction is contained in frictionFPGA.vi, and all the elements of the controller are put together in finalFPGA.vi.

Table B.60: inertiaFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| inputs | Described in Table B.4 | inFPGA.vi |
| control | Described in Table B.2 | bundle.vi |

Table B.61: inertiaFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| inertiaVals | Described in Table B.5 | logData.vi |

### humanFPGA.vi

The function humanFPGA.vi processes the measured forces for use in the admittance model and the trajectory controller as described in Section 4.1.1. When the robot

end-effector accelerates across the workspace, even when no external forces are applied to the outside of the handle, the force transducer measures the inertial forces from the handle. The goal of the inertia compensation is to react to the forces applied to the outside of the handle and not to the transducer, so the inertia of the robot handle above the transducer (about 0.44 kg) multiplied by the handle acceleration is subtracted off from the measured force as explained in Equation 4.2. This "human force" is converted to joint torques by the system Jacobian for use in the trajectory controller. It is also converted into a magnitude and direction (Equation 4.3) for use in the admittance model. The magnitude of force is applied a dead-band to reduce the effects of noise and mismatched phase between force and acceleration as in Equation 4.4.

Table B.62: humanFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |

Table B.63: humanFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| humanForceX_N | Force applied to the handle, X, N | logData.vi |
| humanForceY_N | Force applied to the handle, X, N | logData.vi |
| humanTrqS_Nm | Shoulder joint torques from handle force, Nm | finalFPGA.vi |
| humanTrqE_Nm | Elbow joint torques from handle force, Nm | finalFPGA.vi |
| beta_pirad | The direction of applied force, pi rad | refXyFPGA.vi |
| fh_N | The magnitude of applied force, N | refXyFPGA.vi |

### refXyFPGA.vi

The function refXyFPGA contains the admittance model in Cartesian coordinates. First the function vProjFPGA.vi projects the measured velocity into the direction of the desired constraint and the orthogonal direction. The large case structure contains separate admittance models for the constrained motion and unconstrained motion. If

the constraint is turned off, the magnitude of desired acceleration (ad) is set to the magnitude of force divided by the desired mass as described in Section 4.1.2 Equation 4.5. This acceleration is broken into x and y component using the direction of force $\beta$ to form the desired acceleration (Equation 4.6). If inertia compensation is turned on and is active (the robot has no faults and is not applying safety damping) forward Euler integration is applied on the acceleration reference to determine a desired velocity reference. If inertia compensation is not active, the desired velocity reference is set to the measured velocity, as described in Equation 4.7.

If the constraint is turned on, the admittance model becomes slightly more complicated as described in Section 4.1.3. The magnitude of desired acceleration in the direction (ad, gamma) of the constraint is determined by projecting the applied force into the direction of the constraint (Equation 4.8) and dividing by the desired mass (Equation 4.9). The magnitude of desired acceleration orthogonal to the direction of the constraint (ad, ortho) is determined in adOrthoFPGA.vi. These desired accelerations are broken down into their x and y components and then summed together as the combined desired acceleration as in Equation 4.18. If the compensation is active, the desired acceleration in the direction of the constraint (ad, gamma) is integrated based on the velocity reference in the direction of the constraint (vd, gamma) as in Equation 4.11. The desired acceleration orthogonal to the constraint (ad, ortho) is integrated based on the measured velocity orthogonal to the constraint as in Equation 4.16. This method is employed to keep error from winding up and preventing the controller gain from affecting the desired impedance parameters. If the compensation is not active, the desired velocities in the constraint and orthogonal direction are reset to the measured velocities in the constraint and orthogonal directions, respectively. The desired velocities in the constraint direction and the orthogonal direction are projected into their x and y components and summed together as the combined

desired velocity as in Equation 4.19.

Table B.64: refXyFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| inputs | Described in Table B.4 | inFPGA.vi |
| control | Described in Table B.2 | bundle.vi |
| beta_pirad | The direction of applied force, pi rad | humanFPGA.vi |
| fh_N | The magnitude of applied force, N | humanFPGA.vi |
| unsafe_bool | The end-effector is at an unsafe position or velocity | safetyFPGA.vi |

Table B.65: refXyFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| vdGamma_mps | Desired velocity along constraint, m/s | logData.vi |
| vdOrtho_mps | Desired velocity orthogonal, m/s | logData.vi |
| xyDesiredDDotX_mpss | Desired acceleration X, $m/s^2$ | refAngleFPGA.vi |
| xyDesiredDDotY_mpss | Desired acceleration Y, $m/s^2$ | refAngleFPGA.vi |
| xyDesiredDotX_mps | Desired velocity X, $m/s$ | refAngleFPGA.vi |
| xyDesiredDotY_mps | Desired velocity Y, $m/s$ | refAngleFPGA.vi |

### vProjFPGA.vi

The function vProjFPGA.vi projects the measured velocity into the direction of the desired constraint and the orthogonal direction. First the measured velocity is converted to higher resolution fixed-point number for use in the rest of refXyFPGA.vi. The constraint angle $\gamma$ is converted to pi radians using normFGPA.vi and that angle is used to project the measured velocity in the direction of the constraint and the orthogonal direction.

Table B.66: vProjFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| velX_mps | Measured velocity X, m/s | velFPGA.vi |
| velY_mps | Measured velocity Y, m/s | velFPGA.vi |
| gamma_rad | Constraint direction, rad | bundle.vi |

Table B.67: vProjFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| gamma_pirad | Constraint direction, pi rad | refXyFPGA.vi |
| cosG | cosine of gamma | refXyFPGA.vi |
| sinG | sine of gamma | refXyFPGA.vi |
| vGamma_mps | Velocity in constraint direction, $m/s$ | refXyFPGA.vi |
| vOrtho_mps | Orthogonal velocity, $m/s$ | refXyFPGA.vi |
| velX_mps | Measured velocity X, m/s | refXyFPGA.vi |
| velY_mps | Measured velocity Y, m/s | refXyFPGA.vi |

**adOrthoFPGA.vi**

The function adOrthoFGPA.vi uses the constraint impedance parameters to calculate an acceleration reference in the direction orthogonal to the constraint, when the constraint is turned on. The desired orthogonal velocity is multiplied by the desired damping and the measured orthogonal distance to the constraint is multiplied by the desired stiffness. The sum of these values is divided by the desired mass and summed with the measured orthogonal force divided by desired mass. This yields the desired orthogonal acceleration adOrtho as described in Equation 4.14.

Table B.68: adOrthoFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| vdOrtho_mps | Desired velocity in orthogonal direction, $m/s$ | refXyFPGA.vi |
| r_meters | Orthogonal distance to constraint, m | bundleFPGA.vi |
| sinGmB | sine of gamma-beta | refXyFPGA.vi |
| fhpm_mpss | Force divided by mass, kg | refXyFPGA.vi |

Table B.69: adOrthoFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| adOrtho_mpss | Desired orthogonal acceleration, $m/s^2$ | refXyFPGA.vi |

### refAngleFPGA.vi

The function refAngleFPGA.vi converts the desired acceleration and velocity into joint coordinates and integrates to calculate desired position. The functions invertFPGA.vi, negJdotFPGA.vi, and multFPGA.vi are used to calculate the desired joint accelerations from the desired Cartesian acceleration, as described in Equation 4.22. The desired joint velocity is calculated from the desired Cartesian velocity according to Equation 4.20. The desired joint positions are determined by integrating the desired velocity based on measured position rather than the previous reference, as described in Equation 4.23. Each joint position is normalized between 0 and $2\pi$.

Table B.70: refAngleFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |
| xyDesiredDDotX_mpss | Desired acceleration $\ddot{\theta}_{d,1}$, $m/s^2$ | refXyFPGA.vi |
| xyDesiredDDotY_mpss | Desired acceleration $\ddot{\theta}_{d,2}$, $m/s^2$ | refXyFPGA.vi |
| xyDesiredDotX_mps | Desired velocity $\dot{\theta}_{d,1}$, $m/s$ | refXyFPGA.vi |
| xyDesiredDotY_mps | Desired velocity $\dot{\theta}_{d,2}$, $m/s$ | refXyFPGA.vi |

Table B.71: refAngleFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| thetaDesiredDDotS_rpss | Desired joint acceleration $\ddot{\theta}_{d,1}$, $rad/s^2$ | refAngleFPGA.vi |
| thetaDesiredDDotE_rpss | Desired joint acceleration $\ddot{\theta}_{d,2}$, $rad/s^2$ | refAngleFPGA.vi |
| thetaDesiredDotS_rps | Desired joint velocity $\ddot{\theta}_{d,1}$, $rad/s$ | refAngleFPGA.vi |
| thetaDesiredDotE_rps | Desired joint velocity $\ddot{\theta}_{d,2}$, $rad/s$ | refAngleFPGA.vi |
| thetaDesiredS_rad | Desired joint position $\ddot{\theta}_{d,1}$, rad | refAngleFPGA.vi |
| thetaDesiredE_rad | Desired joint position $\ddot{\theta}_{d,2}$, rad | refAngleFPGA.vi |

### invertFPGA.vi

The function invertFPGA.vi inverts the 2x2 Jacobian matrix.

Table B.72: invertFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| jacobian | 2x2 Jacobian, m | meterFPGA.vi |

Table B.73: invertFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| Jinv | Inverse Jacobian, $1/m$ | refAngleFPGA.vi |

### negJdotFPGA.vi

The function negJdotFPGA.vi calculates and negates the derivative of the end-effector Jacobian described in Equation 2.15.

Table B.74: negJdotFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| jacobian | 2x2 Jacobian, m | meterFPGA.vi |
| velS_rps | Shoulder velocity, $rad/s$ | velFPGA.vi |
| velE_rps | Elbow velocity, $rad/s$ | velFPGA.vi |

Table B.75: negJdotFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| negJdot_mrps | negative inverse Jacobian | refAngleFPGA.vi |

### multFPGA.vi

The function multFPGA.vi multiplies a 2x2 matrix by a 2x1 vector.

Table B.76: multFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| matrix2x2 | 2x2 Matrix | refAngleFPGA.vi |
| vector2x1 | 2x1 Vector | refAngleFPGA.vi |

Table B.77: multFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| product | Product of matrix and vector | refAngleFPGA.vi |

### errorFPGA.vi

The function errorFPGA.vi calculates the sliding error term $\boldsymbol{s}$ described in Equation 4.25 as well as the reference parameters $\ddot{\boldsymbol{\theta}}_{\boldsymbol{r}}$ and $\dot{\boldsymbol{\theta}}_{\boldsymbol{r}}$, and robust error term $\boldsymbol{s}_{\boldsymbol{\Delta}}$. First the position and velocity error terms are calculated using desired and measured positions and velocities, respectively, as described in Equation 4.24. These are used to calculate the reference parameters using Equation 4.26. The sliding variable $\boldsymbol{s}$ is calculated using the reference parameter and measured position according to Equation 4.27. Finally the robust error term $\boldsymbol{s}_{\boldsymbol{\Delta}}$ is calculated by applying a deadband to $\boldsymbol{s}$ according to Equation 4.28.

Table B.78: errorFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |
| userInput | Described in Table B.1 | run.vi |
| thetaDesiredDDotS_rpss | Desired joint acceleration $\ddot{\theta}_{d,1}$, $rad/s^2$ | refAngleFPGA.vi |
| thetaDesiredDDotE_rpss | Desired joint acceleration $\ddot{\theta}_{d,2}$, $rad/s^2$ | refAngleFPGA.vi |
| thetaDesiredDotS_rps | Desired joint velocity $\dot{\theta}_{d,1}$, $rad/s$ | refAngleFPGA.vi |
| thetaDesiredDotE_rps | Desired joint velocity $\dot{\theta}_{d,2}$, $rad/s$ | refAngleFPGA.vi |
| thetaDesiredS_rad | Desired joint position $\theta_{d,1}$, rad | refAngleFPGA.vi |
| thetaDesiredE_rad | Desired joint position $\theta_{d,2}$, rad | refAngleFPGA.vi |

### YaFPGA.vi

The function YaFPGA.vi calculates the feedforward term of the adaptive tracking controller $\boldsymbol{Y}\hat{\boldsymbol{a}}$ where the configuration dependent term $Y$ is defined by Equation 4.32 and the adaptive terms are a vector of three terms described by Equation 4.31.

Table B.79: errorFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| thetaRDDotS_rpss | Reference joint acceleration $\ddot{\theta}_{r,1}$, $rad/s^2$ | YaFPGA.vi<br>a-lawFPGA.vi |
| thetaRDDotE_rpss | Reference joint acceleration $\ddot{\theta}_{r,2}$, $rad/s^2$ | YaFPGA.vi<br>a-lawFPGA.vi |
| thetaRDotS_rps | Reference joint velocity $\dot{\theta}_{r,1}$, $rad/s^2$ | YaFPGA.vi<br>a-lawFPGA.vi |
| thetaRDotE_rps | Reference joint velocity $\dot{\theta}_{r,2}$, $rad/s^2$ | YaFPGA.vi<br>a-lawFPGA.vi |
| sS_rps | Sliding error $s_1$, rad/s | finalFPGA.vi |
| sE_rps | Sliding error $s_2$, rad/s | finalFPGA.vi |
| sDeltaS_rps | Robust sliding error $s_{\Delta,1}$, rad/s | a-lawFPGA.vi |
| sDeltaE_rps | Robust sliding error $s_{\Delta,2}$, rad/s | a-lawFPGA.vi |

Table B.80: YaFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |
| thetaRDDotS_rpss | Reference joint acceleration $\ddot{\theta}_{r,1}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDDotE_rpss | Reference joint acceleration $\ddot{\theta}_{r,2}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDotS_rps | Reference joint velocity $\dot{\theta}_{r,1}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDotE_rps | Reference joint velocity $\dot{\theta}_{r,2}$, $rad/s^2$ | errorFPGA.vi |
| a1_kgmm | Adaptive parameter $a_1$, $kgm^2$ | adaptFPGA.vi |
| a2_kgmm | Adaptive parameter $a_2$, $kgm^2$ | adaptFPGA.vi |
| a3_kgmm | Adaptive parameter $a_3$, $kgm^2$ | adaptFPGA.vi |

Table B.81: YaFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| YaS_Nm | Shoulder Y*a, Nm | finalFPGA.vi |
| YaE_Nm | Elbow Y*a, Nm | finalFPGA.vi |
| Y31 | Y matrix element [1,3] | a-lawFPGA.vi |
| Y32 | Y matrix element [2,3] | a-lawFPGA.vi |

**a-lawFPGA.vi**

The function a-lawFPGA.vi calculates the value of $\dot{\hat{\boldsymbol{a}}}$ according to Equation 4.41.

Table B.82: a-lawFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| thetaRDDotS_rpss | Reference joint acceleration $\ddot{\theta}_{r,1}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDDotE_rpss | Reference joint acceleration $\ddot{\theta}_{r,2}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDotS_rps | Reference joint velocity $\dot{\theta}_{r,1}$, $rad/s^2$ | errorFPGA.vi |
| thetaRDotE_rps | Reference joint velocity $\dot{\theta}_{r,2}$, $rad/s^2$ | errorFPGA.vi |
| sDeltaS_rps | Robust sliding error $s_{\Delta,1}$, rad/s | errorFPGA.vi |
| sDeltaE_rps | Robust sliding error $s_{\Delta,2}$, rad/s | errorFPGA.vi |
| Y31 | Y matrix element [1,3] | YaFPGA.vi |
| Y32 | Y matrix element [2,3] | YaFPGA.vi |

Table B.83: a-lawFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| adaptation1 | Adaptive variable 1 adaptation rate | adaptFPGA.vi |
| adaptation2 | Adaptive variable 2 adaptation rate | adaptFPGA.vi |
| adaptation3 | Adaptive variable 3 adaptation rate | adaptFPGA.vi |

**adaptFPGA.vi**

The function adaptFPGA.vi integrates the adaptation law calculated in a-lawFPGA.vi to calculate the adaptive parameters when adaptation is turned on. The initial values for these parameters are defined in Equation 4.42. Integration is stopped if the adaptive values stray too far from their initial values. This can be done while maintaining stability, as described in Section 4.2.3.

Table B.84: adaptFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| adaptation1 | Adaptive variable 1 adaptation rate | a-lawFPGA.vi |
| adaptation2 | Adaptive variable 2 adaptation rate | a-lawFPGA.vi |
| adaptation3 | Adaptive variable 3 adaptation rate | a-lawFPGA.vi |

Table B.85: adaptFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| a1_kgmm | Adaptive parameter $a_1$, $kgm^2$ | YaFPGA.vi |
| a2_kgmm | Adaptive parameter $a_2$, $kgm^2$ | YaFPGA.vi |
| a3_kgmm | Adaptive parameter $a_3$, $kgm^2$ | YaFPGA.vi |

**frictionFPGA.vi**

The function frictionFPGA.vi contains a basic model of kinetic friction described by Equation 2.18 based on the joint velocities.

Table B.86: frictionFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| inputs | Described in Table B.4 | inFPGA.vi |

Table B.87: frictionFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| frictionS_Nm | Estimated shoulder joint friction, Nm | finalFPGA.vi |
| frictionE_Nm | Estimated elbow joint friction, Nm | finalFPGA.vi |

**finalPGA.vi**

The function finalFPGA.vi combines all the terms of the control law in Equation 4.34.

Table B.88: finalFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| userInput | Described in Table B.1 | run.vi |
| sS_rps | Sliding error $s_1$, rad/s | errorFPGA.vi |
| sE_rps | Sliding error $s_2$, rad/s | errorFPGA.vi |
| humanTrqS_Nm | Shoulder joint torques from handle force, Nm | humanFPGA.vi |
| humanTrqE_Nm | Elbow joint torques from handle force, Nm | humanFPGA.vi |
| YaS_Nm | Shoulder Y*a, Nm | YaFPGA.vi |
| YaE_Nm | Elbow Y*a, Nm | YaFPGA.vi |
| frictionS_Nm | Estimated shoulder joint friction, Nm | frictionFPGA.vi |
| frictionE_Nm | Estimated elbow joint friction, Nm | frictionFPGA.vi |

Table B.89: finalFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| inertiaTorqueS_Nm | Inertia compensation joint 1 torque, Nm | outFPGA.vi |
| inertiaTorqueE_Nm | Inertia compensation joint 2 torque, Nm | outFPGA.vi |

**force2torqueFPGA.vi**

The function force2torqueFPGA.vi uses the kinematic Jacobian to convert end-effector commanded forces to joint torques according to Equation 2.16.

Table B.90: force2torqueFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| forceX_N | Commanded X force, N | outFPGA.vi |
| forceY_N | Commanded Y force, N | outFPGA.vi |
| jacobian | End-effector Jacobian, m | meterFPGA.vi |

Table B.91: force2torqueFPGA.vi Outputs

| Parameter | Description | Destination |
|---|---|---|
| trqOutS_Nm | Transformed torque 1, Nm | motorFPGA.vi |
| trqOutE_Nm | Transformed torque 2, Nm | motorFPGA.vi |

**motorFPGA.vi**

The function motorFPGA.vi reads in the desired joint torques and compares them to the motor limitations, scaling the torques down if necessary using scaleFPGA.vi. The motor torques are then converted to voltages using the relationship described in Table 2.1 and the output voltages are set accordingly.

Table B.92: motorFPGA.vi Inputs

| Parameter | Description | Source |
|---|---|---|
| trqOutS_Nm | Commanded torque 1, Nm | outFPGA.vi |
| trqOutE_Nm | Commanded torque 2, Nm | outFPGA.vi |

### scaleFPGA.vi

The function scaleFPGA.vi examines the desired output torque of each motor. If either desired torque exceeds the limit of 28.8 Nm, both torques are scaled back proportionally such that the highest torque is equal to 28.8 Nm. This is a crude attempt to preserve direction of the output which will not actually preserve the direction (the Jacobian would be needed for that), but is better than nothing.

Table B.93: scaleFPGA.vi Inputs

| **Parameter** | **Description** | **Source** |
|---|---|---|
| torqueS_Nm | Desired output torque 1, Nm | outFPGA.vi |
| torqueE_Nm | Desired output torque 1, Nm | outFPGA.vi |

Table B.94: scaleFPGA.vi Outputs

| **Parameter** | **Description** | **Destination** |
|---|---|---|
| torqueS_Nm | Scaled down output torque 1, Nm | motorFPGA.vi |
| torqueE_Nm | Scaled down output torque 1, Nm | motorFPGA.vi |

# References


[1] N. Hogan and D. Sternad, "Dynamic primitives of motor behavior," *Biological Cybernetics*, vol. 106, pp. 727–739, Dec. 2012.

[2] J. Charnnarong, "The design of an intelligent machine for upper-limb physical therapy," Master's thesis, Massachusetts Institute of Technology, 1991.

[3] H. I. Krebs, N. Hogan, M. L. Aisen, and B. T. Volpe, "Robot-aided neurorehabilitation," *IEEE Transactions on Rehabilitation Engineering*, vol. 6, pp. 75–87, Mar. 1998.

[4] N. Hogan and H. I. Krebs, "Interactive robots for neuro-rehabilitation," *Restorative Neurology and Neuroscience*, vol. 22, pp. 349–358, Oct. 2004.

[5] D. Adebiyi, "Fabrication and characterization of beta-prototype mit manus: an intelligent machine for upper-limb physical therapy," Master's thesis, Massachusetts Institute of Technology, 1998.

[6] C. Foster, "A performance characterization of an interactive robot," Master's thesis, Massachusetts Institute of Technology, 1999.

[7] B. Laboratories, "Inmotion arm interactive therapy system." `http://bionikusa.com/inmotion-arm-the-new-standard-of-care/`, 2018. [Online; accessed 09-April-2018].

[8] J. A. Doeringer, *An Investigation into the Discrete Nature of Human Arm Movements*. PhD thesis, Massachusetts Institute of Technology, 1999.

[9] B. Wilcox, "Study of human motor control and task performance with circular constraints," Bachelor's thesis, Massachusetts Institute of Technology, 2016.

[10] R. Koeppen and M. E. Huber, "Controlling physical interactions: Humans do not minimize muscle effort," in *ASME 2017 Dynamic Systems and Control Conference*, 2017.

[11] M. Mahvash and A. M. Okamura, "Enhancing transparency of a position-exchange teleoperator," in *Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2007.

[12] J. J. Gil, Á. Rubio, and J. Savall, "Decreasing the apparent inertia of an impedance haptic device by using force feedforward," *IEEE Transactions on Control Systems Technology*, vol. 17, pp. 833–838, July 2009.

[13] G. Aguirre-Ollinger, J. E. Colgate, M. A. Peshkin, and A. Goswami, "Design of an active one-degree-of-freedom lower-limb exoskeleton with inertia compensation," *The International Journal of Robotics Research*, vol. 30, pp. 486–499, July 2011.

[14] N. Colonnese and A. Okamura, "M-width: Stability and accuracy of haptic rendering of virtual mass," in *Robotics: Science and Systems VIII* (N. Roy, P. Newman, and S. Srinivasa, eds.), pp. 41–48, Cambridge, Massachusetts: The MIT Press, 2013.

[15] A. Erwin, E. Pezent, J. Bradley, and M. K. O'Malley, "The effect of robot dynamics on smoothness during wrist pointing," in *International Conference on Rehabilitation Robotics*, 2017.

[16] N. Hogan, "Impedance control: An approach to manipulation: Part 1 - theory," *Journal of Dynamic Systems, Measurement, and Control*, vol. 107, pp. 1–7, Mar. 1985.

[17] E. Colgate and N. Hogan, "The interaction of robots with passive environments: Application to force feedback control," *Advanced Robotics*, pp. 465–474, 1989.

[18] E. Colgate and N. Hogan, "An analysis of contact instability in terms of passive physical equivalents," in *International Conference on Robotics and Automation*, 1989.

[19] W. S. Newman, "Stability and performance limits of interaction controllers," *Journal of Dynamic Systems, Measurement, and Control*, vol. 114, pp. 563–570, Dec. 1992.

[20] S. P. Buerger and N. Hogan, "Complementary stability and loop shaping for improved human-robot interaction," *IEEE Transactions on Robotics*, vol. 23, pp. 232–244, Apr. 2007.

[21] National Instruments, *NI cRIO-9034 Embedded CompactRIO Controller with Real-Time Processor and Reconfigurable FPGA User Manual*, 2015.

[22] Kollmorgen, a Danaher Corporation, *SERVOSTAR® CD Setup and Reference Guide*, 2002.

[23] National Instruments, *NI 9263 4-Channel, ±10V, 16-Bit Analog Voltage Output Module Operating Instructions and Specifications*, 2009.

[24] Gurley Precision Instruments, *Model VB Virtual Absolute™ Interpolating Decoder User's Manual*, 2000.

[25] National Instruments, *NI 9403E 32-Channel, TTL Digital Input/Output Module Operating Instructions and Specifications*, 2008.

[26] ATI Industrial Automation, *F/T Controller Six-Axis Force/Torque Sensor System Installation and Operation Manual*, 2016.

[27] National Instruments, *NI 9205 32-Channel,* $\pm 200mV$ *to* $\pm 10V$*, 16-Bit Analog Input Module Operating Instructions and Specifications*, 2008.

[28] N. Hogan, "Impedance control: An approach to manipulation: Part 2 - implementation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 107, pp. 8–16, Mar. 1985.

[29] L. N. Hand and J. D. Finch, *Analytical Mechanics*. Cambridge, UK: Addison-Wesley, 1998.

[30] S. Andersson, A. Söderberg, and S. Björklund, "Friction models for sliding dry, boundary and mixed lubricated contacts," *Tribology International*, vol. 40, pp. 580–587, Apr. 2007.

[31] Github, "Newman laboratory inmotion2 software repository." `https://github.mit.edu/newmanlab/InMotion2/`, 2018. [Online; accessed 09-April-2018].

[32] N. Hogan, "Impedance control: An approach to manipulation: Part 3 - applications," *Journal of Dynamic Systems, Measurement, and Control*, vol. 107, pp. 17–24, Mar. 1985.

[33] J. K. Salisbury and J. J. Craig, "Articulated hands: Force control and kinematic issues," *The International Journal of Robotics Research*, vol. 1, pp. 4–17, Apr. 1982.

[34] M. H. Raibert and J. J. Craig, "Hybrid position/force control of manipulators," *Journal of Dynamic Systems, Measurement and Control*, vol. 103, pp. 126–133, June 1981.

[35] T. Valency and M. Zacksenhouse, "Accuracy/robustness dilemma in impedance control," *Journal of Dynamic Systems, Measurement and Control*, vol. 125, pp. 310–319, Sept. 2003.

[36] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Upper Saddle River, New Jersey: Prentice Hall International Inc., 1991.

[37] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *International Journal of Robotics Research*, vol. 6, pp. 49–60, Oct. 1987.

[38] J.-J. E. Slotine and J. A. Coetsee, "Adaptive sliding controller synthesis for non-linear systems," *International Journal of Control*, vol. 43, pp. 1631–1651, Sept. 1985.

[39] Y. Tanaka, T. Tsuji, and H. Miyaguchi, "Analysis of human perception ability for robot impedance," in *16th IFAC World Congress*, 2005.

[40] G. Ekman, "Weber's law and related functions," *Journal of Psychology*, vol. 47, pp. 343–352, Apr. 1959.

[41] H. E. Ross and E. E. Brodie, "Weber fractions for weight and mass as a function of stimulus intensity," *The Quarterly Journal of Experimental Psychology*, vol. 39, pp. 77–88, Feb. 1987.

[42] R. Petrella, M. Tursini, L. Peretti, and M. Zigliotto, "Speed measurement algorithms for low-resolution incremental encoder equipped drives: a comparative analysis," in *2007 International Aegean Conference on Electrical Machines and Power Electronics Electrical Machines and Power Electronics*, 2007.