Design and Implementation of Scalable Real Time Motor Controller Architecture for Humanoid Robots and Exoskeletons

Shriya Shah

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Alexander Leonessa
Alan T. Asbeck
Pratap Tokekar
Daniel J. Stilwell

July 6, 2017
Blacksburg, Virginia

Keywords: Embedded Systems, RTOS, Series Elastic Actuator, Exoskeleton, Humanoid Robot
Copyright 2017, Shriya Shah

Design and Implementation of Scalable Real Time Motor Controller Architecture for Humanoid Robots and Exoskeletons

Shriya Shah

(ABSTRACT)

Embedded systems for humanoid robots are required to be reliable, low in cost, scalable and robust. Most of the applications related to humanoid robots require efficient force control of Series Elastic Actuators (SEA). These control loops often introduce precise timing requirements due to the safety critical nature of the underlying hardware. Also the motor controller needs to run fast and interface with several sensors. The commercially available motor controllers generally do not satisfy all the requirements of speed, reliability, ease of use and small size. This work presents a custom motor controller, which can be used for real time force control of SEA on humanoid robots and exoskeletons. Emphasis has been laid on designing a system which is scalable, easy to use and robust. The hardware and software architecture for control has been presented along with the results obtained on a novel Series Elastic Actuator based humanoid robot THOR.

This work is supported by NSF through grant number 1525972.

Design and Implementation of Scalable Real Time Motor Controller Architecture for Humanoid Robots and Exoskeletons

Shriya Shah

(GENERAL AUDIENCE ABSTRACT)

Humanoid robots can be used in several applications such as disaster management, replacing manual work in hazardous environments, helping human beings in navigation and day to day activities, etc. This increase in interests in humanoid robotics and related research in exoskeletons has led to the need of reliable embedded systems which is used to control the machines. These embedded systems are often required to be low in cost, scalable and robust. The specification required from the electronics and the embedded systems vary based on the robot's capabilities. Also, there is a gap between the requirements of humanoid robots in research and in industrial setting. This work focuses on bridging the gap by proposing a solution which is semi-custom, low in cost, reliable and scalable. The work has been shown to perform as expected on the stat-of-art humanoid robot THOR which was built at Virginia Tech. Using the proposed design technique can not only deliver good performance but can also act as a quick prototyping tool for other robotics projects related to humanoid robotics and exoskeletons.

To my mother for her *love*, *support* and *confidence*.

Acknowledgments

First and foremost, I would like to thank my advisor Dr. Alexander Leonessa for giving me an opportunity to work with him in the Terrestrial Robotics and Control group at Virginia Tech. His enthusiasm and immense knowledge helped me discover my interests and kept me focused on achieving my goals. I could not have imagined having a better advisor and mentor in my graduate studies.

My sincere thanks goes to Dr. Alan Asbeck for accepting me as a member of the prestigious exoskeleton project. This work has helped me grow my skills immensely as an Electrical Engineer. The financial assistance provided by him helped me concentrate completely on my work.

I would also like to thank the rest of my thesis committee. Dr. Pratap Tokekar has been a great source of inspiration. His course on Robotics helped me deepen my understanding of the subject and also gave me an opportunity to work on a project that I am still proud of. I thank Dr. Daniel Stilwell for agreeing to serve on my committee and for his valuable time.

I wish to express sincere gratitude to Dr. Steve Southward for his lectures on Control System, which helped me not only in my research but also in landing a job in the industry. I also thank Dr. Peter Neuhaus for accepting me as a summer intern at the Institute of Human and Machine Cognition (IHMC). My work experience at IHMC greatly helped me in my research as well as job search.

The members of the exoskeleton team have contributed to my personal and professional growth at Virginia Tech. Special thanks go to my senior and mentor, Robert Griffin. It was solely because of you that I understood a few things about the high level control architecture for humanoids, which I still believe is extremely complicated.

I am also grateful to Sarbani Mukherjee for introducing me to the world of Embedded Systems and Bijo Sebastian for our endless conversation on robotics which further motivated me to come to graduate school.

I sincerely thank my friends here at Virginia Tech: Amith Kaushal, Amruta Kulkarni, Aravind Preshant, Arpit Goyal and Koel Sinha. My last two years in Virginia Tech were eventful and absolutely memorable because of you. I thank my roommate Jonilda Bahja for always being nice and making the house feel like home. I am also grateful to my best friend

Prerna Nidhi for always being there for me.

Finally, I would like to express my heartfelt thanks to my family; my brother Shreyash Shah for always encouraging me, my uncle Bibhas Sarkar for his immense belief in my abilities, my aunt Smita Lath for her love, care and support.

Lastly, I would like to extend my special thanks to my best friend and boyfriend Tanumaya Tiwari for his unconditional love and confidence in me. You have always been my inspiration and I owe it all to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The research in humanoid robotics gained significant interest over the last few years. There are several applications where a humanoid robot can assist or replace a human worker such as mines, nuclear plants and other potentially hazardous environments. An advantage of these robots is the human like form factor which enables them to move easily in our human centric world. However, this also leads to major design challenges. The robot needs to interact with humans and often times the environment is unstructured. The use of traditional position control is not appropriate anymore because of its stiffness. Therefore force controllable actuators were proposed which have been used in the design of several legged robots such as Valkyrie [1], Atlas, TORO [2], Hume, COMAN [3], THOR, ESCHER [4], M2V2 etc.

Another recent trend in humanoid robotics research is building exoskeletons. For exoskeletons, the use of force controlled actuators is of paramount importance because of its safety critical nature. The lower body exoskeletons have been shown to be a successful alternative to wheelchairs for spinal cord injury (SCI) patients [5]. Other than this, several force controlled exoskeletons are being used for rehabilitation purpose [6], [7], sport training etc.

Pratt [8] proposed the Series Elastic Actuator (SEA) which can be used for force control of joints on legged robots. The SEA is characterized by a spring element in series with the load. These actuators can be linear or rotary and is generally powered using a DC motor. The output force is measured by measuring the deflection of the spring element or recording the output force using a load cell.

Several strategies have been proposed for force control of SEA. These strategies generally depend on the design of the SEA, linear or rotary, the capabilities of the embedded systems, performance requirements etc. These strategies have been well tested and implemented on several humanoid robots.

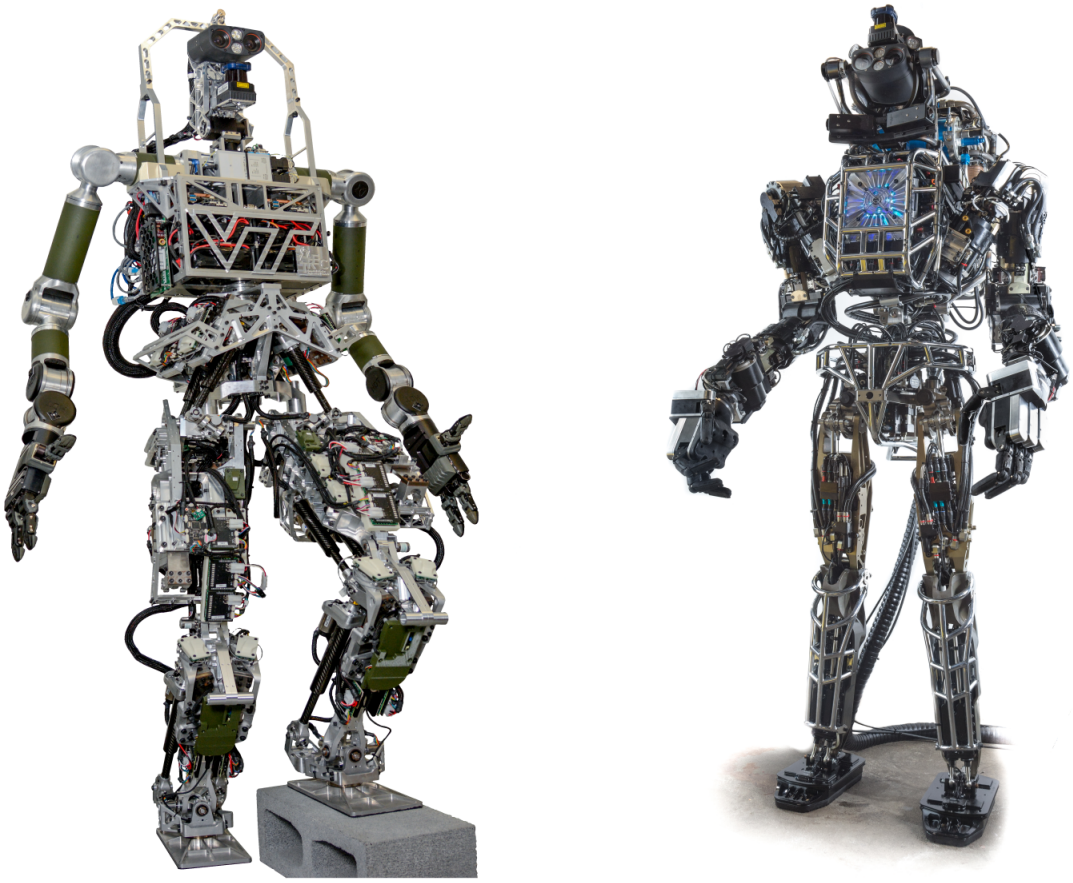


Figure 1.1: Figure shows ESCHER humanoid robot of Virginia Tech in the left and Atlas robot from Boston dynamics in right

Examples of a compliant humanoid robots are THOR and ESCHER, both built at the Terrestrial Robotics Engineering and Controls Laboratory at Virginia Tech. These robots use Series Elastic Actuators [9] which are controlled using custom motor slugs [10], which are powered with an ARM Cortex M4 processor. These controllers were proposed as a solution to the design requirements specific to THOR and ESCHER. However, it was observed that these boards pose severe restriction to expandability and robustness. The full specification for control of a SEA often requires interfacing with several sensors, the capability to run real time control algorithms and robustness in hardware design. Because of these strict requirements, it is often difficult to find an off-the-shelf solution, necessitating the need for a custom board.

1.2 Contribution

The aim of this work is to present a comprehensive solution for control of Series Elastic Actuators. The control of a humanoid robot depends on the use of several sensors. Each joint on such a robot typically has load cells, incremental encoders, absolute encoders, current sensor and motor control. Therefore, a custom motor controller hardware which is easy to use, expandable, low in cost and extremely robust, is designed and implemented. The controller uses several off-the-shelf products which makes prototyping fast and efficient.

The control of SEA typically requires very high loop frequency and often has strict timing requirements. To cater to these needs, a real time operating system based software stack is proposed. The developed software can achieve position and force control. The proposed real time operating system based software stack makes the system dependable and safe.

The force control loop is the building block of the controller architecture in a humanoid robot. Therefore its stability is extremely important. A PID based force control law is proposed, implemented and validated. To improve the control performance and to demonstrate the flexibility and ease of use of the motor controller architecture, a first order adaptive controller is also implemented.

Finally, the complete embedded system architecture for a fully autonomous lower body exoskeleton is proposed. This architecture can also be used for replacing the existing electronics on THOR and ESCHER and upgrading them with a real time stack.

1.3 Thesis Organization

The structure of the rest of the document is as follows. Chapter 2 provides a literature review to understand the state of art related to the field in discussion. Chapter 3 is an overview of the system, including the hardware and software. Chapter 4 is a detailed discussion on the hardware development which lists all the sensors and peripherals used in this work. Chapter 5 continues the detailed discussion on the software development with focus on the Real Time Operating System and the Board Support Packages. This chapter also presents the related results. Chapter 6 discusses two control applications where the entire hardware and software stack is tested. Finally, chapter 7 concludes the work and highlights the future work.

Chapter 2

Background

Humanoid robots are still in research and development stage. While there is a lot of interest in humanoid robots to replace the human counterpart in hazardous environment and domestic applications, most of the research concentrates on making a system which is safe and dependable. The same criteria apply to the ongoing research in exoskeletons. This chapter discusses the current state of art in humanoid robotics and exoskeletons with focus on series elastic actuators and their controllers.

Humanoid robotics started with the use of rotary actuators in each joint, such as that in Honda's ASIMO [11]. These robots use the well tested position controlled servos to achieve desired joint space configurations [12], [13]. However, this requires precise knowledge of the kinematics and the environment where the robot operates. Also, position controlled robots are unsafe to operate in unstructured and dynamic environment. The presence of obstacles can cause the position error to grow, which can lead to injuries, even fatal [14].

As a solution to this, force controlled actuators were proposed. By controlling the forces instead of positions, it can be made sure that the actuators do not apply unintended forces on the external environment, making the system compliant [15]. Pratt et al proposed series elastic actuators as a complaint mechanism for use in humanoid robots [8]. These actuators have a spring element in series with the load. The deflection of the spring can be used to estimate the force being applied at the output based on Hooke's law.

Several robots such as Boston Dynamics's Atlas [16], NASA's Valkyrie [1], Virginia Tech's ESCHER [4] etc. use compliant mechanisms. Compliant control of hydraulic humanoid joint is discussed in [17].

In terms of control of the actuators on the robot, there are two equally acceptable approaches. The first one is the centralized approach where a very fast embedded computer is responsible for computing the desired joint current set points, which is then communicated to each and every joint through some communication medium. The other approach is the decentralized one, where the embedded computer runs the high level planning and behavior based controller

and each joint has a low level embedded computer or microcontroller. The local computers take the joint set points in terms of position or force from the central computer and then runs a faster low level control loop which in turn generates the desired current to be supplied to the motor [18].

Valkyrie [1] uses a decentralized control approach where a central computer runs the high level planning and control algorithm and each joint of the robot has a local low level controller which executes force control actions.

A similar approach was followed in the design of ESCHER [4]. ESCHER has 2 main computers, one to control the lower body and walking behaviors and the other for controlling arms and collecting perception data from cameras and lidar. The SEA on each joint is controlled by a custom motor slug [19]. CAN communication is used in the entire robot. ESCHER was developed as an improvement to SAFFIR [20] and THOR.

IHMC on the other hand, uses a centralized control approach [21]. The embedded computer computes the desired joint set point which in this case is the desired joint position. The servo driver, ELMO Twitter Gold, takes the desired position as input, calculates the desired current and powers the motors [22]. Other exoskeletons designed at IHMC such as Mina [23] and NASA X1 [24] follow a similar architecture.

The selection of the high level embedded computer is not that challenging and mainly depends on requirements such as operating system, processor speed, parallel processing capabilities etc. However, the choice of the low level controller is often non-trivial. Embedded computers are generally used in research settings where cost and time of development are the primary criteria. On the other hand, microcontroller based solutions are used in commercial products where customization is required. Rouse et. al [25] used Raspberry Pi, an embedded computers for control of a prosthetic knee. For a similar prosthetic knee application, authors in [26] used a microcontroller based architecture.

Some researchers have tried to use off-the-shelf components to control prosthetics and exoskeletons such as [27] and [28]. A few researchers have also used commercial motor driver for centralized control approaches. These implementations often face the problem of lack of portability and higher cost. In [29], authors proposed a solution to this problem by developing a flexible electronic architecture for robotic applications. This highly capable board is easy to use and is scalable as well. However, the design is based on the development of custom boards which is expensive and difficult to modify. Thus, there is a need of an electronic architecture which is low in cost, scalable, easy to learn and use and robust.

Chapter 3

System Overview and Design Specification

3.1 Exoskeleton and Humanoid Robot Architecture

The motor controller and related electronics presented in this work is designed keeping the requirements of a compliant lower body exoskeleton in mind. These boards also act as an improvement to the existing motor slug on THOR and ESCHER humanoid robots which were developed at Virginia Tech [4]. Figure 3.1 shows the electronic architecture for the exoskeleton. The system is battery powered with 48V. The power distribution board converts the battery supply voltage to 24V, 12V and 15V rails for distribution across the entire system. There is a logic board which interfaces with the emergency e-stop button, display LCD and Motor relays.

The main computer is the Gigabyte Model GB-BXi7-4770R which runs Ubuntu 14.04 with a real time patch from IHMC [30]. Table 3.1 gives the specification of the embedded computer. A PCIE CAN card as shown in figure 3.2 is used for establishing CAN networking with the motor controllers. Table 3.2 lists the specifications for the card. An IMU sits over the pelvis and communicates over USB. The IMU data is used to estimate the orientation of the pelvis, which is then used by the high level control of walking behavior.

The main computer uses software from IHMC which is based on JAVA. For a humanoid robot or an exoskeleton, controllers are designed for achieving behaviors such as walking, stair climbing, ramp climbing, etc. One method of walking is based on Zero Moment Pivot (ZMP) where the controller tries to track the reference center of pressure trajectory and in order to do so, it generates a center of mass (COM) trajectory that the robot should follow [31]. This high level COM trajectory information is then used to find joint angles required by the robot to reach the specified COM. The desired joint position is then used by the controllers on the motors to perform local position control.

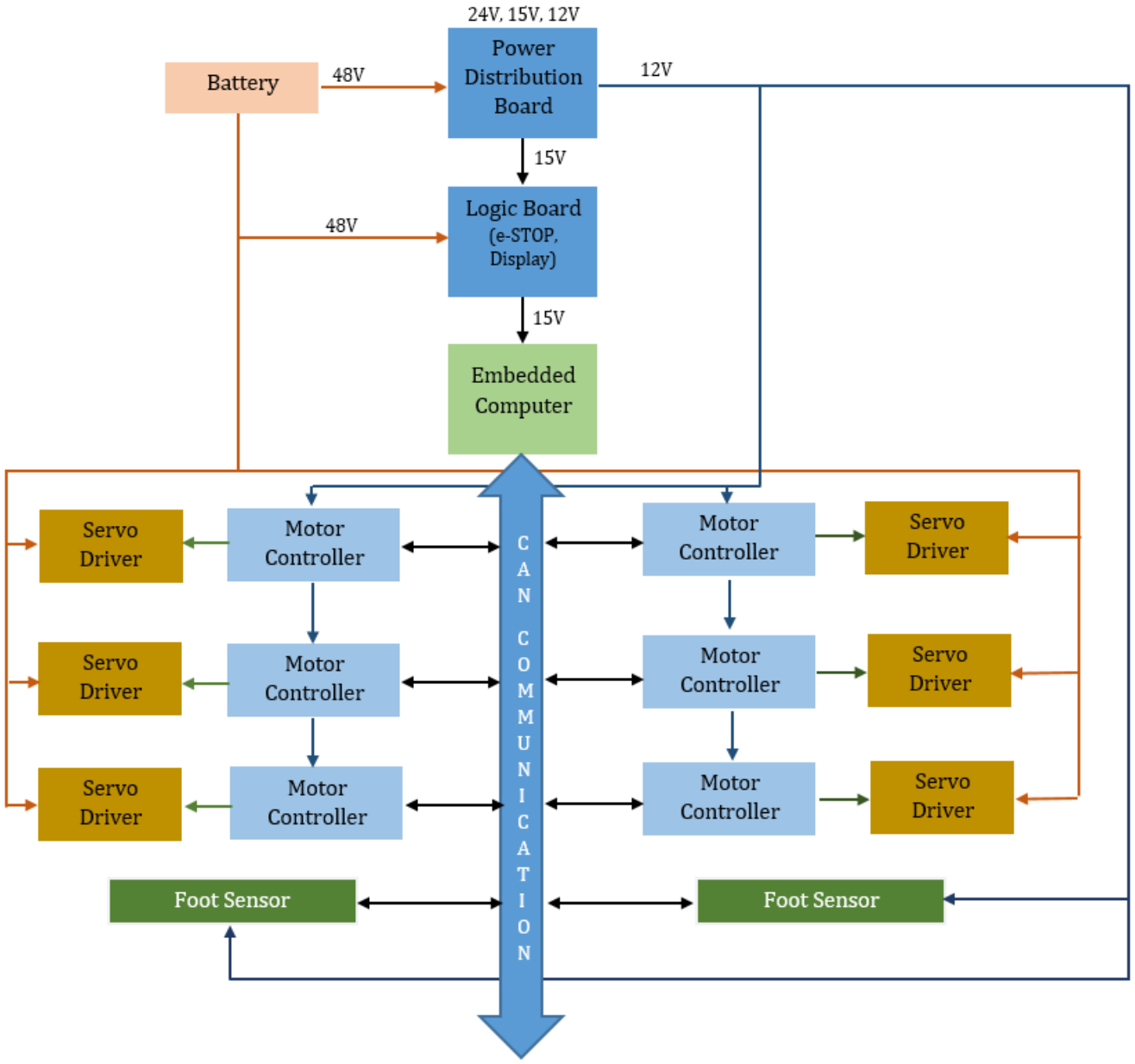


Figure 3.1: Electronics architecture for the lower body exoskeleton

Features	***Description***
CPU	4^{th} generation Intel Core i7-4770R, 3.90GHZ
Memory	2xSO-DIMM DDR3L slots (DDR3 1.35V), 1333 / 1600 MHz, Max 16 GB
Power Supply	Input: AC 100-240V, Output: DC 19V, 7.1 A
Chip Set	Intel HM87
Dimension	62x111.4x111.4 mm
LAN	Gigabit LAN (Realtek RTL8111G)

Table 3.1: Gigabyte Brick Specification

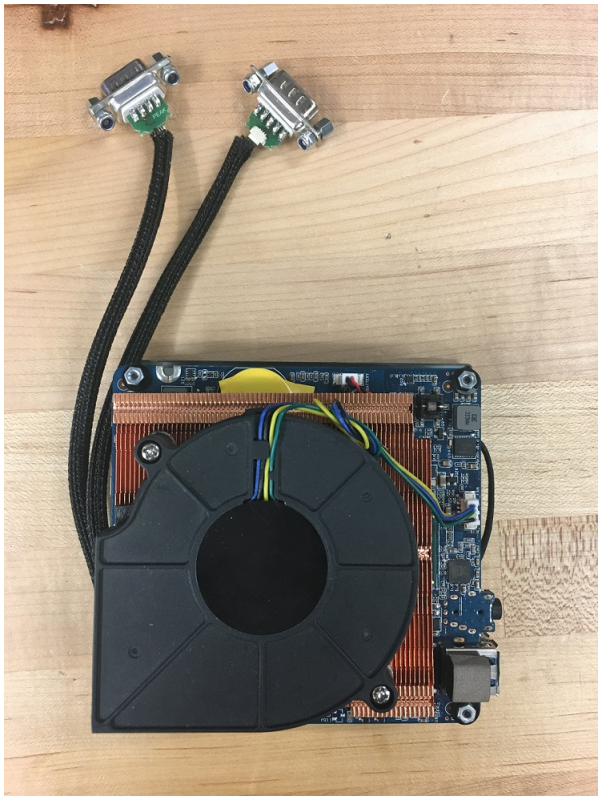


Figure 3.2: Embedded computer with CAN card

Features	***Description***
Interface	PCI express mini slot
Dimension	30x51x4 mm
Baud Arte	max 1 Mbit/sec
Specification	CAN 2.0A and 2.0B

Table 3.2: PCIe CAN card Specification

The position control approach has several limitations. It imparts stiffness to the robot which makes it unsuitable for operation in a dynamic environment and in the presence of humans. Also, the approach fails when the robot kinematics is not accurate. In order to make the joint complaint, the series elastic actuators are used. THOR and ESCHER shown in figure 3.3 use a custom SEA for every joint. These series elastic actuators allow for precise force measurement at the output through the use of a load cell.

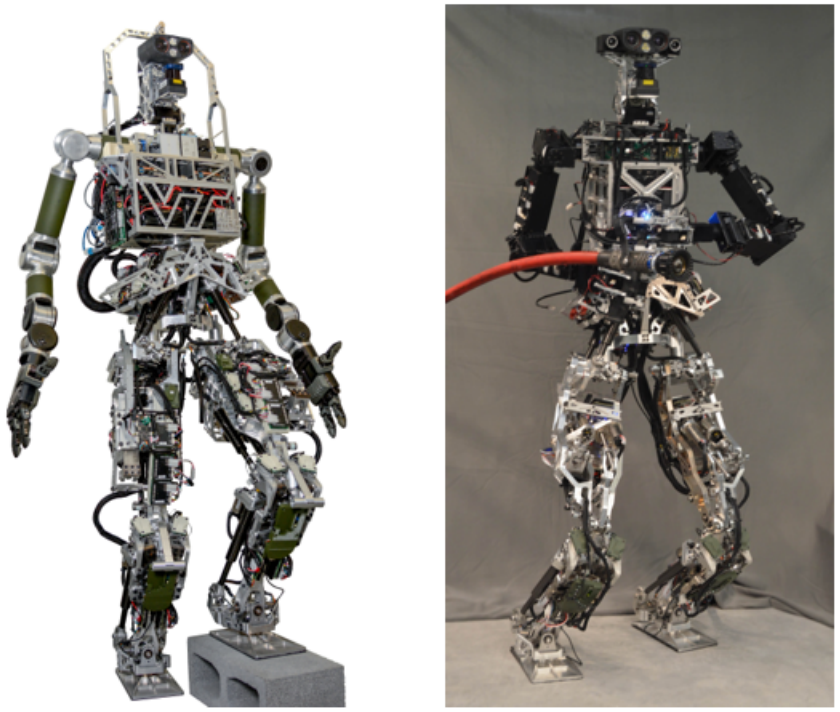


Figure 3.3: THOR and ESCHER built at Virginia Tech

Now, in order to control force instead of position, the high level control strategy is modified to generate torque, velocity and acceleration set points for each and every joint.

3.2 Control Architecture

As discussed in the last section, the high level controller generates acceleration and velocity set points for every joint of the robot based on the desired behavior, which for example can be walking on flat ground. Figure 3.4 shows the control architecture which can be used on a humanoid robot or an exoskeleton.

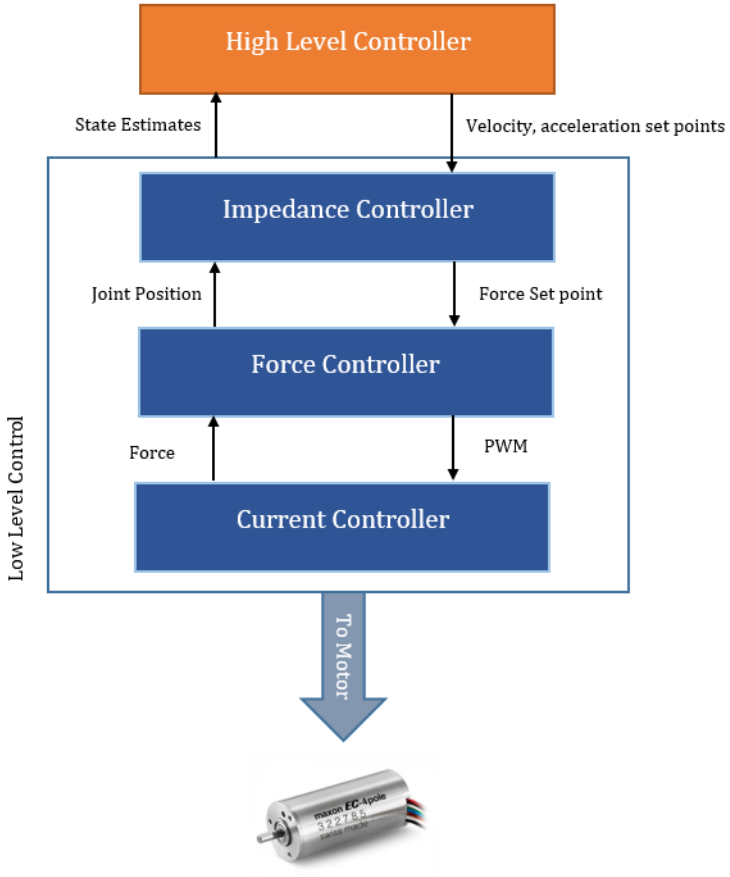


Figure 3.4: Control Architecture

The joint set points communicate over CAN network. The impedance controller uses the velocity and acceleration set points and calculates force or torque set points using inverse static and kinematic on respective joints. The impedance controller uses feedback from absolute encoders on the joints which gives the position, velocity and acceleration of the joint.

The force set point is then used by the force control loop to achieve compliant control. The force controller uses feedback from the load cell in-line with the actuator. A PWM signal

which is proportional to the desired current on the motor is generated. Finally, a current control loop running on the servo driver controls the current and runs the motor.

3.3 Motor Controller Architecture

The low level control of the series elastic actuators comprises of the impedance control and the force control loops. This requires interfacing with absolute encoders, incremental encoders, load cells, current sensors, motor controller, etc. In addition to this, several applications call for the use of several other sensors and peripherals. As such, the embedded system on every joint needs to fulfill the following criteria:

- Scalable, allow interfacing with several inputs and outputs
- Supports CAN communication
- Robust and minimum wiring required
- Software is easy to program, modular
- Supports Real Time Operating System development in a modular fashion
- Allows quick prototyping from both hardware as well as software side
- Low cost

A general off-the-shelf embedded system solution is found insufficient to solve all the above mentioned criteria. However, the design of a custom board increases the complexity and the price of the system. Therefore a solution which adds features to an existing development board is proposed. Figure 3.5 shows the high level block diagram of the motor controller architecture.

The proposed design is stackable and therefore scalable too. There is a servo driver that can supply the commanded current to the motor. An interfacing board is used to interface the servo driver with the motor controller, which in this work is the Texas Instrument's TIVA Launch Pad (details in hardware development section). The motor controller is followed by a TIVA Shield, which houses all the electronics needed for sensor interfacing. The architecture can further be expanded to include custom or off-the-shelf booster packs such as LCD, SD card etc.

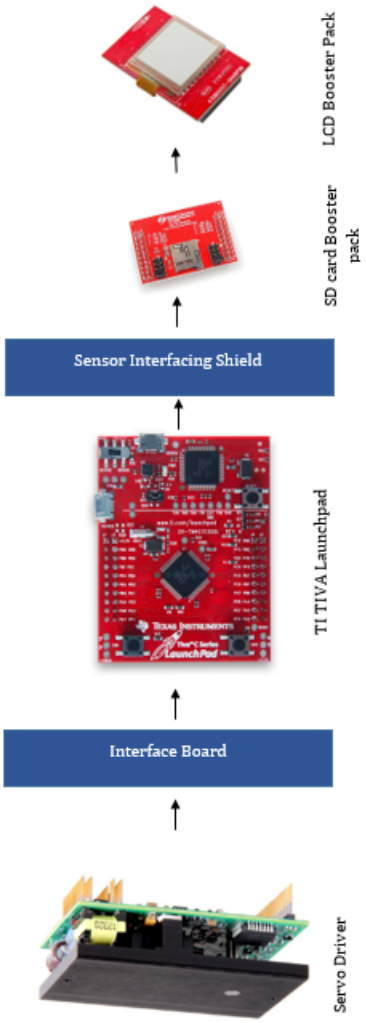


Figure 3.5: Electronics Architecture

The software for the motor controller is designed to be modular as well. The proposed Board Support Package eases the use of the peripherals on the Tiva shield. Moreover, custom function prototypes are provided for analog filtering, PID control, adaptive control, etc., which can help in quick prototyping of other prosthetic or exoskeleton projects. The software also allows easy integration of RTOS into the application.

Chapter 4

Hardware Development

4.1 Microcontroller or SBC

The electronic architecture for humanoid robots and exoskeletons are based on microcontrollers as well as embedded computers. The choice depends on several factors such as development time, size, weight, resource availability etc. The embedded computer based solutions such as Beaglebone Black and Raspberry Pi are being used in research labs such as [25]. IHMC uses off the shelf motor controller boards, ELMO Twitter for the low level control of the Elastic Actuators [32]. The microcontroller based solutions are used predominantly for commercial applications as the development time and effort are generally higher. But at the same time, microcontroller based architecture provides advantages such as flexibility in choice of peripherals and connectors, board optimized bare metal code, smaller size and weight, etc. Terrestrial Robotics Engineering and Control group at Virginia Tech developed an in-house board based on STM32F controllers [19] which is a 32 bit ARM Cortex architecture that support floating point computations.

Table 4.1 shows a comparison between microcontroller based architectures and single board computers. In this work, microcontroller based solution is employed. A comparative study is performed between Beaglebone Black (rev c) [33] and TI's Tiva Launch Pad boards. The Tiva boards as shown in figure 4.1 are chosen as the development platform because of its smaller size, availability of sufficient number of I/O pins and peripherals, fast processor and robustness. This solution also allowed for reduction in development time as compared to developing a custom board as described in [19]. These $15 boards have specifications as given in table 4.2. The Tiva TM4C Launch Pad houses an ARM Cortex M4 based microcontroller. It has all the major peripherals built in along with USB 2.0 device interface and on board CAN channels [34]. The stackable headers interface makes it easy to expand the functionality of the board by using existing or by designing custom booster packs [35].

Microcontroller	*Single Board Computer*
Board size depends on developer	Board size often fixed
Takes time to develop	Faster development time with great support
Can be costly because of the effort and the tool chain but has lower production cost	Generally very cheap and mostly runs Linux
Efficient bare metal coding	Inefficient high level coding
Easy to modify to suit a particular application	Modification not possible or very difficult

Table 4.1: Comparison between Microcontroller and Single Board Computers

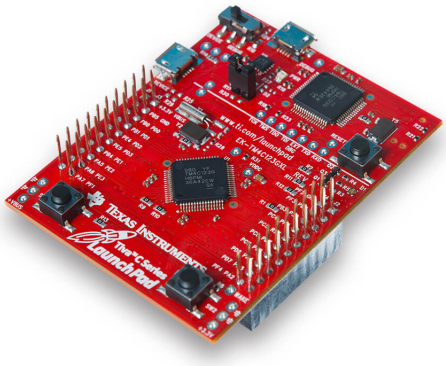


Figure 4.1: TI TIVA Launchpad

4.2 Sensor Interfacing

The motor controller presented here is capable of controlling one Series Elastic Actuator. Each actuator interfaces with several sensors which helps in the Adaptive force control, Impedance control and high Level control of stable walking behaviors. Figure 4.2 shows the hardware block diagram of all the sensors, actuators and communication peripheral on every actuator. Each sensor uses a specific communication channel which is clearly marked in the diagram. Detailed discussion of every sensor is presented in the following pages.

Moreover, because of a general purpose design, the board can be used to control several types of actuators being used in robotic applications. One of the goal of the work is to design an electronic architecture which can be used in each and every on-going project in our group

Features	***Description***
Performance	80 MHz; 100 DMIPS
Flash	256KB single cycle flash memory
EEPROM	2KB
Universal Asynchronous Receiver/-Transceiver (UART)	Eight UARTs
Synchronous Serial Interface (SSi)	Four SSi Module
Inter-Integrated Circuit (I^2C)	Four modules
Controller Area Network (CAN)	Two CAN 2.0 controllers
Pulse Width Modulation (PWM)	2 PWM module supporting a total of 16 PWM Outputs
Quadrature Encoder Interface (QEI)	Two QEI
Analog to Digital Converter (ADC)	Two 12 bit ADC with a maximum sampling rate of one million samples per second
JTAG and Serial Wire Debug (SWD)	One JTAG on board
General Purpose Timer (GPTM)	Six 16/32 bit GPTM

Table 4.2: TM4C launchpad Specifications

at Virginia Tech. To achieve this flexibility, several additional connectors and peripheral support are added which is discussed next.

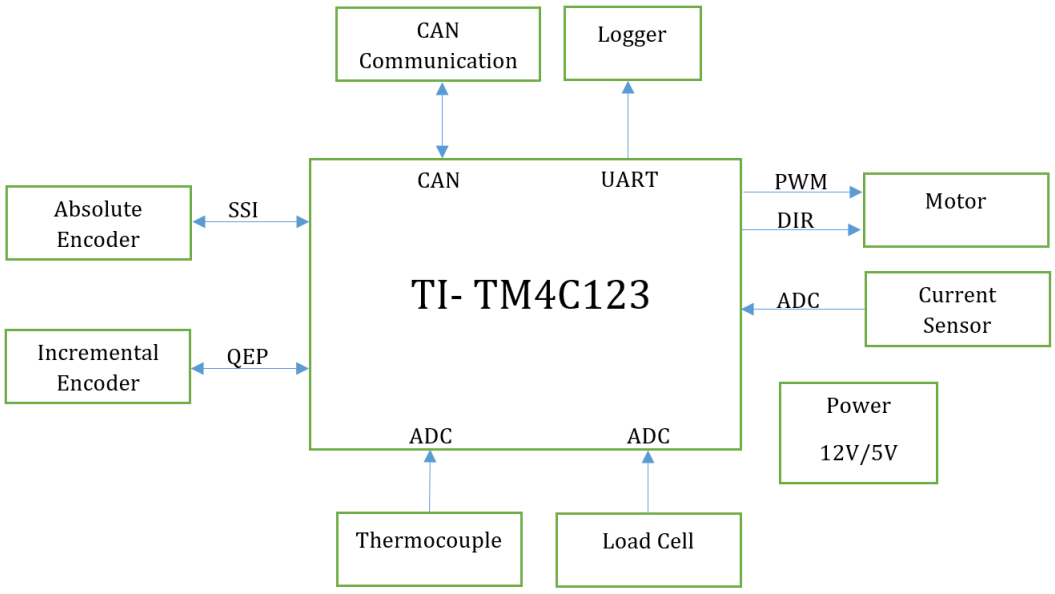


Figure 4.2: Hardware Block Diagram

4.2.1 Load Cells

Load Cell is a very common bridge sensor which can be used to measure both compression and tension forces. The operation is based on change of resistance of the strain gauge. But these resistance changes are very small and a sensitive circuitry is required to measure the changes. Typically a modified Wheatstone bridge circuit (single point bridge sensor) as shown in figure 4.3 is used where three resistor values are known and therefore even slight changes in the fourth resistor can be measured easily. The output voltage is given by equation 6.1. Here, R is the resistance of the fixed branch and R_G is the resistance of the strain gauge.

$$V_o = V_s\Big[\frac{R}{R+R_G} - \frac{1}{2}\Big] \tag{4.1}$$

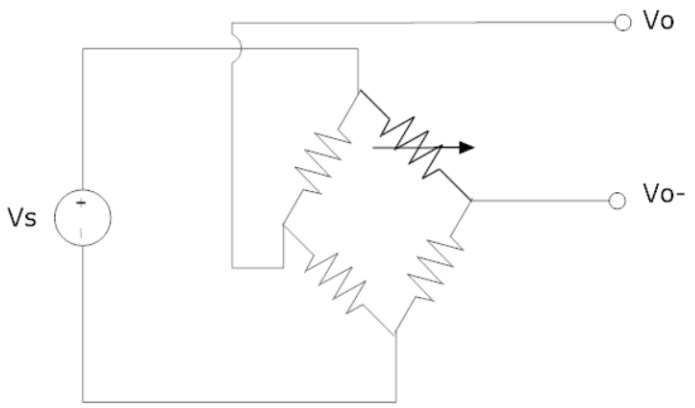


Figure 4.3: Single Point Bridge Sensor

There are a few terminologies associated with load cells as shown below.

1. Sensitivity: It is measured is mV/V and corresponds to the voltage output of the bridge when it is excited with 1 volt and the sensor operates at full scale.
2. Bridge Resistance: Resistance of bridge without any load.
3. Accuracy: The percentage accuracy of the output.

Table 4.3 gives the specifications for the Futek LCM200 load cells being used in our setup [36]. The bridge resistance determines the power consumption which for 10V excitation is

$28mA$. A sensitivity of $2mV/V$ means that the output of the load cell is $2mV$ for full scale of $500lb$ at excitation of 1V. For $10V$ excitation, the maximum output for full load is $20mV$. Since the output voltage is so small, an amplifier with high gain is required at the output. Figure 4.4 shows the basic circuit required for reading a load cell output voltage. A difference amplifier is used for converting the voltage from $0 - 20mV$ to $0 - 5V$. The output of the amplifier stage is then filtered to remove noise.

Description	***Values***
Sensitivity	$2mV/V$
Bridge Resistance	350 ohm
Nonlinearity	$\pm 0.5\%$ of RO
Max excitation	15V
Max Load	500 lb

Table 4.3: Futek LCM200 Load Cell Specifications

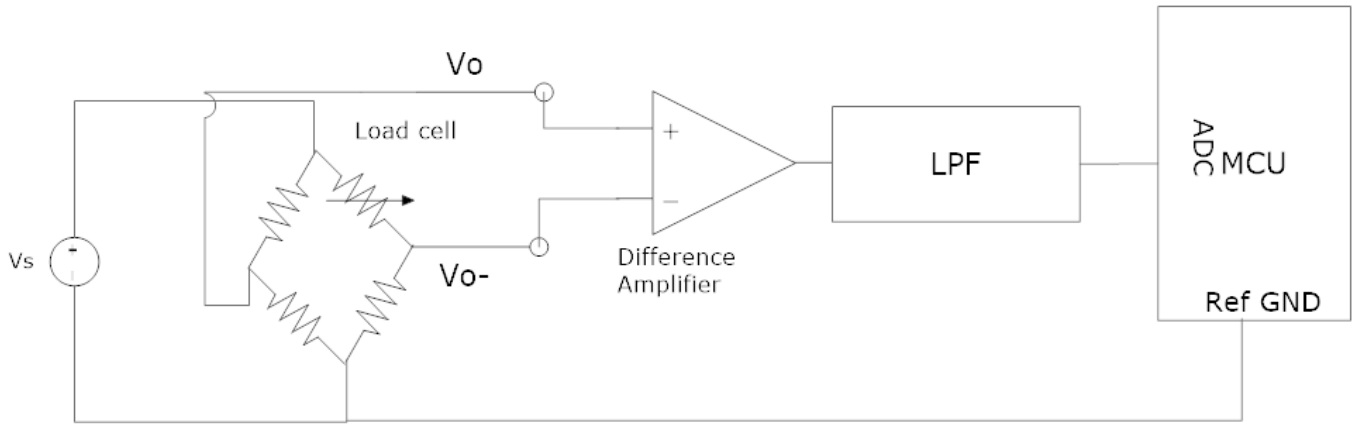


Figure 4.4: Basic circuit for reading load cell

For this work, INA125 instrumentation amplifier is used which has internal precision voltage reference which is very convenient for biasing the output [37]. Some other important specifications from the data sheet is pointed out in table 4.4. The LCM200 load cell has a natural frequency of $26.8kHz$. Therefore, to match the bandwidth of operation, a suitable gain is chosen using the trimming potentiometer. The suitable gain value can be obtained from the gain vs frequency chart shown in figure 4.5. Figure 4.6 given the resistance values for some typical gain values.

Description	***Values***
Common-mode rejection	$100dB$ at $G = 100$
Single as well as dual supply	$+2.7V$ to $+36V$ or $\pm 1.35V$ to $\pm 18V$
Gain selection using single resistor	$4 - 10000$
Adjustable voltage reference	$2.5V$, $5V$, $10V$

Table 4.4: INA125Instrumentation Amplifier Specifications

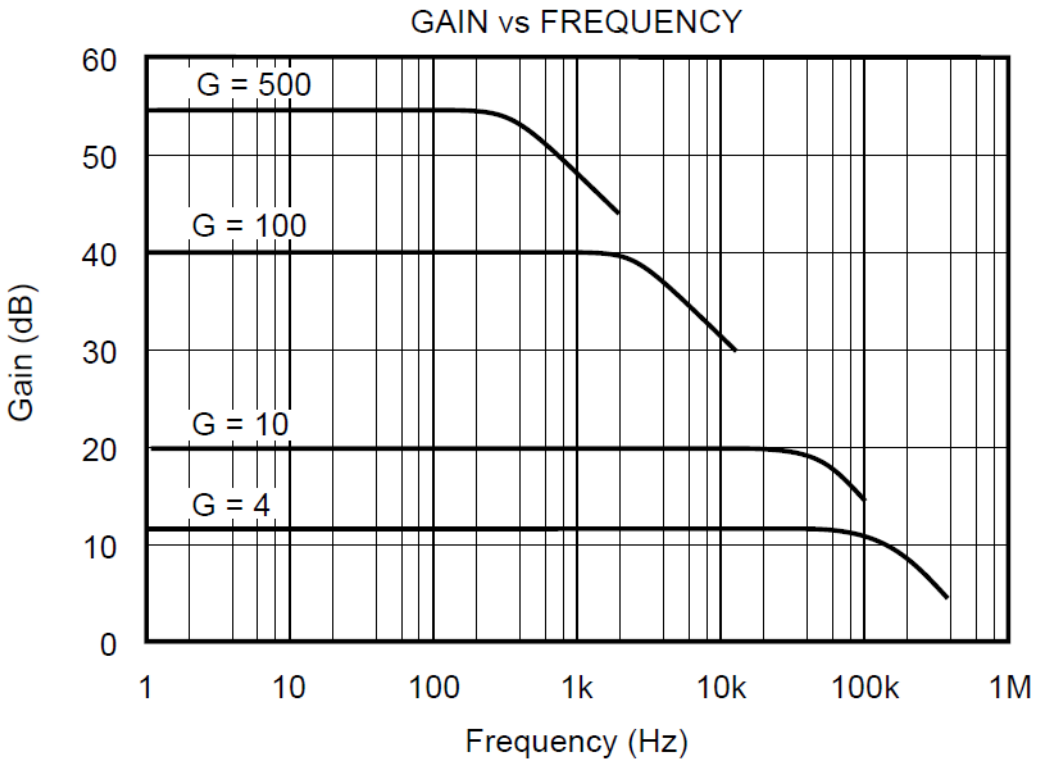


Figure 4.5: INA125 Gain vs Frequency

An additional LPF circuit can be added on board or implemented in software [38], [39]. Since the load cell amplifier boards are required to be general purpose for use with different load cells, the LPF is implemented on software. Figure 4.8 given the circuit schematic and 4.7 gives the board design for the stand alone amplifier boards. The output voltage VO is given by equation 6.2.

$$V_o = V_{offset} + (V_+ - V_-)G \tag{4.2}$$

where,

$$G = 4 + \frac{60k\Omega}{R_G} \tag{4.3}$$

DESIRED GAIN (V/V)	R_G (Ω)	NEAREST 1% R_G VALUE (Ω)
4	NC	NC
5	60k	60.4k
10	10k	10k
20	3750	3740
50	1304	1300
100	625	619
200	306	309
500	121	121
1000	60	60.4
2000	30	30.1
10000	6	6.04

NC: No Connection.

Figure 4.6: Gain values and the resistance required to achieve the gains. Taken from the data sheet [37].

The offset voltage can be adjusted using a potentiometer. For ADC with 3.3V input, the offset should be around 1.65V for load cells that records both compression and tension. The gain values can be adjusted using another potentiometer. The maximum input to the board is 12V.

The Tiva SEA booster pack developed in this work includes a load cell amplifier with slight modifications. Since the TIVA ADC allows 3.3V inputs, the offset potentiometer is removed and a voltage divider is used to supply the required voltage on the IAREF pins. Initially the idea of putting DIP switches to select gain resistors was considered, but due to space limitations on the board, a trimming potentiometer is used to select the gains. Table 4.5 gives the specifications of the load cell amplifier board.

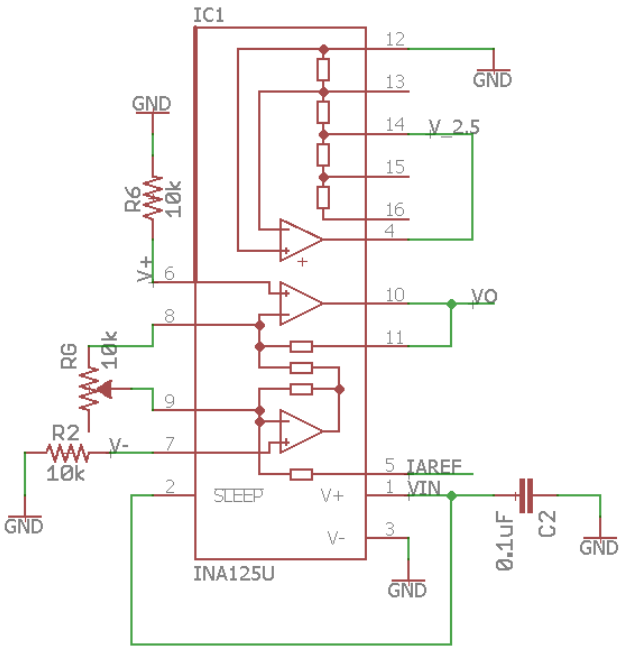


Figure 4.7: Load Cell Amplifier Circuit

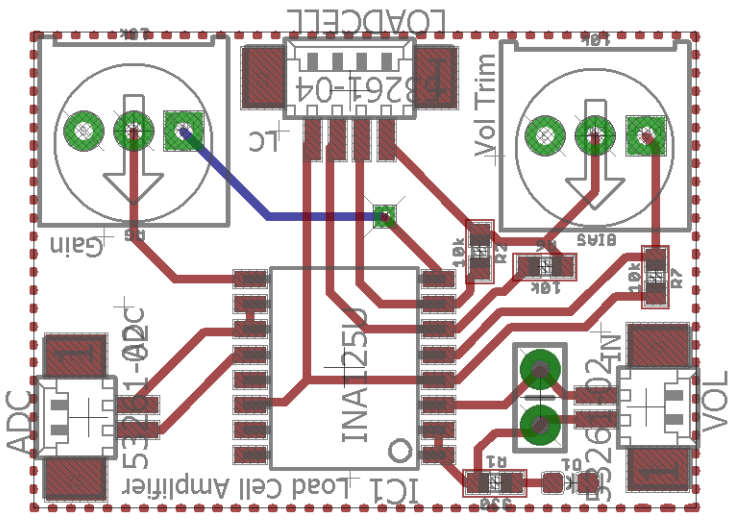


Figure 4.8: Load cell Amplifier Board

Description	***Values***
Input Supply	5 - 12V
Voltage Offset Control	0 - 2.5V
Gain trim	4 - 10000
Load Cell Type	Full Bridge
Max Output voltage	Offset ± 2.5V

Table 4.5: Load Cell Amplifier Board Specifications

4.2.2 Thermocouple

Thermocouples are used for measuring temperature. It consist of two metal wires which produces a voltage due to temperature change between the leads as given by Seebeck effect. These sensors are often a better choice as compared to semiconductor based temperature sensors when the measured temperature range is very high.

A thermocouple is added to the design to measure the temperature of the motor in overdrive mode, where the motor is supplied more current than, its rated capacity and therefore it tends to get heated faster. The temperature reading can then be used to devise a cooling strategy which can be based on pumping coolant or increasing the speed of the fan. For the current work, a type-K thermocouple from Adafruit [40] was used with specification shown in table 4.6.

Description	***Values***
Temperature measuring range	-100°C to 500°C
Output range	-6 to +20mA
Precision	±2°C
Length	1 meter
Thickness	2.18mm

Table 4.6: Type-K Glass Braid Insulated Thermocouple Specifications

Since the voltage output from a thermocouple is extremely small, an amplifier is required to amplify the voltages to a level where it can be read properly. The AD8495 thermocouple amplifier [41] with cold compensation is used as shown in figure 4.9.

The amplifier is powered from 5V supply and produces an analog output. The output temperature is given by equation 6.3. This amplifier works with any K-type thermocouple.

$$Temp = \frac{(V_{out} - 1.25)}{0.005} V \tag{4.4}$$

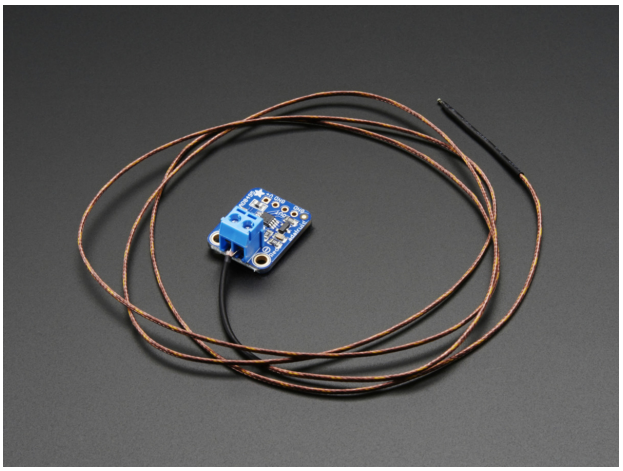


Figure 4.9: K Type thermocouple and AS8495 Amplifier (Source www.adafruit.com)

4.2.3 Absolute Encoders

Each degree of freedom of THOR and ESCHER is equipped with an absolute rotary encoder. These encoders are used to sense the position of the each joints which is required by the high level controllers. The absolute encoder used on THOR and ESCHER is the Gurley Precision Instruments A19 [42] which is shown in figure 4.10.

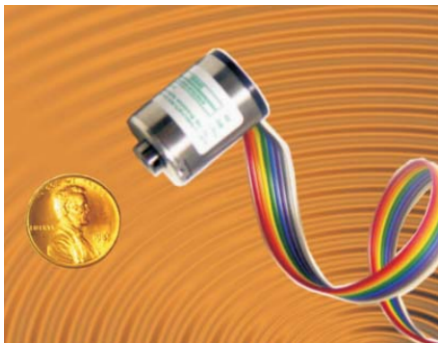


Figure 4.10: Gurley precision A19 Absolute Encoders [42]

This is a single turn encoder with 15 bit output resolution. The output format is SSI and it uses gray or binary code to represent the information. However, both the clock and the output signal are differential which needs to be converted to single ended signals using a RS-485 line driver IC. The output of the IC is then connected to SPI channels on the Tiva board. The 8 pin IC SP490, which is a full duplex RS-485 transceiver is used [43]. The circuit schematic is shown in figure 4.11.

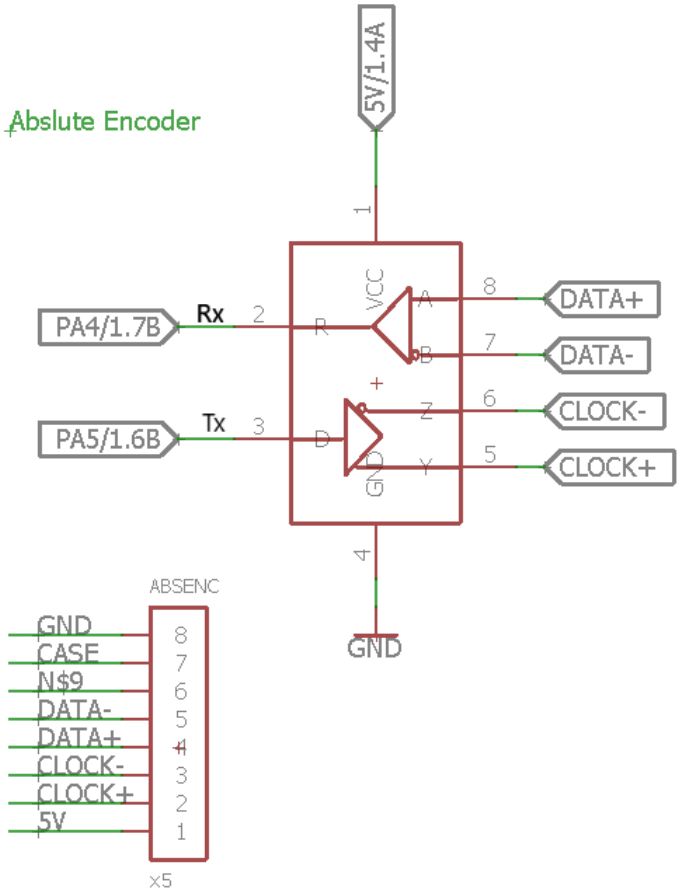


Figure 4.11: RS-485 line driver schematic for absolute encoders

For the design of exoskeleton, it is determined that the A19 encoders could not be mounted due to space constraints. Therefore, Orbis absolute encoders [44] is selected which is shown in figure 4.12. This encoder comprises of diametrically magnetized permanent ring magnet and a circuit board comprising of 8 hall sensors. There is an additional built-in calibration algorithm which improves the accuracy of the encoder. It is a 14 bit resolution sensor with multi-turn counter option and on-board status LEDs. The SSI output is differentially signaled and the same RS-485 driver can be used to convert the signals into single ended. Alternatively, a SPI channel can be used to read the sensor output as the output type can be chosen while placing the orders.

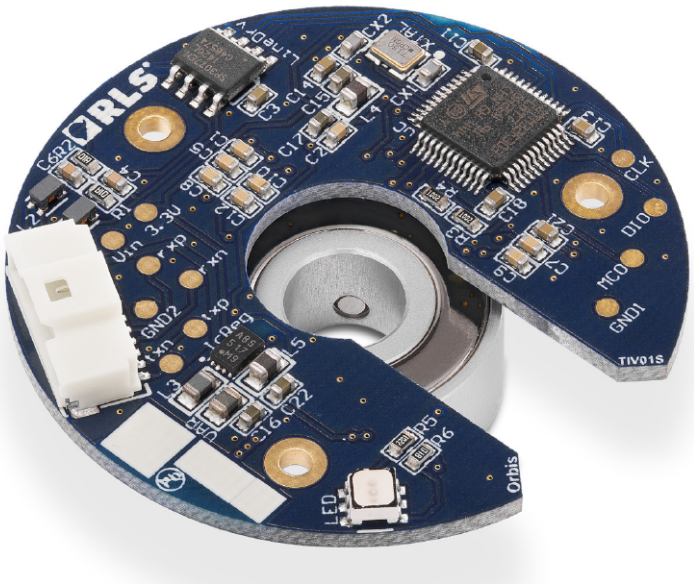


Figure 4.12: Orbis absolute encoders [44]

Another non contact off-axis absolute rotary encoder AksIM [45], shown in figure 4.13, is used for the exoskeleton. This encoder is designed for easy integration in space constrained application. An example of usage of the orbis and AksIM encoders on the exoskeleton in shown in figure 4.14. This encoder is 20 bit resolution and interfaces with SSI or SPI. Table 4.7 gives the specification of all three types of absolute encoders considered in the design.

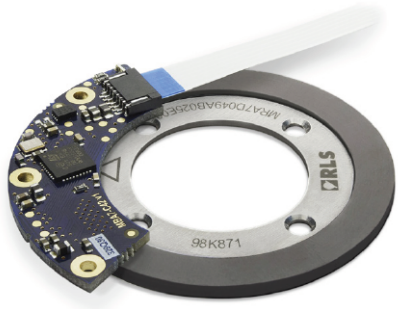


Figure 4.13: AksIM absolute encoders [45]

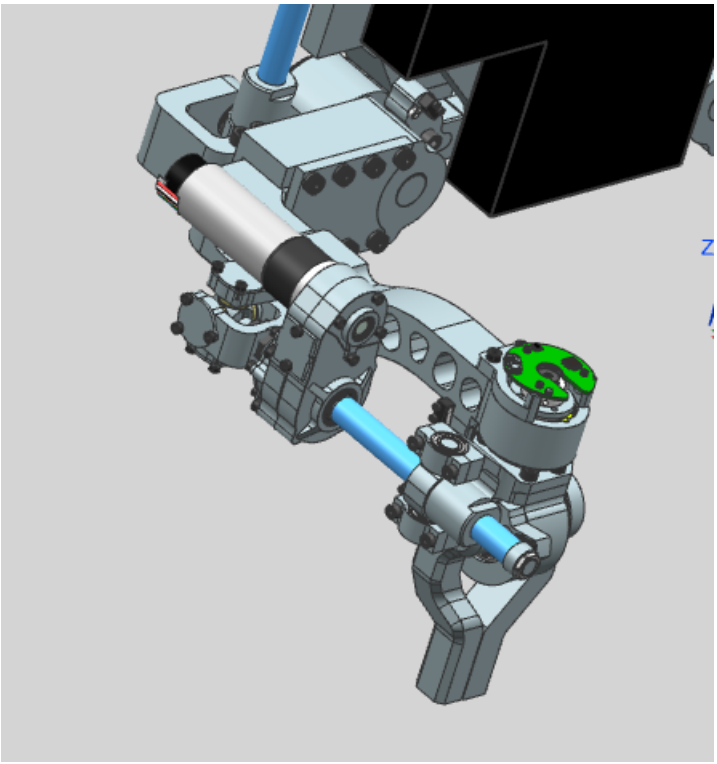


Figure 4.14: Absolute encoder placement on Exoskeleton. Used with permission of Ben Mccuen.

Description	***A19***	***Orbis***	***AksIM***
Max Resolution	15	14	20
Input Voltage	4.75-5.25	4-6	4-6
Max Speed	2400 rpm	10000 rpm	10000 rpm
Output Format	Differential SSI	Differential SSI	Single ended SSI or SPI
Type	Contact	Non contact	Non contact

Table 4.7: Absolute Encoder Specification

4.2.4 Incremental Encoders

The Maxon motors beings used in the SEA have an attached incremental encoder [46]. These encoders are used to calculate the relative position and direction of motion of the motor shaft. The encoder output is in a differential format which needs to be converted to a single ended signal using a RS-485 driver chip. SN75175 is a quadruple differential line receiver with 3-state output [47]. High input impedance, increased noise immunity and input sensitivity

of $\pm 200mV$ over a common mode input voltage range of $\pm 12V$ makes it an ideal candidate IC. The output of the line receiver is then fed to the QeP input on the TIVA board as shown in the schematic in figure 4.15.

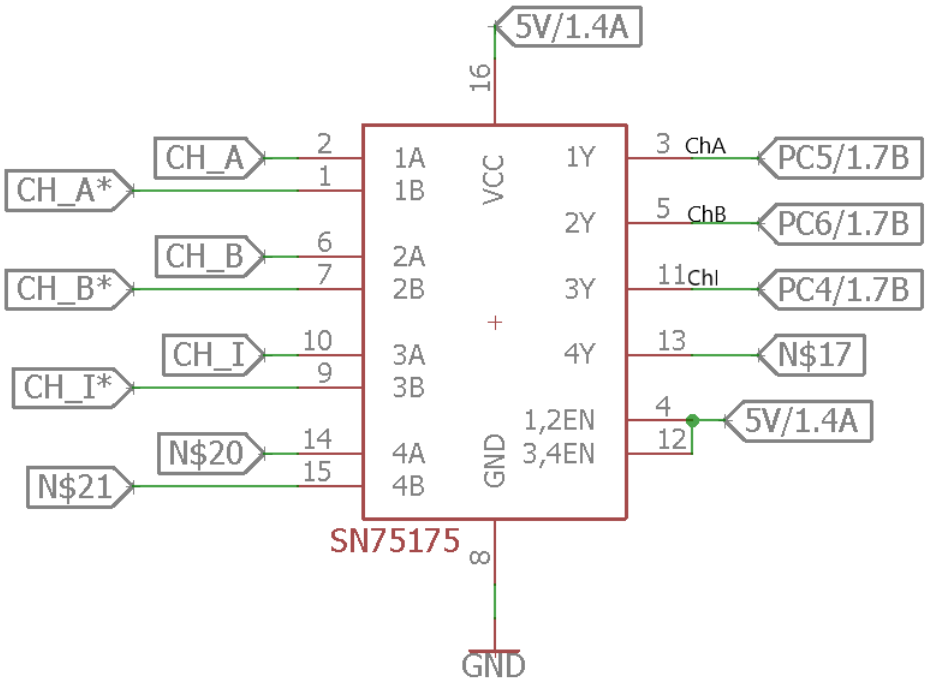


Figure 4.15: Incremental Encoder connection diagrams

The incremental encoders generate massive amount of data. As an example, the 225780 incremental encoder on Maxon motors has 1000 counts per revolution and the maximum speed is 12000 RPM. Therefore, it is necessary to use hardware which can read this data without needing CPU interruption. The Tiva TM4c123 controller has 3 Quadrature Encoder peripheral inputs which can be directly used to interface with high speed incremental encoders.

4.2.5 Additional Sensors and peripherals

Often times, it is required to interface with additional sensors such as IMU, limit switches, current or temp sensors, etc. Due to this, the ability to interface additional sensors was incorporated on the board. Table 4.8 gives the information about the additional channels.

Peripheral	*Number of Channels*
ADC	2
Digital input/Output	3
Debug LED	2
Reset Button	1
I2C	1
SPI	1

Table 4.8: Additional Sensors and Peripherals

4.3 Motor Control

The low level controller, which can be position, force or velocity control takes input from incremental encoders or load cells and generates a commanded output which is proportional to the desired current to the motor. Since micro-controllers are not capable of allowing high currents required to run motors, a servo driver is used.

The low level controller running on the motor slug generates a PWM signal which is sent to the servo drivers. The servo drivers use switching power electronics such as MOSFETs to supply the required amount of current to the brush-less DC motor.

Based on the current requirements of different joints on the proposed exoskeleton as well as on THOR and ESCHER, it was determined that a servo driver with continuous current capabilities of more than 20A is required. An off the shelf servo driver from Advanced Motion Controls AZBDC60A8 is selected [48]. Table 4.9 gives the specification of the driver board.

Description	*Values*
Continuous Current	30A
Peak Current	60A
Supply Voltage	10-80VDC
Max continuous power	120W
Hardware Protection	Over and Under voltage, Over current, Temperature, Short circuit
Input	PWM, Direction
Feedback	Hall Sensor
Input/Output	Fault, current monitor, current reference, Inhibit

Table 4.9: AMC Servo Driver Specifications

These servo drives have inbuilt current sensor which is very useful to monitor the current drawn by the load. The fault out and inhibit-in are also useful features for debug and control.

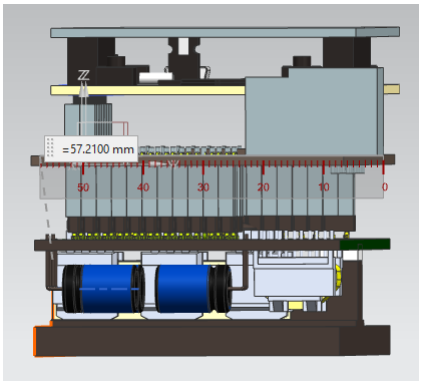


Figure 4.16: Stackable design of motor driver. Used with permission of Ben Mccuen.

A custom PCB was designed to interface the servo driver with the motor slug. As the major design challenge was robustness, a stackable design was chosen which minimizes the number of connectors. From the previous experience with the robots, we have learned that the connectors are major failure points and therefore minimizing them adds to the longevity and robustness of the hardware. Figure 4.16 shows the CAD model for the stackable design.

Figure 4.17 is the PCB layout for the connector board. This is a 2 layer board with debug LEDs connected to fault out, inhibit, current monitor and current reference. The connectors for motors are rated for 20A continuous current while the power supply terminals are directly solder on the board to allow high current transmission. The motor connectors can be replaced easily if required for future upgrades.

The current monitor output of the servo drives has a scaling of $20A/V$. Since the maximum current is $60A$, the maximum output voltage is $3V$ which is safe for the ADC input pin on TIVA board.

4.4 Communication

4.4.1 Logger

Texas Instruments EC-TM4C123GXL Launch Pad board has a virtual serial communication port within the In-Circuit Debug Interface (ICDI). This port is internally connected to the UART0 peripheral on the chip. The board is connected to the computer using a USB cable. Once the software is setup correctly (more in the software development chapter), data is received on a serial port on the host computer. Several terminal software exists such as putty and RealTerm, which can be used for data monitoring as well as logging. For this work, RealTerm was used.

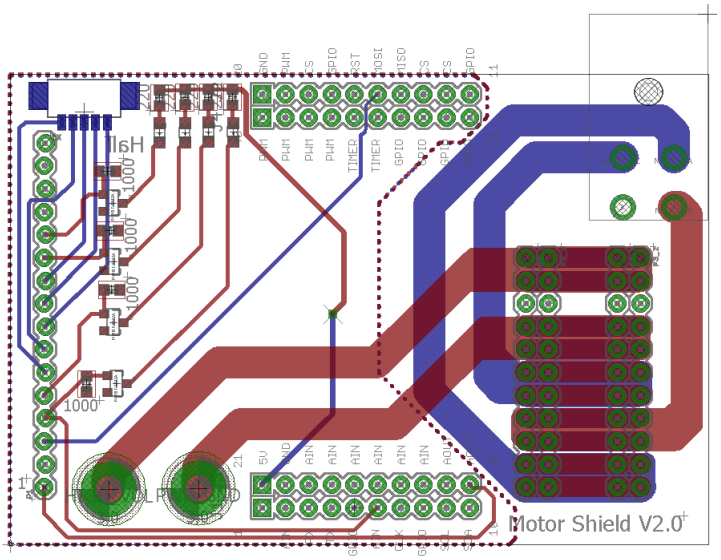


Figure 4.17: Connector board for AMZBDC60A8

4.4.2 CAN

For humanoid robots and exoskeletons, there is a main computer that plans motion trajectories and computes desired joint positions, velocities and accelerations as explained in System overview chapter. In order to compute the joint set points, the high level controller needs information from sensors such as absolute and incremental encoders. Also, the joint set point information has to be sent out to each and every joint on the robot. This set point information is then used by the low level controller to generate the required control effort. Therefore, a two way communication channel is required, which has high speed and is bidirectional and safe.

Controller Area Network is used as a standard communication protocol in automobiles [49]. CAN is now being used in various applications because of its low cost and simplicity. We employ a CAN based communication network. EtherCAT networking was also considered, but because of its complex nature and requirement of external hardware, CAN bus was finally chosen. TIVA TM4C123 has 2 CAN interface built into the board.

The CAN interface on board on TIVA produces single ended inputs/outputs. However, CAN specifications require a differential signal which can be obtained using a CAN transceiver. The MCP2551 high speed CAN transceiver was used which acts as the CAN physical layer, and also protects the micro-controller pins from high voltage transients [50]. Figure 4.18 shows the schematic for the transceiver.

As shown in the schematic, a 120 ohm terminating resistor is required by CAN bus specification at the end of each branch. However, the TIVA board discussed here can be used both

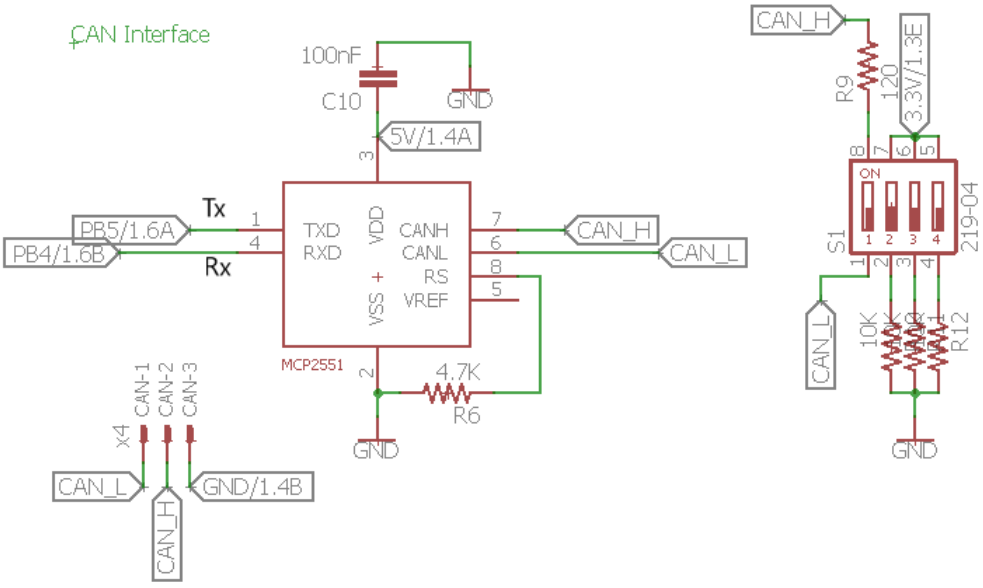


Figure 4.18: Circuit schematic for CAN transceiver

as an intermediate node or a terminal node. Therefore, in order to achieve the functionality of selecting or removing the termination resistor, a DIP switch in series with a 120 ohm resistor is used.

4.5 TIVA SEA Shield Layout

The shield for the TIVA TM4C123 board is described in this section. It is a 2 layer board which has all the converters, amplifiers and connectors necessary to interface with the sensors and communication peripherals on board. Table 4.10 highlights the board capabilities.

The input voltage to the shield is 12V. There is a 80% efficiency, 2W, DC/DC converter that provides the stable 5V supply required to power the TM4C123 processor. The 12V input is also used to power the load cell amplifier circuit. Test points are placed strategically to help in debugging.

The on-board load cell amplifier has only one gain trimming potentiometer. The voltage offset is set to 2.5V. Table 4.11 shows the pins to which the peripherals are wired. This table can be used as a reference for programming the controller and changing channels if needed in future revisions.

Features	***Description***
On-board power converter	12V/5V
Load Cell	Amplifier and ADC connection
Incremental Encoders	Signal conversion IC and QeP
Absolute Encoders	SPI
Thermocouple	ADC
Servo driver	PWM, Dir, Current
LED	2 power, 2 debug
Switch	1 reset switch
Spare channels	I2C, ADC, SPI
Test points	voltage and GND
Communication	Transceiver and CAN channels, UART

Table 4.10: Tiva SEA shield

The pins for motor control are connected using the daughter board in between TIVA and AMC servo driver as explained in section 4.3. The thermocouple is connected to the amplifier board which in turn is connected to the shield. The thermocouple leads come to the amplifier directly and therefore an off-board amplifier was chosen to allow better control on board placement on the robots. The amplifier board draws 5V power from the shield and is connected directly to an ADC channel.

An absolute encoder interfaces with the SPI channel through an on-board RS-485 transceiver as discussed in section 4.2.3. The incremental encoders also use a RS-485 line driver for conversion of differential signals to single ended. The output of the driver is then connected to the Quadrature Encoder Peripheral on the TIVA board. These encoders power from the 5V on-board supply.

The TIVA board inherently supports a virtual serial communication based on the UART. The pins PA0, PA1 is connected to the USB port and is used for data logging. A Reset button is installed for easy accessibility. Two debug LED is present on-board for a programmer's needs. Several additional channels are configured and ready to use. Figure B.1 shows the PCB design for the SEA shield. The Eagle schematic is attached in the appendix.

Peripheral	***Channel***
Load Cell	PE3
Test points	voltage and GND
Motor Control	PWM - PF1, Dir - PB7, Current Monitor - PE5
Thermocouple	PE0
Absolute Encoder	Tx - PA5, Rx - PA4
Incremental Encoder	ChA - PC5, ChB - PC6, ChI - PC4
Serial Communication	Rx - PA0, Tx - PA1
CAN	Tx - PB5, RX - PB4
Additional ADC	PE2
SPI	Rx - PF0, CLK - PF2, CS - PF3, Tx - PD3
I2C	SCL - PA6, SDA - PA7
Debug LED	Blue - PD7, Yellow - PC7
GPIO	PB6, PF4, PE1
Button	Reset

Table 4.11: Peripherals and Pin assignment

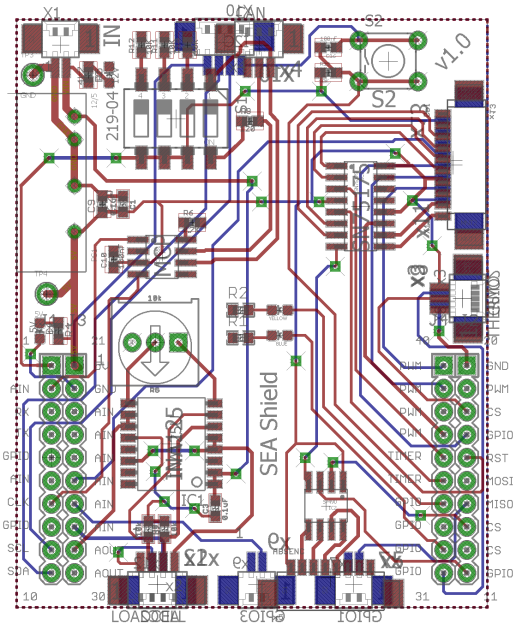


Figure 4.19: PCB Layout of SEA Shield for Tiva Launchpad

4.6 Power Distribution Board

The entire robot needs different voltages to operate different components. Table 4.12 shows the various voltage requirements. The schematic is attached in the appendix. Figure C.1 shows the board layout. The board contains several 24V and 12V output pins for interfacing with a logic board that controls the e-stop, fans and pumps, display LCD, etc. There are on board indication LEDs for every voltage bus. The test points help in easy debugging and the presence of fuses makes the board safe. The design of the power distribution board is based on that being used at IHMC for Mina V2 Exoskeleton [32].

Peripheral	***Voltage Required***
Battery Supply	48V
Servo Drivers	48V
Fans or pump	24V
Main Computer	15V
Motor Slugs	12V

Table 4.12: Power Requirements on the Robot

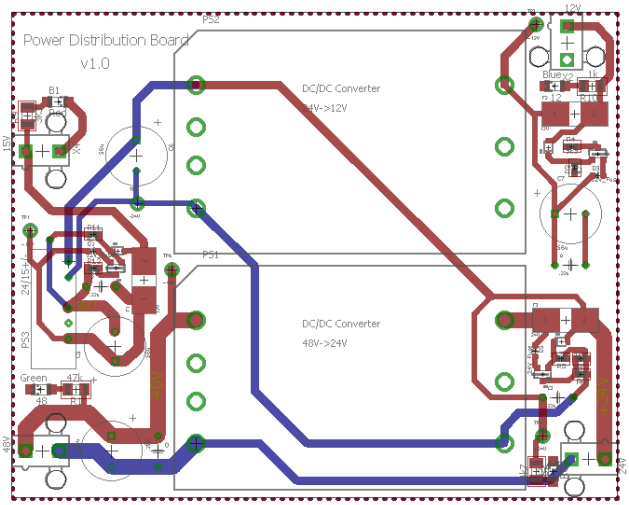


Figure 4.20: Power Distribution Board

Chapter 5

Software Development and Validation

5.1 Real Time Operating System

Firmware for embedded controllers can be based on either interrupts or use a dedicated operating system. A simple application such as that of reading a few sensors and performing simple computations can be easily handled by writing software as a super loop or having timer based interrupts. However, more complex functionality such as that required for humanoid robots need strict timing requirements and can cause problems if these requirements are not met. Furthermore, in safety critical applications such as exoskeletons, the hardware and software has to be fail safe. This leads to the need of Real Time Operating System (RTOS) for writing firmware [51]. Some of the most important characteristics of an RTOS is:

- The threads can run in real time which means that the latencies are minimum and can be calculated i.e. there is a worst case latency associated with each and every process.
- The Kernel calls are deterministic.
- The scheduler is pre-emptive i.e., the highest priority tasks runs first.
- The RTOS gives users various methods that helps in message passing and communication between the threads.
- More efficient use of CPU resources.

Figure 5.1 gives the structure of an RTOS based embedded application. The BSP stands for the Board Support Package which is a hardware driver specific to the micro-controller being used. RTOS Kernel on the other hand comprises of several API such as scheduler, timer, task manager, inter-thread communication, dynamic memory allocation etc.

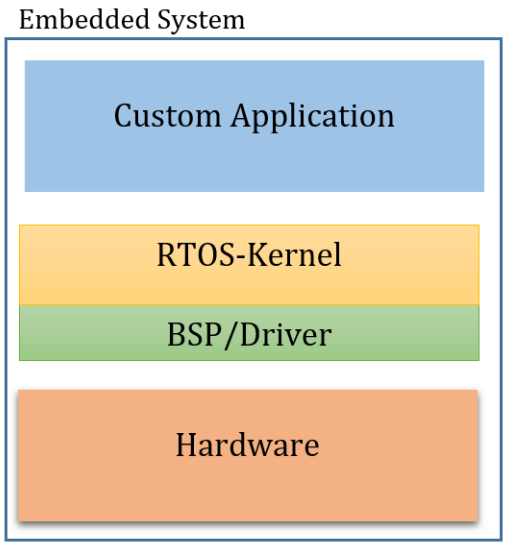


Figure 5.1: Structure of a RTOS

Tasks or threads are the basic block of processing for an RTOS. The task manager is responsible for scheduling different tasks. These scheduling algorithms can be either clock driven or priority driven.

A clock driven scheduling is one in which the scheduling time of the tasks are known a priori and the clock drives events based on the calculated timing. Figure 5.2 shows an example of a clock driven system where the goal is to toggle an LED. Every few millisecond, an event is scheduled which either turn on or off an LED.

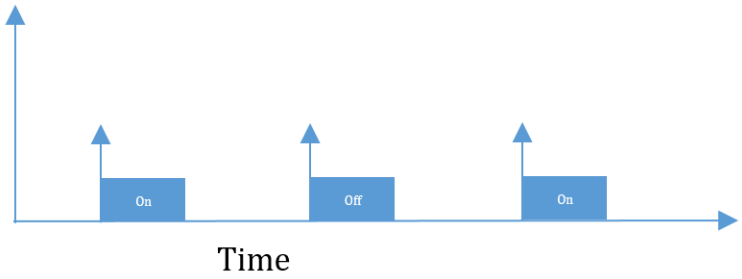


Figure 5.2: Clock Driven Scheduling

In complicated applications, it is often difficult to understand the strict time in which an event would be ready and therefore priority based scheduling is used where the scheduling decisions are made when the task is available and based on the priorities of other tasks ready to be scheduled. In these systems, the higher priority task can preempt a lower priority one.

RTOS also offers inter process communication (IPC) services as safe and secure communi-

cation between different threads are of utmost importance for an RTOS application. Some of the most widely used IPC primitives are mutex, spinlock, semaphore etc. An example to demonstrate the need of IPC is the access (read/write) of a memory location by two threads. If this access is not protected by a locking mechanism then incorrect information can be used by the threads. Finally, RTOS includes services for memory and resource management, which is responsible for dynamic memory allocation, resource access and use etc.

5.2 TI-RTOS

The Microcontroller development board of our choice, TI TIVA TM4C123 supports RTOS such as FreeRTOS and TI-RTOS. TI-RTOS is chosen because of several advantages such as availability of learning material, RTOS specially designed for the hardware, free plug-ins for Code Composer Studio (CCS), etc. TI-RTOS is composed of the real time kernel, the driver libraries and the file system and communication packages as shown in figure 5.3.

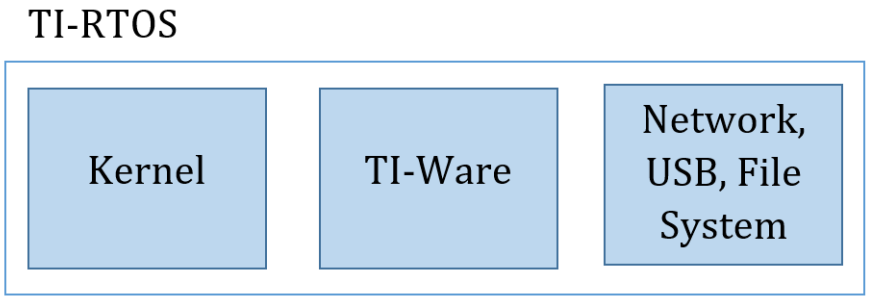


Figure 5.3: TI-RTOS Software Stack

One of the most useful feature of the TI-RTOS is the inbuilt debugging tool. The RTOS supports Unified Instrumentation architecture (UIA) which is a target side API that allows users to add debug features and visibility such as function calls to printf and log. It also supports thread loading and execution graphs. Another debug feature, RTOS Object Viewer (ROV) enables the user to interrogate every BIOS object in the system. It also allows the user to determine the status of the stack/heap. These debugging tools take fewer cycles, often less than 50, to gather data and the processing and display of data is performed only after the processor halts and thereby not loading the processor [52]. In this work, these features have been used at multiple places for trouble shooting, generating results and understanding the performance of the designed system.

5.2.1 Thread Types

TI-RTOS supports 4 different types of threads which are functions running within a specific context (such as priority) as shown in figure 5.4. Each thread group has a defined priority and within each group there can be multiple priorities. The RTOS has a BIOS Scheduler which is responsible for scheduling and running the pending tasks [52].

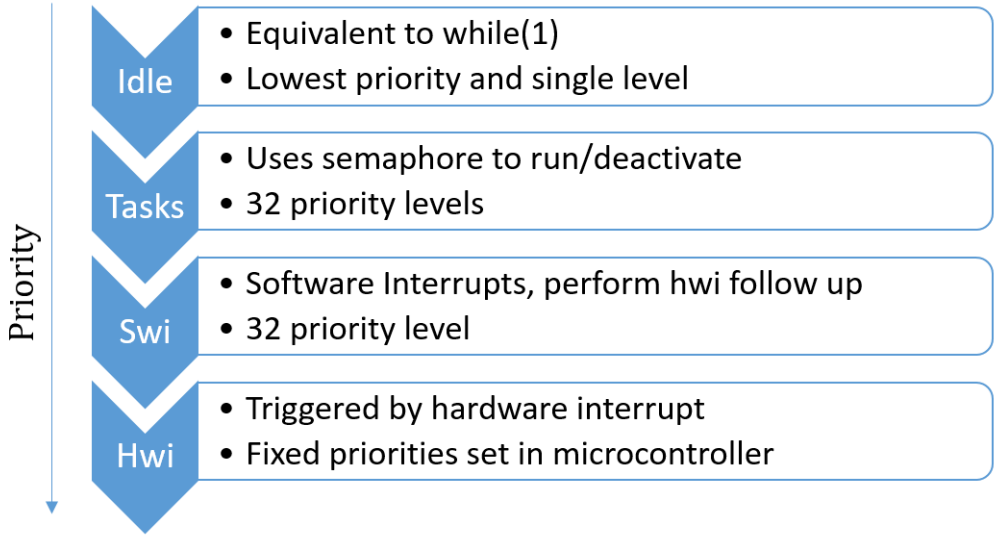


Figure 5.4: TI-RTOS threads

Idle Thread

The Idle thread has the lowest priority and is equivalent to the infinite while loop. This thread is processed by the scheduler only when none of the other high priority threads require attention. Appendix details the step by step process of creating an RTOS project in code composer studio and adding different threads to it.

Hardware Interrupt

The HWI or the hardware interrupt has the highest priority amongst all threads. These threads are associated with interrupts occurring due to time or an external event such as UART, ADC conversion etc. The priority of HWI is determined on silicon and can be obtained from the data sheet of the MCU [34]. Since the priority is fixed, it is recommended to minimize the processing being done inside an HWI.

Software Interrupt

A good design approach is to post a thread which performs follow up activities for HWI. This thread is the SWI or the software interrupts. An SWI typically does follow up activities for HWI. An an example, an HWI is responsible for acquiring data from an ADC and a SWI can then implement a filter on the acquired data.

A SWI can have a priority between 1 and 32. SWI is scheduled based on priority. If two SWI with same priority occurs at the same time, it is executed in a first come first serve fashion.

Tasks

Tasks are threads which run continuously in a system. They make the system truly concurrent. The main difference between Task and SWI is that Tasks have their own stack and therefore a local memory while SWI uses the global stack. Tasks are enabled or disabled using a RTOS service known as semaphore which acts as a flag and controls the switching. Like SWI, Tasks also have 32 priority levels.

5.2.2 RTOS Scheduling

An RTOS has several threads of different priorities running concurrently in the system. The scheduler is responsible for deciding the order in which the threads should be executed. The use of correct scheduling policy is of utmost importance for ensuring that all threads are executed properly without missing any deadlines.

Some of the scheduling algorithms available in TI-RTOS are:

- Deadline Monotonic : Used with fixed priority preemptive scheme, where tasks with shortest deadlines are assigned the highest priority.
- Rate Monotonic: A fixed priority scheme where threads with smallest period is assigned highest priority.
- Stu Monotonic : Opposite to Rate Monotonic, this scheme assign lowest priority to threads with smallest periods.

In chapter 6, scheduling analysis is discussed and performed for a sample application.

5.2.3 Inter Thread Communication

An application with multiple threads requires some sort of communication between the threads. In general embedded style programming globals are generally used for storing data which is used system wide such as sensor values. Protected access of globals is challenging as it is available freely system wide. In an RTOS, this problem of sharing data between threads is solved by used of RTOS services such as mailboxes, semaphores, etc. Given below is a short description of these services along with their use.

Queues and Mailboxes

In an application where the threads follow a producer consumer model, queues or mailboxes can be used. A producer thread generates data such as sensor reader thread. The consumer thread on the other hand uses the data such as the filtering thread. In this model, the consumer thread waits for the producer thread to generate data. The blocking mechanism is implemented using semaphores. Examples of data access in this fashion is given in the Application chapter.

Mutex

If threads need to access data concurrently, mutex is used. A mutex uses a mutually exclusive semaphore which protects the critical section of the code. Use of mutex can lead to problems such as priority and deadlocks. Therefore, careful use is required. In this work, Mailboxes and Queues are used. However, in several applications and scenarios, the use of mutex is absolutely required.

5.3 Board Support Package

A Board Support Package (BSP) is the low level driver that helps in safe access of hardware peripherals on the development board of choice. Having a BSP acts as an abstraction layer between writing application software and low level register based programming. A BSP was written for the TIVA SEA shield discussed in the hardware development chapter. The package uses TI's Tivaware libraries which can be used for both RTOS as well as non-RTOS applications.

The main motivation behind the development of a BSP was to allow ease of programming. The next few sections show how easy it is to work with different peripherals using the BSP. Also, adding an RTOS layer becomes easier as the programmer doesn't need to worry about dealing with registers and other low level programming work.

5.3.1 Debug LED

The TIVA shield has a yellow and a blue debug LED. Given below is the sample code for initialization and use of the on-shield blue debug LED. The code toggles the blue led at a specified rate. The function $BlueLED_Init()$ initializes the pin, unlocks it and makes it ready for a GPIO output functionality. A timer module can be added to control the timing of the toggle. Several functions for control of RGB led and Buttons on TIVA board is implemented in the BSP. The users need to make sure that these pins on the TIVA boards are not being by the TIVA SEA shield peripherals.

```
#include "slug.h"
int main(void){
    Clock_set_40MHz();

    toggleBlueLED();
}

void toggleBlueLED(){
    BlueLED_Init();

    while(1){
        BlueLED_Set();
        delayMS(100);
        BlueLED_Clear();
        delayMS(100);
    }
}
```

5.3.2 Timer and Interrupt

To add a timer interrupt, a timer needs to be initialized and then a period has to be determined for the interrupt generation. The toggle led routine is now called though the timer handler. If an RTOS is being used, the timer handler is called though the RTOS thread and no changes have to be made in the startup file.

```
void testTimerToggle(){
    BlueLED_Init();

    // Define toggle frequency
    int freq = 10; //10Hz frequency of toggling

    // Initialize Timer0A
```

```
    initTimer0(freq);

    // Enable system wide interrupts
    EnableInterrupts();

    while(1){
    }
}
```

The programmer needs to decide which timer they want to use and call the corresponding timer initialization function. The timers can be initialized as periodic or one shot. In periodic mode, a frequency is defined for timer initialization. The actual code for processing in written in the interrupt service routine of the timer which is shown below:

```
void Timer0IntHandler(void){
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    toggleBlueLED();
}
```

The timer handler can call any function such as ADC filtering, PID control, Logger etc.

5.3.3 Logger

The Motor Controller Stack can talk to a remote computer using the UART peripheral on board. The logger interface uses printf statements which take up to 100 CPU cycles. A less intrusive method of logging is the use of an external memory card over SPI peripheral. There are two methods of logging data. If one shot logging is required, the *initconsole*() function can be used. A unit test is given below with the output shown in figure 5.5.

```
void testConsole(){
    int BaudRate = 115200;
    bool flag = 1;
    int i = 0;
    initConsole(BaudRate);

    UARTprintf("System Startup");

    while(1){
        if(flag==1){
            UARTprintf("%d \n", i);
            i++;
        }
```

```
        if(i==10){
            flag = 0;
        }
    }
}
```

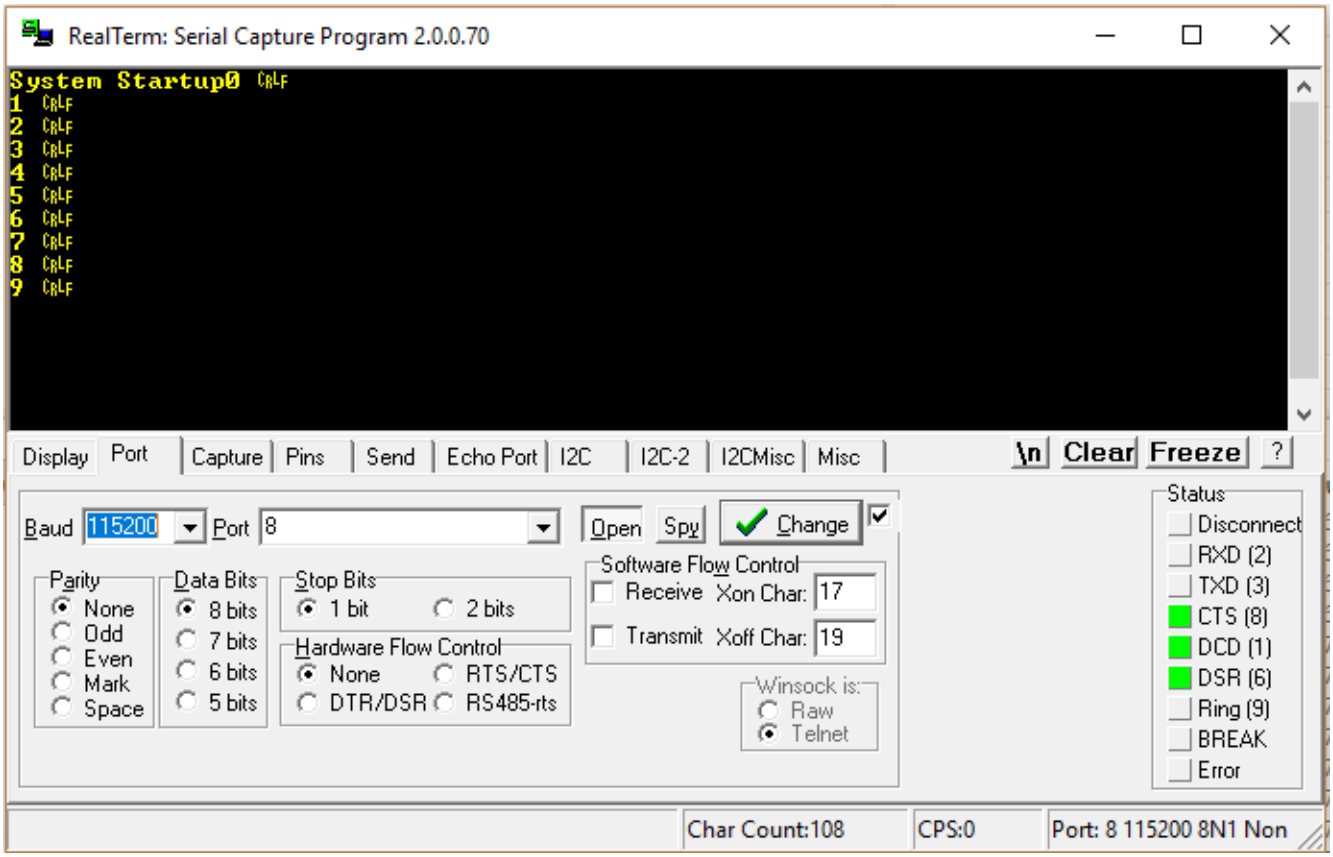


Figure 5.5: Console Output

When periodic logging is required, which is generally true when logging system data, the logger initialization routine is called. The logger uses timer 2 for periodic logging. The code snippet given below initializes the logger.

```
void testLogger(){
    int BaudRate = 115200;

    uint32_t loggerFreq = 10; //10 Hz

    Logger_Init(loggerFreq, BaudRate);
```

```
    UARTprintf("System Startup");

    while(1){
        // Do nothing
    }
}
```

The data that is being logged is defined in the interrupt service routine for the logger as shown below. Here, a global variable is being incremented and published at a rate specified in the initialization routine.

```
void LoggerIntHandler(void){
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);

    globalDummy += 1;
    UARTprintf("%d \n", globalDummy);

}
```

5.3.4 Temperature Sensor

The TI's TM4C123GH6PM has a temperature sensor on-chip. This sensor can be used to sense the internal temperature of the chip and notify the system if the temperature is too high or low. It can also be used for calibration of the hibernate module of the Real Time Clock trim value. The temperature sensor is wired internally to ADC0. Every ADC has an interrupt associated with it to report the sampled value. Therefore, the temperature sensor initialization routine initializes the ADC0 peripherals and configures the interrupts. The ADC conversion process starts by calling the respective *startConversion*() function. The ADC0 interrupt handler receives the converted data and updates the corresponding variable. To keep the access of global variables safe, getters and setters are defined. The *getAvgTemp*() function fetches the current value of the average temperature. Functions for converting the ADC values into temperatures in Celsius and Fahrenheit is defined.

```
void testTempSensor(){
    tempSensor_Init(hardwareAverage);
    initConsole(BaudRate);
    EnableInterrupts();

    while(1){
        //Trigger temp read
        tempSensor_startConversion();
```

```
            raw = getAvgTemp();
            tempC = convert2C(raw);
            tempF = convert2F(raw);
            UARTprintf("Avg temp is: %4d\n", tempC);
            delayMS(1);
        }
}
```

The raw ADC samples often catch external noise which can be eliminated by the use of hardware averaging which also acts as a low pass filter. The higher precision results do cost extra samples and thus affect the throughput. The hardware allows averaging of up to 64 samples accumulated in the sequencer FIFO. Figure 5.6 shows the results obtained without hardware averaging while the figure 5.7, 5.8 and 5.9 show the results with hardware averaging of 2, 8 and 16 samples respectively. A stark improvement in performance can be noted, though the use of hardware averaging of 8 samples. The difference in performance between 8 samples and 16 samples averaging is hardly noticeable and therefore 8 samples averaging can be safely used in this case. The user can configure the hardware average value during temperature sensor initialization. The allowed values are 2, 4, 8, 16, 32 and 64 samples. If the input is 0, no hardware averaging is performed.

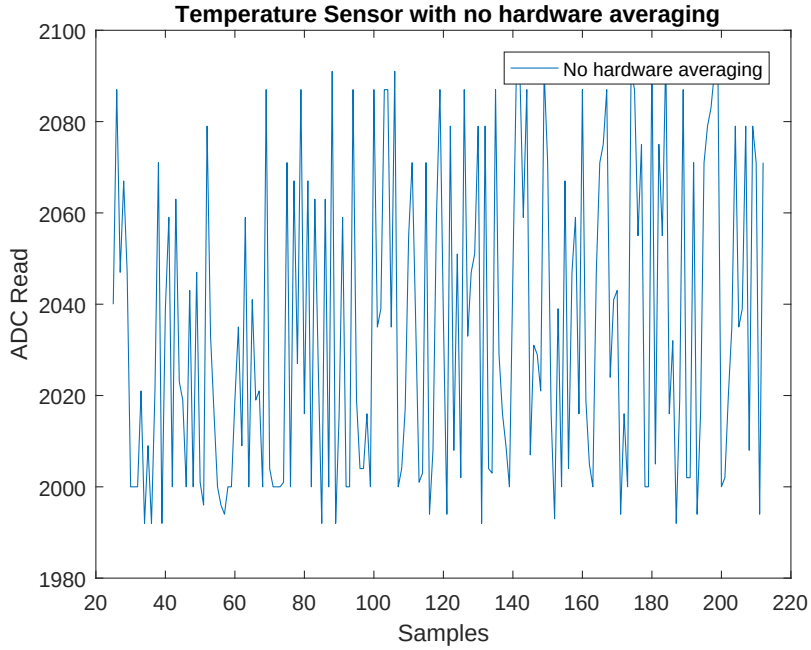


Figure 5.6: Temperature sensor with no hardware averaging

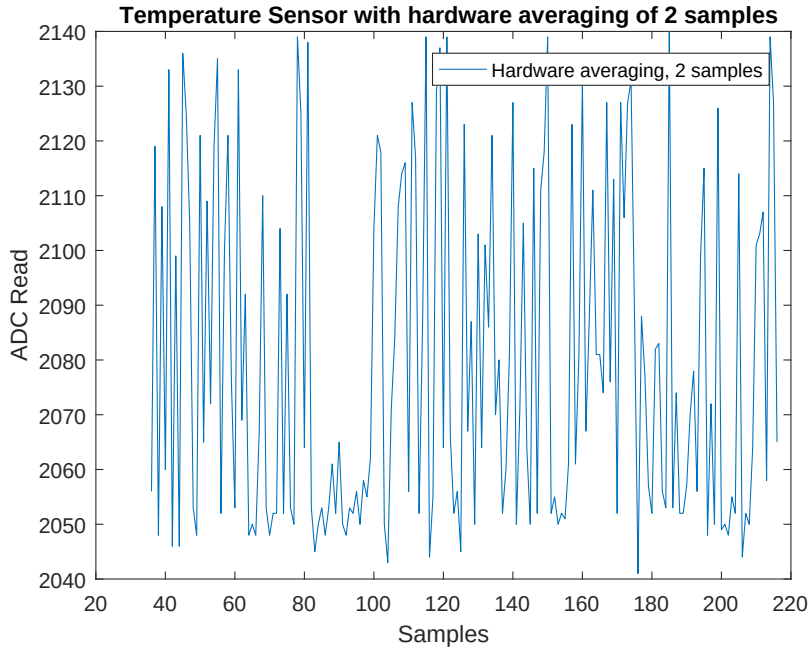


Figure 5.7: Temperature sensor with hardware averaging of 2 samples

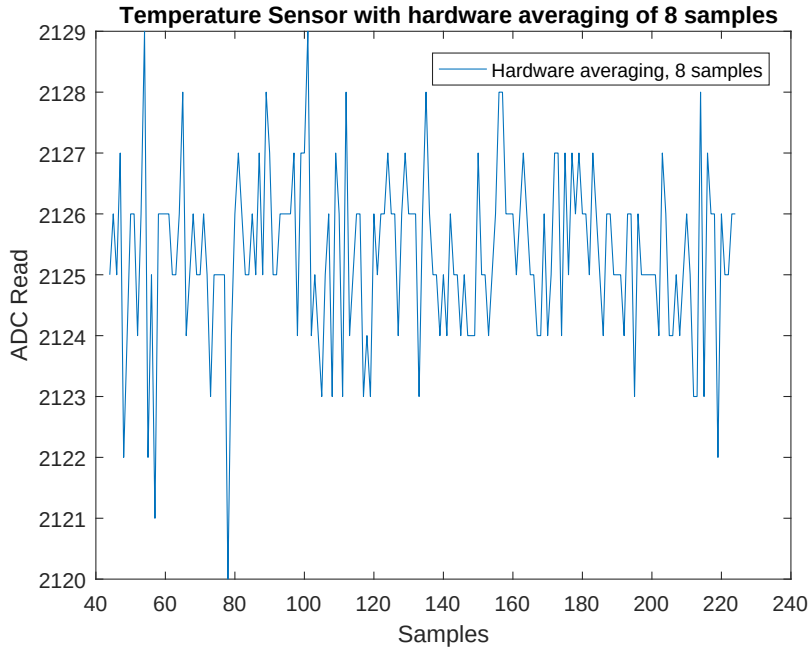


Figure 5.8: Temperature sensor with hardware averaging of 8 samples

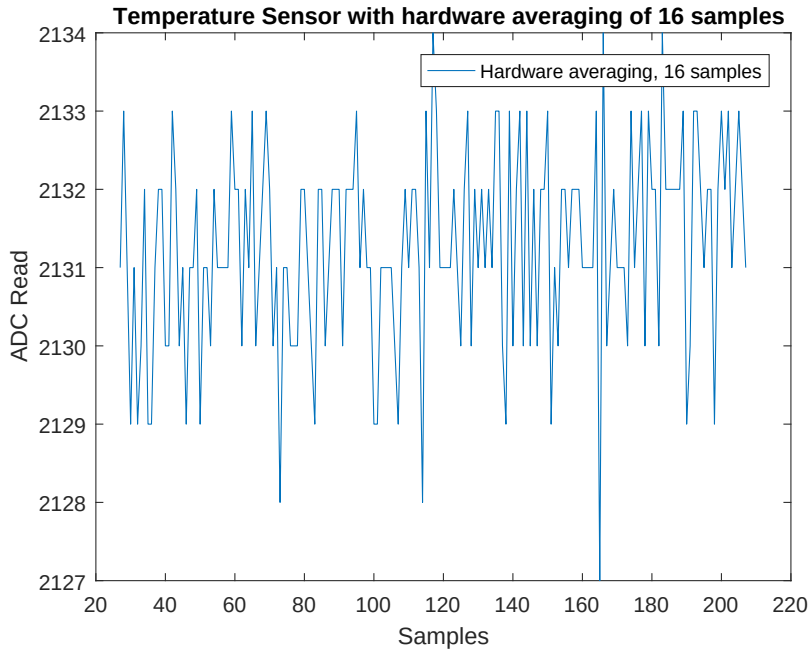


Figure 5.9: Temperature sensor with hardware averaging of 16 samples

5.3.5 Motor

In order to control the motors, the direction of rotation, the frequency of PWM and the duty cycle is specified in the software. Given below is the test routine to test full leg swing on THOR.

```
void testMotor(){
    uint32_t pwmFreq = 10000; // 10KHz
    Motor_Init(pwmFreq);
    while(1){
        motorSendCommand(10, 1);
        delayMS(1000);
        motorSendCommand(10, 0);
        delayMS(1000);
    }
}
```

The motor is initialized by calling the initialization routine and specifying the desired PWM frequency. The PWM and the Direction pins are configured and initialized. The Motor is then controlled by sending the desired Duty cycle which is a number in between 0-100 and the direction of rotation which can be 0 or 1 to the *motorSendCommand*() function. A 0 duty cycle turns the motor off which is helpful in designing controllers. Several other

functions are implemented in the BSP which can be used to get the current duty cycle, PWM frequency and direction of motion command being used.

5.3.6 Load Cell

The load cell is connected to the PE3 pin and ADC0, sequencer 3 is used for data collection. The initialization routine configures the pin as ADC input, initializes the sequencers, configures the interrupts and enables them. A timer can be used to act as the trigger source. By default, Timer0 is used which can be easily changed based on the application requirements. The trigger frequency can be selected during initialization.

```
void testloadCell(){
    int hardwareAveraging, ADCsampleFreq;

    hardwareAveraging = 4;
    ADCsampleFreq = 500; //500 Hz

    LoadCell_init(hardwareAveraging, ADCsampleFreq);
    EnableInterrupts();

    while(1){
        rawLoadCellVal = getLoadCellValue();
        LoadCellVol = adc2Vol(rawLoadCellVal);
        delayMS(1);
    }
}
```

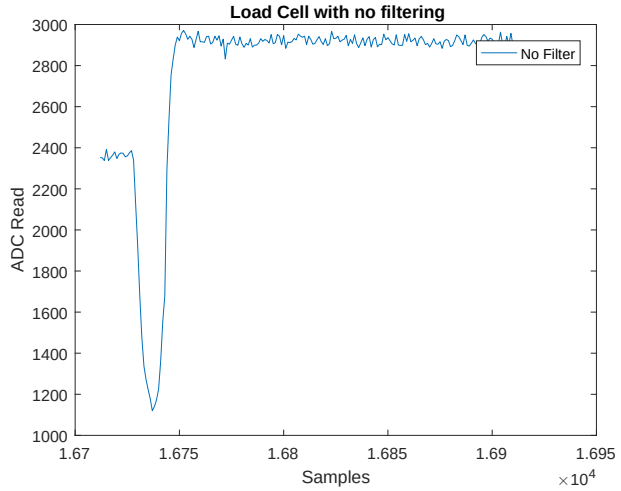


Figure 5.10: Load Cell with no hardware filtering

Figure 5.10 shows the raw load cell measurement obtained when the load cell is installed on the knee SEA of THOR. Random disturbance is applied to see the performance of the amplifier. Figure 5.11 repeats the same experiment but uses hardware low pass filtering. The results are improved. Finally, figure 5.12 shows the results obtained when the leg of THOR is executing a swinging motion. The load cell values are predictable and the amplifier is shown to work as expected.

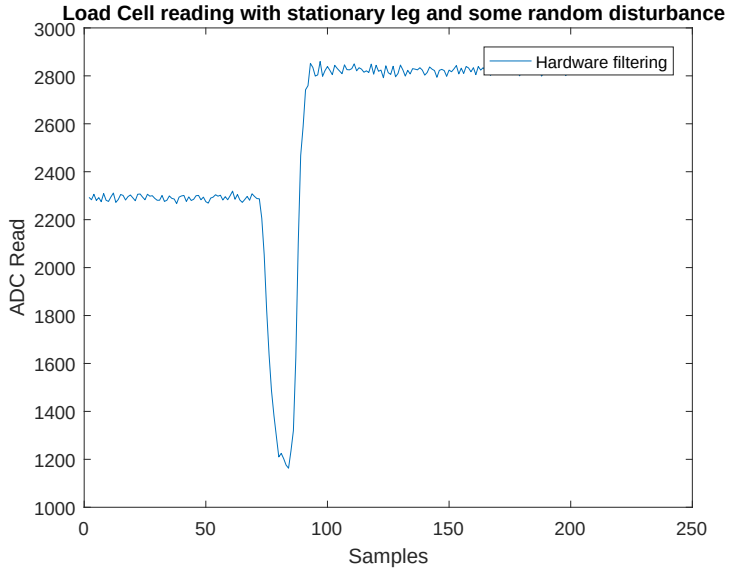


Figure 5.11: Load Cell with hardware filtering

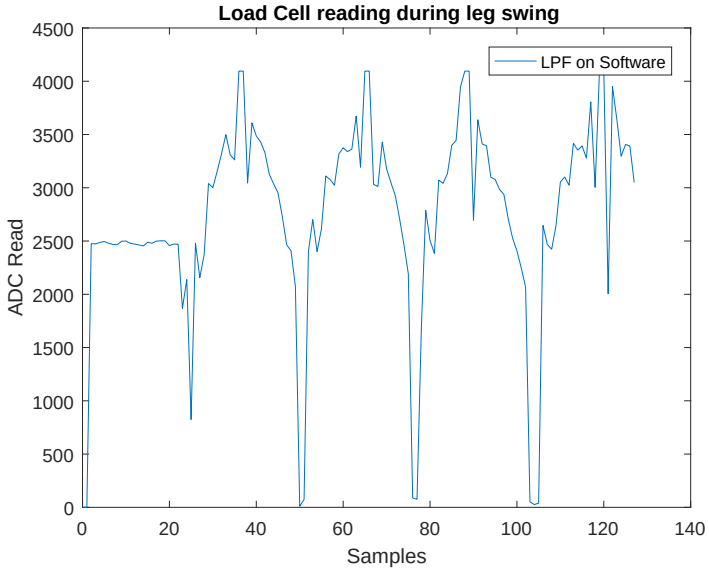


Figure 5.12: Load Cell with hardware filtering and swinging leg motion

5.3.7 Controller

The controller function provides handles for experimentation with different control algorithms such as PID, LQR, Adaptive control etc. A test feed forward controller for achieving leg swing on THOR is shown below. The feedforward controller is responsible for achieving leg swings based on the trajectory specified by the high level controller.

```
void testController(){
    // Initialize Console
    int BaudRate = 115200;
    uint32_t loggerFreq = 100; //100 Hz
    Logger_Init(loggerFreq, BaudRate);

    // Initialize motor
    uint32_t pwmFreq = 20000; // 20KHz
    Motor_Init(pwmFreq);

    // Initialize Load Cell
    int hardwareAveraging, ADCsampleFreq;
    hardwareAveraging = 4;
    ADCsampleFreq = 500; //500 Hz
    LoadCell_init(hardwareAveraging, ADCsampleFreq);


    // Initialize controller
    uint32_t controllerFreq = 2000; //2kHZ
    Controller_Init(controllerFreq);
    ControllerEnable();

    // Enable all interrupts and channels
    EnableInterrupts();

    // For debugging
    RGBled_Init(0, 0, 1); //Blue

    while(1){
        // Do nothing
    }
}
```

As this is an application, several peripheral needs to be initialized such as Motor, Logger, Load cell and LED for debugging. The initialization routine for the controller requires a controller frequency. The controller is implemented on timer 1 on the hardware. The

interrupt service routine for the timer 1 calls the relevant controller as shown below.

```
void ControllerIntHandler(void){
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    // feed forward leg swing
    Swing_control();
}
```

The *Swing_control*() function implements the controller algorithm. Figure 5.13 shows the performance of the controller. The load cell along with a hardware LPF is used. The noise is the load cell reading is because of the vibration in the test stand. SInce the test stand is the knee of THOR, there is an induced noise due to movement of the other degrees of freedom. A better result is shown in figure 5.14 where the swing motion is slower.

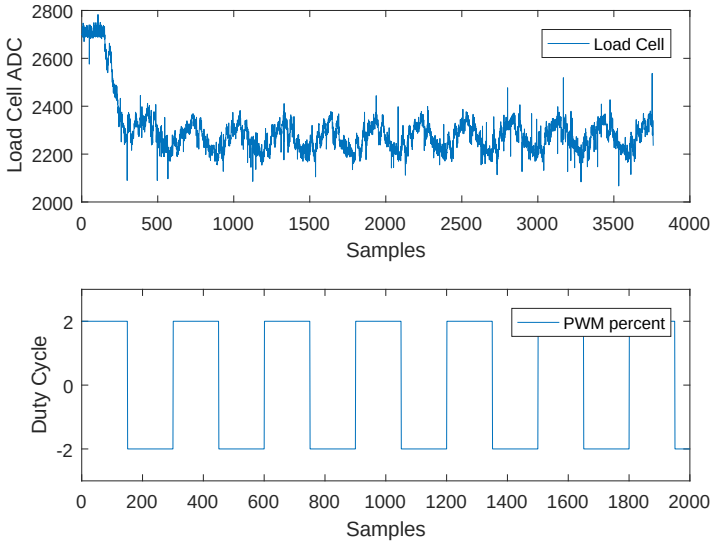


Figure 5.13: A feedforward controller to achieve leg swing

The logger interrupt handler calls the logging function which can be defined based on use. Below is the logging function used for this experiment. Figure 5.15 shows the logger output.

```
void print_loadCell(){
    int rawLoadCellVal, log_dir;
    uint32_t log_duty;
    loggerCount++;

    log_duty = getglobalduty();
    log_dir = getglobaldirection();
    rawLoadCellVal = getLoadCellValue();
```

```
        UARTprintf("%d,%d,%d,%d\n",loggerCount,rawLoadCellVal,log_duty,log_dir);
}
```

The next chapter discusses two more application which requires use of PID control and Adaptive control strategies. The use and benefits of RTOS is also discussed in the next chapter.

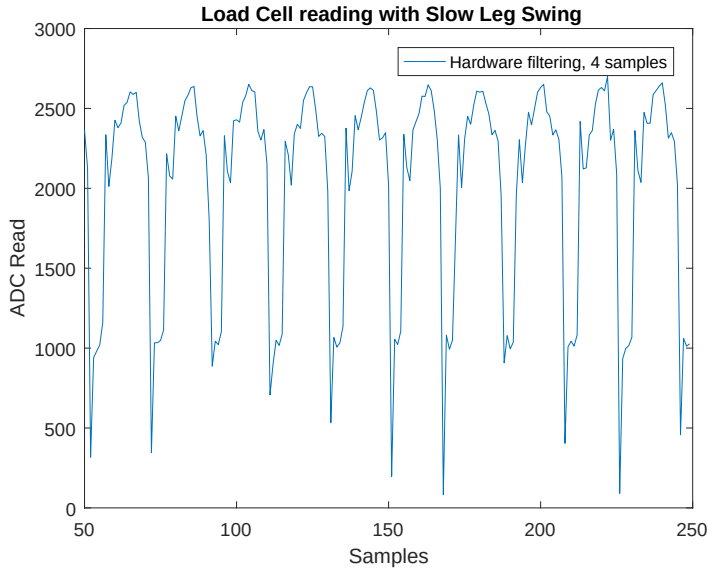


Figure 5.14: Load Cell with hardware filtering and swinging leg motion (SLOW)

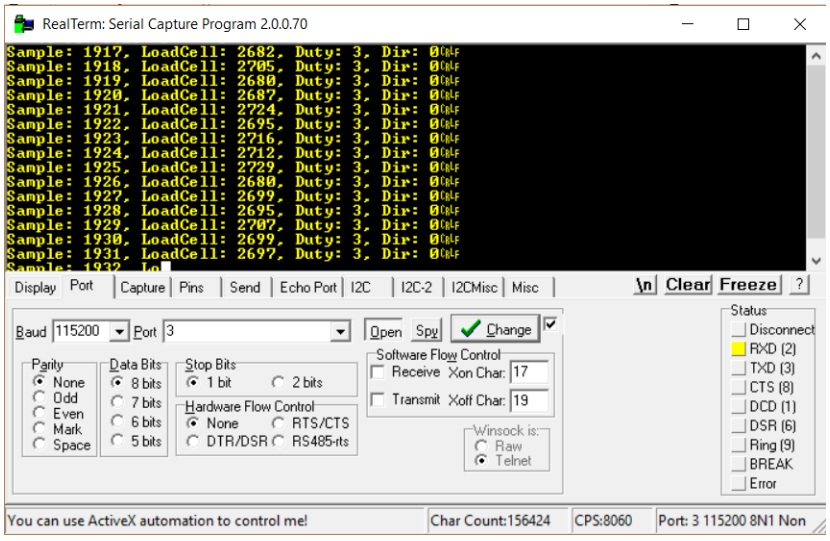


Figure 5.15: Logger output for feedforward controller to achieve leg swing

Chapter 6

Applications

This chapter provides two applications where the motor controller architecture has been shown to perform as expected. The ease of use of the hardware and the modularity of the software significantly reduced the time spent on prototyping. Both the application deal with implementing force control of the Knee joint on THOR. In the first application, a PID force control law is proposed, implemented and validated. Several features such as dead band, integral wrap up, gain scheduling, etc. is used to demonstrate the capabilities of the hardware and software.

In the second application, a first order adaptive control law is proposed and then implemented to solve the force control problem on the knee joint of the robot. PID Force control application is implemented in real time as well as non real time setting. Analysis is presented to prioritize threads in RTOS application.

6.1 PID Force Control

The SEA on THOR is modeled using a second order forced mass-spring-damper system as shown is figure 6.1 and discussed in [10]. Here, k_s is the spring stiffness, f is the external force, l_s is the displacement of the mass, f_M is the linear force applied by the motor, M_{EQ} is the lumped spring mass and b_M is the damping coefficient due to friction. A transfer function between the output force F and the motor current I_M as proposed in [10] is shown below in equation 6.1.

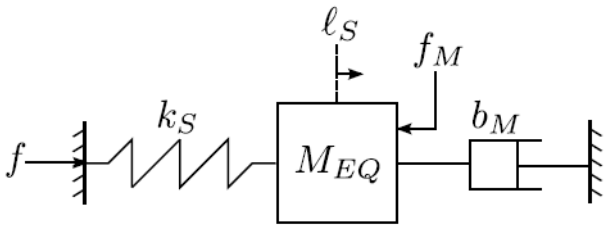


Figure 6.1: Mass spring damper model of SEA. The picture is taken from [10]

$$\frac{F}{I_M} = \frac{k_f \frac{k_s}{M_{EQ}}}{s^2 + \frac{b_M}{M_{EQ}} s + \frac{k_s}{M_{EQ}}} \tag{6.1}$$

The force F is the output force that is measured by the load cell. A system identification is performed in [10] to obtain the unknown parameters in equation 6.1. The final transfer function is given in equation 6.2.

$$\frac{F}{I_M} = \frac{11358.64}{s^2 + 3.823s + 50.126} \tag{6.2}$$

Figure 6.2 shows the pole zero map. Figure 6.3 shows the open loop response of the plant for different step input. The plant is stable but has extremely slow response time. Figure 6.4 shows the bode plot of the open loop plant.

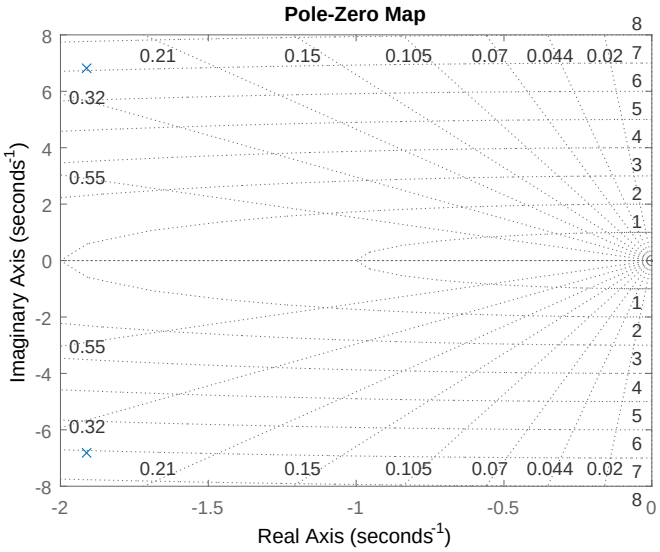


Figure 6.2: Pole Zero Map of the Open loop plant

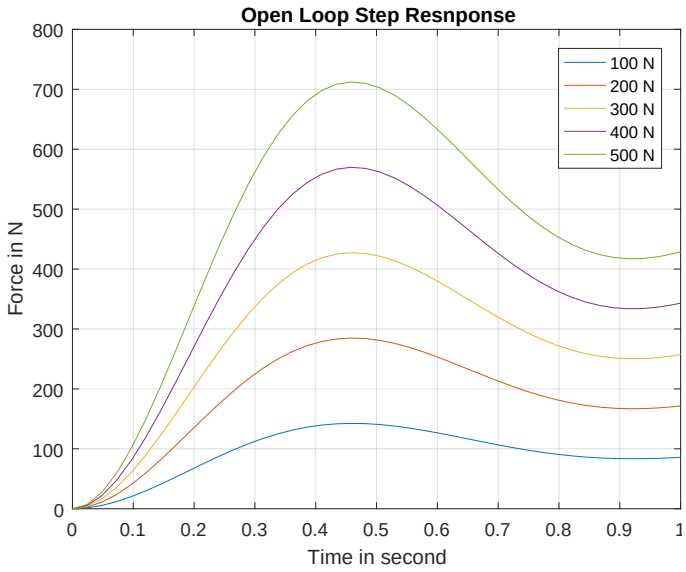


Figure 6.3: Open loop Step Response

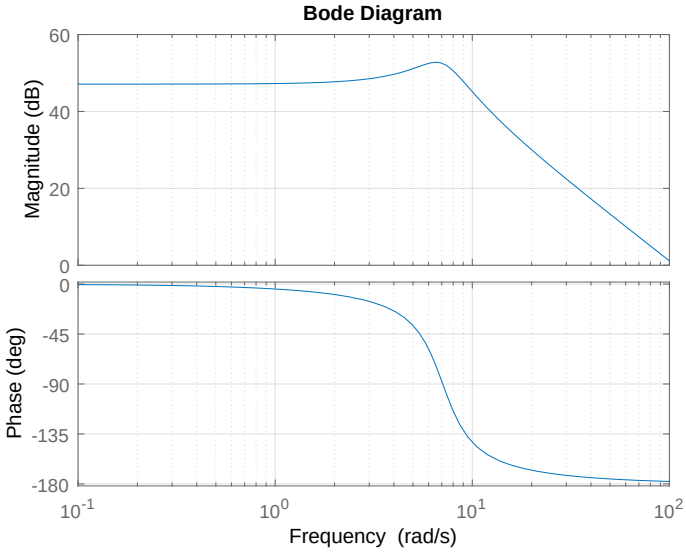


Figure 6.4: Open loop Step Response

6.1.1 Simulation results

A force control loop as shown in figure 6.5 is used. The force controller proposed here relies on a feedforward term and a PID control law. The measurement is obtained through the load cell in-line with the SEA. A desired force is commanded by either the high level controller or the low level impedance controller.

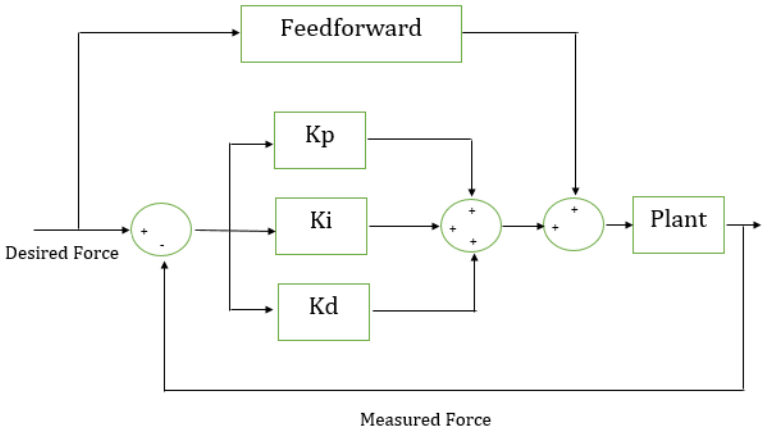


Figure 6.5: PID Force Control loop

Simulation is performed to understand system behavior with PID control. The PID gain values of $K_p = 0.01$, $k_d = 0.001$ and $K_i = 0.05$ is used. Figure 6.6 shows the closed loop step response. The rise time is less than 0.2 second and the settling time is around 1 second. The system has very small overshoot of less than 5%. Figure 6.7 shows the bode plot of the system with the PID controller.

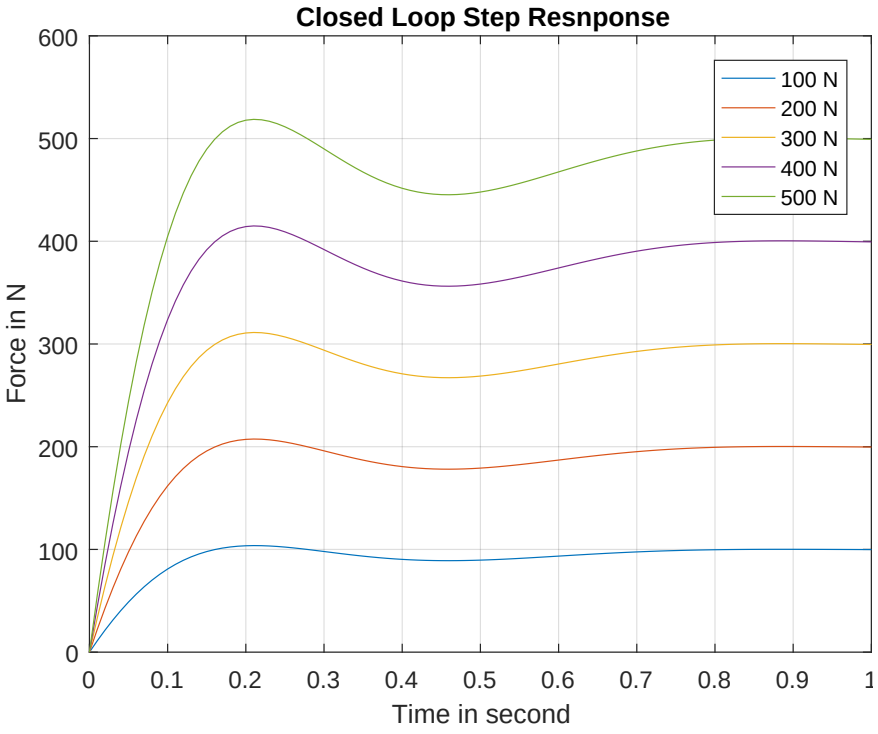


Figure 6.6: Step response of closed loop system

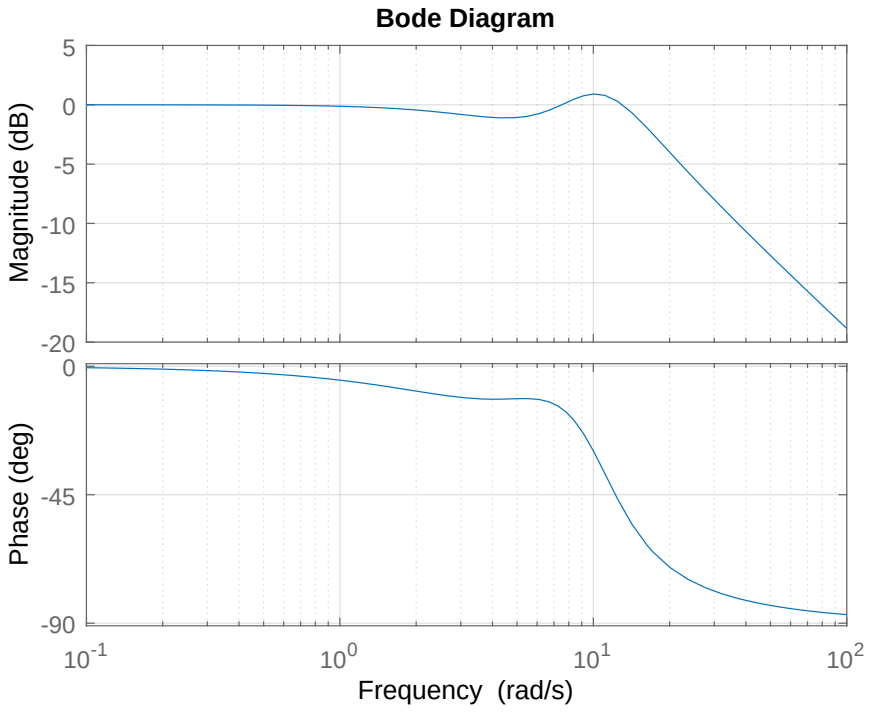


Figure 6.7: Step response of closed loop system

6.1.2 Hardware Implementation

For this application, the goal is to control the knee pitch on THOR. Figure 6.8 shows the experimental setup. The feedforward term is used to offset the D.C. component of the error. PID control acts on the error signal given by $e(t) = f_d(t) - f_m(t)$. The output of the PID controller is the desired current to be sent to the motor controller. The PID control law is given by equation 6.3. A first order low pass filter is applied to the derivative term to compensate for the high frequencies in the error signal.

$$PID(s) = K_p + K_i\frac{1}{s} + K_d s \tag{6.3}$$

As unmodeled dynamics of a SEA can result in poor tracking performance and therefore a disturbance observers can be designed as discussed in [1]. Instead of designing a disturbance observer, adaptive control strategies were experimented with in this work. The PID performance is further improved by implementing integral windup, deadband and gain scheduling.

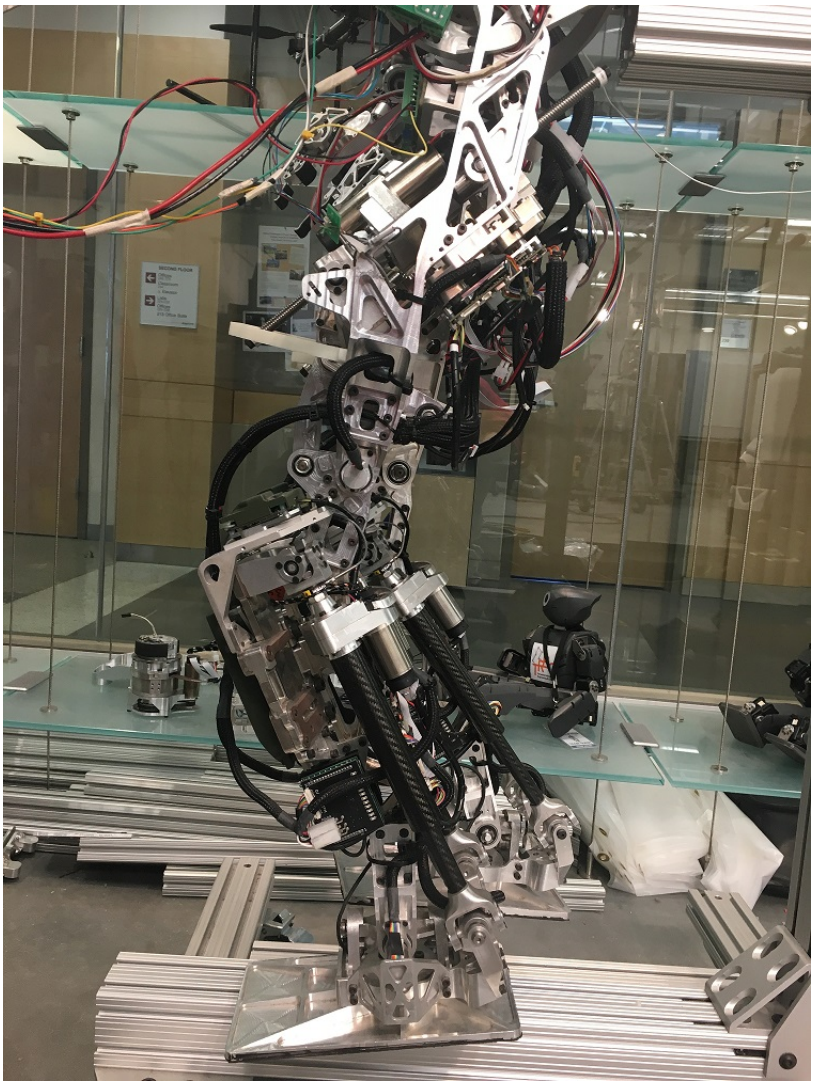


Figure 6.8: Experimental Setup, Knee joint on THOR

Given below is the PID Force control implementation as a non-RTOS project.

```
#include "slugTest.h"
double setPointForce = 50;
int main(void){
    // Initialize clock at 80 MHZ frequency
    Clock_set_80MHz();

    // Initialize Console
    int BaudRate  = 115200;
    uint32_t loggerFreq = 1000; //1 KHz
    Logger_Init(loggerFreq, BaudRate);
```

```
    // Initialize motor
    uint32_t pwmFreq = 20000; // 20KHz
    Motor_Init(pwmFreq);

    // Initialize Load Cell
    int hardwareAveraging, ADCsampleFreq;
    hardwareAveraging = 4;
    ADCsampleFreq = 1000; //1 KHz
    LoadCell_init(hardwareAveraging, ADCsampleFreq);


    // Initialize controller
    setGoalForce(setPointForce);
    uint32_t controllerFreq = 2000; //2kHZ
    Controller_Init(controllerFreq);
    ControllerEnable();

    // Enable all interrupts and channels
    EnableInterrupts();

    // For debugging
    RGBled_Init(0, 0, 1); //Blue

    while(1){
        // Do nothing
    }
}
```

The controller interrupt handler calls the PID control loop as shown below.

```
void ControllerIntHandler(void){
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    PID_control();
}
```

The PID control loop then executes the following PID algorithm. The algorithm checks for dead band, incorporates integral wrap up and has gain as a function of error.

```
void PID_control(void){
    //check if goal reached
    if(~getGoalFlag()){

        //calculate error
        ERROR = (getGoalForce() - measuredLoad());

        //Goal reaching criteria
        if(ERROR < 0.05){
            setGoalFlag(1);
            RGBled_Set(0, 1, 0); //turn on Blue
        }
            Kp = Kbar*abs(ERROR);
            P = Kp*ERROR;

            D = Kd*(ERROR-LAST_ERROR);
            LAST_ERROR = ERROR;

            TOTAL_ERROR = TOTAL_ERROR + ERROR;
            TOTAL_ERROR = checkIntegralLimit(TOTAL_ERROR);
            I = Ki*TOTAL_ERROR;

            PID_OUT = P + D + I;
    }else{
        PID_OUT = 0;
    }
    checkLimits(PID_OUT);
}
```

Several experiments were performed to evaluate the behavior of the controller. The step response performance is shown in figure 6.9. The load set point is 50 pound. The response is as expected. The video for the experiments can be found in [53].

The samples are collected at a frequency of 100 Hz. Therefore, each sample corresponds to 10 millisecond. Based on this, the settling time obtained is .520 seconds. The oscillations in the response is mainly due to vibrations induced in the setup due to motion of the robot. Table 6.1 gives the gain values that is used in the experiment. Several different set points are also used and the controller works as per the requirements.

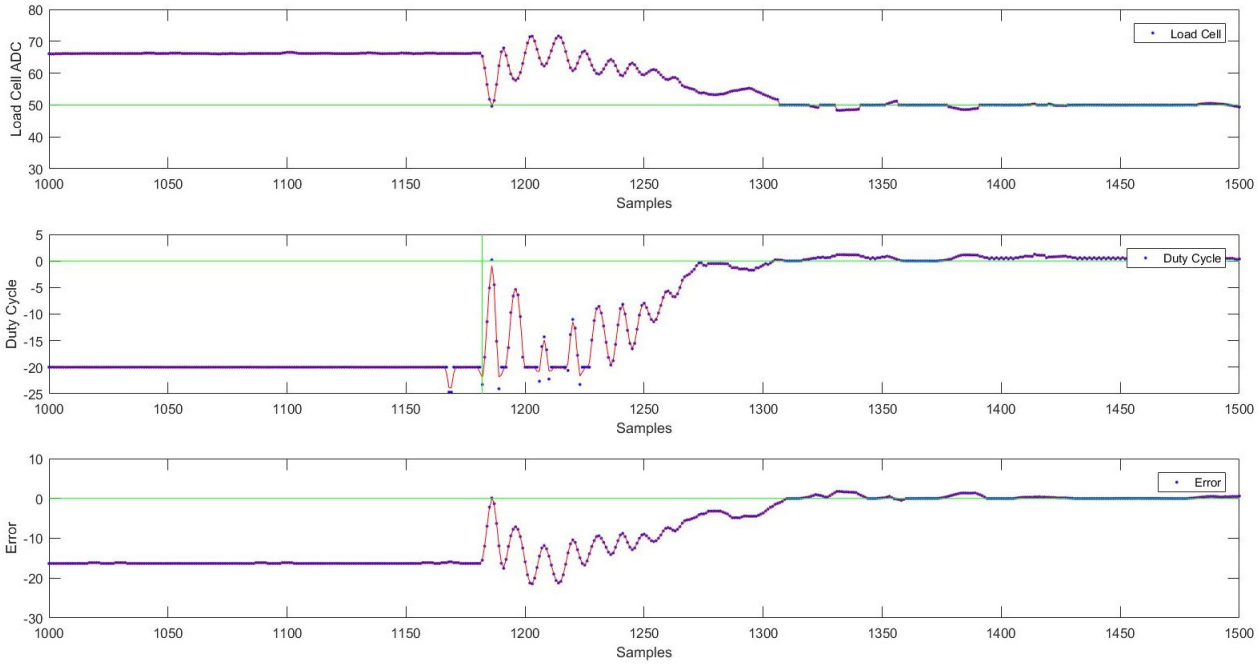


Figure 6.9: Closed loop Step Response on Hardware

PID Terms	***Values***
K_p	$K_P = K_{bar} * error$
K_{bar}	0.1
K_i	0.01
K_d	0
Dead band	0.01

Table 6.1: PID Gain values

6.1.3 RTOS application

Any application can be designed as an RTOS or a Non-RTOS using the board support package. A simple application of angular velocity control of a differential drive robot is used in [54] to demonstrates the need of RTOS when CPU load increases and the application becomes complex. In this work, the PID force control application was implemented on an RTOS.

To understand the RTOS implementation, table 6.2 shows all the Hardware and the software interrupts used in the PID Force control application. The table 6.2 also lists the use of the thread and the thread type.

Some experimentation with priorities can help in choosing the value that works best for the application. However, this approach can lead to failure in worst case situations. Therefore, a methodological scheduling analysis is performed to ensure that each task can meet its deadline even under worst case condition. While the CPU load is an important criteria in understanding the system behavior, a complete scheduling analysis is required for safety critical systems such as humanoid robots and exoskeletons.

Thread Name	***Thread Use***	***Thread Type***
$LoggerHwi$	Triggers Logger	hwi
$LoggerSwi$	Prints data	swi
$MotorHwi$	Send Duty cycle and direction command	hwi
$LoadCellTriggerHwi$	Triggers the ADC	hwi
$LoadCellADCHwi$	Read ADC sample	hwi
$LoadCellFilterSwi$	Filter Load Cell readings	swi
$ControllerTriggerHwi$	triggers the controller and sends motor commands	hwi
$ControllerSwi$	Run the control algorithm	swi

Table 6.2: Threads on the PID Force Control RTOS Application

For this work, Rate Monotonic Analysis (RMA) is performed. RMA guarantees that all tasks will meet their deadlines, if a few design conditions are satisfied. If tasks are assigned priorities based on RMA, there is a formal guarantee that no task will skip its deadline if the CPU utilization is less than 69.3% [55]. This condition can further strengthen if the tasks have harmonic periods (multiples of each other) such as 1, 10, 100 and so on. In this case, CPU can be loaded up to 100% and still the guarantees hold true.

To perform RMA, the following information is required.

1. List of all the RTOS threads or tasks

2. Period of each thread
3. Worst Case Execution Time (WCET) of each thread
4. Maximum blocking time experienced by each thread

The threads are given priorities based on their period. The fastest thread gets the highest priority. The WCET for each thread can be obtained by running a thread several times and recording the worst case run time. Two threads of same priorities can exist but they need to have the same period.

Table 6.3 lists the threads in the PID force control application. The periods (T_i), worst case execution time (C_i), the utilization (U_i) and the priorities are also listed. The utilization can be calculated as shown in equation 6.4. The total utilization is the sum of the utilization of all the threads as given by equation 6.5.

$$U_i = \frac{C_i}{T_i} \tag{6.4}$$

$$U = \sum_{i=1}^{k} U_i \tag{6.5}$$

Thread Name	***Period***	***WCET***	U_i	***Priority***
$LoggerHwi$	10 ms	0.4 ms	4%	2
$LoggerSwi$	10 ms	.92 ms	9.2%	1
$MotorHwi$	0.05 ms	0.002 ms	4.11%	8
$LoadCellTrigger$	1 ms	0.03 ms	3%	6
$LoadCellADC$	1 ms	0.068 ms	6.8%	5
$LoadCellFilter$	1 ms	0.073 ms	7.3%	4
$ControllerTrigger$	0.5 ms	0.0206 ms	4.12%	7
$Controller$	1 ms	0.116 ms	11.6%	3

Table 6.3: Rate Monotonic Analysis of RTOS threads

Based on table 6.3 the total utilization of the application is 50.13%. There is still sufficient room to add more functionality such as CAN communication. A similar analysis can be performed for a different application to provide hard real time guarantees. The reported CPU load for this application is 38.9%. Thus, it is concluded that the scheduling priority obtained through RMA ensures that no task will miss its deadline.

6.2 Adaptive Force control

Adaptive control is a method of designing controllers that adapt to a plant which changes over time. An example would be to control an actuator whose performance changes over time due to change in the friction coefficient or the wearing of motors etc. An adaptive controller adapts itself to these changes by estimating the uncertainties using the measured signals from the system. The major benefit of using adaptive control is to obtain consistent system performance even in presence of uncertainties and variations in the plant.

There are several types of adaptive controller. The one studied in this work is the Model Reference Adaptive Controller (MRAC). Figure 6.10 shows the high level block diagram of a MRAC. The design of MRAC starts with defining a reference system which we want our actual plant to converge to. The actual output from the plant is compared to the desired reference model output and the error is used to update the parameters of the closed loop controller.

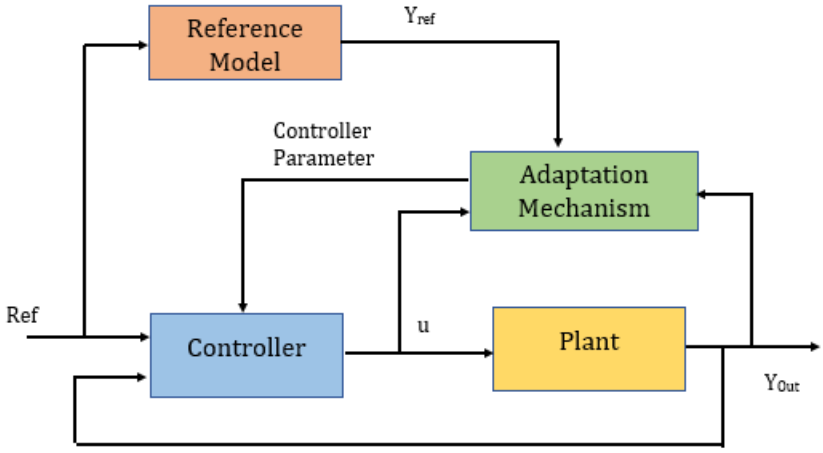


Figure 6.10: Model Reference Adaptive Control

The first step in MRAC is to define a reference system. The reference system is how we want our plant to behave eventually. In this case, a first order system is taken as reference, which is given by equation 6.6. Once a reference model is determined, the next step is to calculate error between the plant and the reference which is given by equation 6.7.

$$G_{ref} = \frac{x_r}{r} = \frac{b_m}{s + a_m} \tag{6.6}$$

$$e = x - x_r \tag{6.7}$$

Consider the controller output, which is the input to the actual plant is given by equation 6.8. Since the actual plant parameters are unknown, online estimation is performed to determine the parameters θ_x and θ_r. The goal of the controller to make the error go to zero. This can be achieved my the use of Lyapunov's direct method.

$$u = \theta_x x + \theta_r r \tag{6.8}$$

Lyapunov's direct method can be used to analyze the stability of a dynamical system. It uses the generalization of energy concept where a scalar energy like cost function is used in the stability analysis. The variation in energy of this scalar function along system trajectory gives insight into system behavior.

Lyapunov's method describes the use of a candidate lyapunov function $V(x)$ which is positive definite and has continuous partial derivatives. If the time derivative of this function along system trajectories $\dot{V}(x)$, is negative definite, then the system is globally stable.

This theorem can be extended for use in MRAC. A candidate Lyapunov function such as that given in equation 6.9 is chosen. The time derivative of the lyapunov function is given by equation 6.10.

$$V(x) = \frac{1}{2}e^2 + \frac{1}{2\gamma_x}|b|\tilde{\theta_x}^2 + \frac{1}{2\gamma_x|b|\tilde{\theta_r}^2} \tag{6.9}$$

$$\dot{V}(e, \tilde{\theta_x}, \tilde{\theta_r}) = a_r e^2 + e.|b|\tilde{\theta_x} x.sgn(b) + e.|b|\tilde{\theta_r} r.sgn(b) + \frac{1}{\gamma_x}|b|\tilde{\theta_x}\dot{\tilde{\theta}}_x + \frac{1}{\gamma_r}|b|\tilde{\theta_r}\dot{\tilde{\theta}}_r \tag{6.10}$$

By choosing the correct θ_x and θ_r, it can be ensured that $\dot{V}(x)$ is negative semidefinite as given by **??**, 6.12 and 6.13.

$$\dot{\theta}_x = -\gamma_x e.x.sgn(b) \tag{6.11}$$

$$\dot{\theta}_r = -\gamma_r e.r.sgn(b) \tag{6.12}$$

$$\dot{V}(e, \tilde{\theta_x}, \tilde{\theta_r}) = a_r e^2 \leq 0 \tag{6.13}$$

6.2.1 Simulation Results

Simulation is performed using the SEA model discussed in [10]. A First order reference system given by 6.6 is used.Here, $b_m = 1$ and $a_m = 5$.

Tuning of γ_x and γ_r is required to obtain satisfactory performance. Figure 6.11 shows the simulation output vs time. The reference system is given by blue curve. The adaptive controller drives the actual system toward the reference which can be seen from the red curve.

Figure 6.12 shows the error vs time plot. The error decreases and at about 4 second, it goes to zero. The performance can be further improved by tuning.

Figure 6.13 shows control action versus time plot. The control action starts from zero and then tries to maintain the output required for tracking. Finally, figure 6.14 shows the changes in the controller parameter over time which is part of the online estimation loop. In this experiment, the SEA model given in equation 6.2 is used.

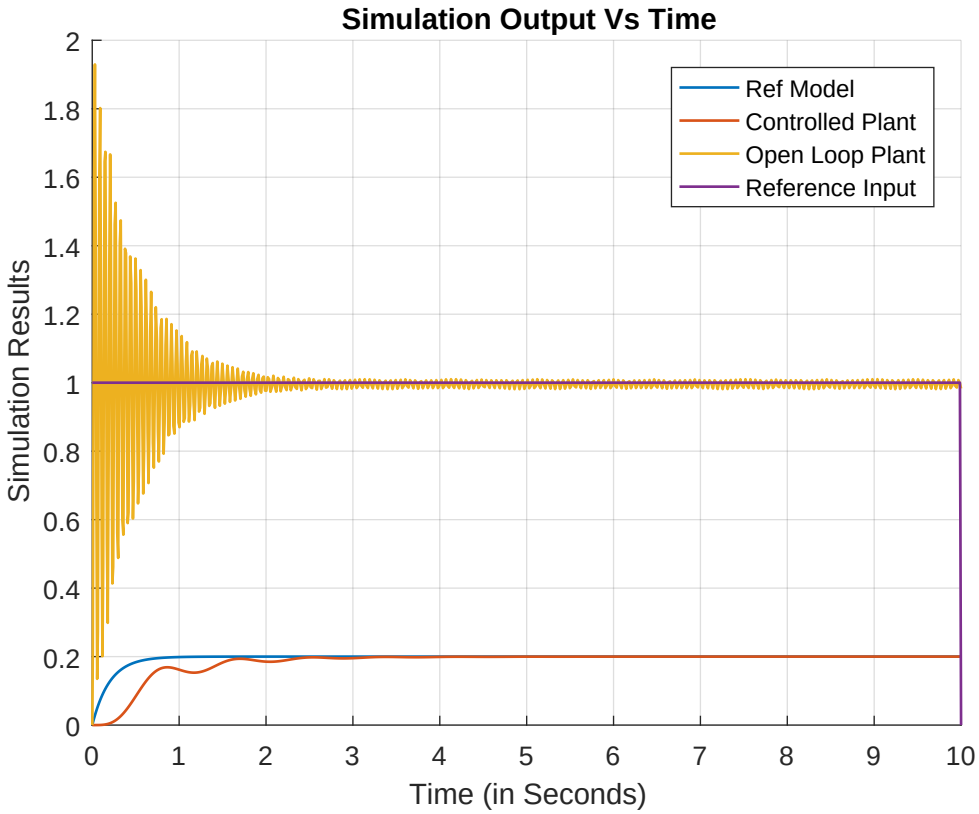


Figure 6.11: Simulation output vs Time

Figure 6.12: Error vs Time

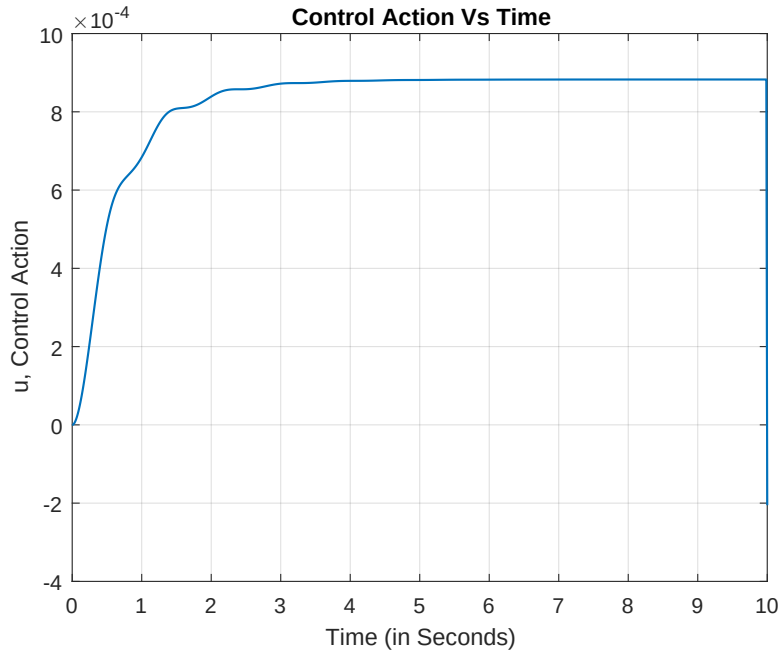


Figure 6.13: Control Action Vs Time

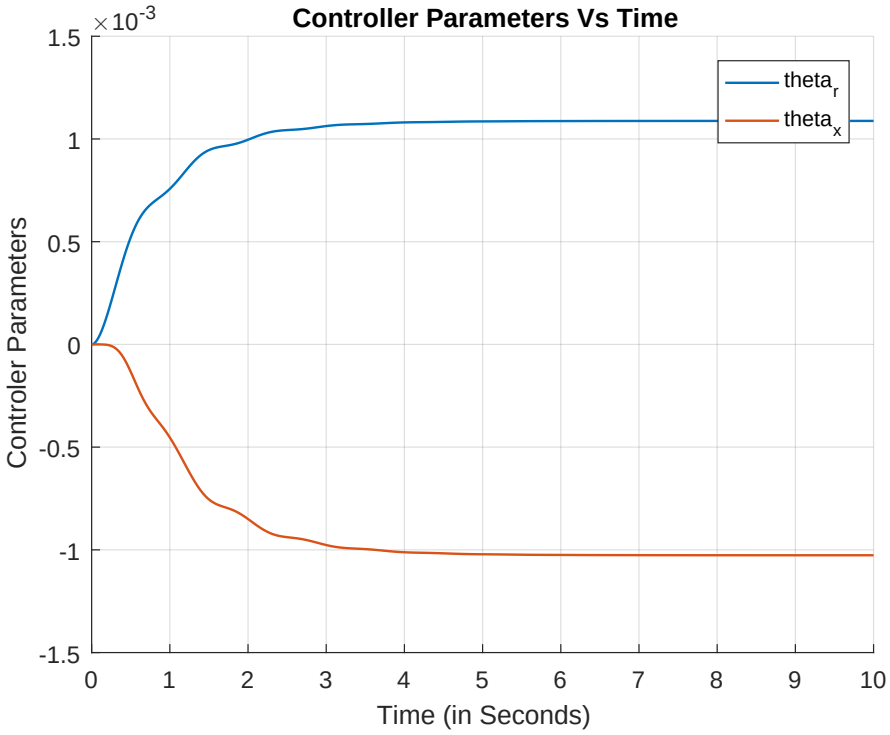


Figure 6.14: Evolution of controller parameters

6.2.2 Hardware Implementation

MRAC is used to control the knee pitch on THOR. The skeleton code for the application, which is shown below, is exactly the same as that for the PID application.

```
#include "slugTest.h"

double ref_input = 50; //Input to the reference model

int main(void){

    // Initialize clock at 80 MHZ frequency
    Clock_set_80MHz();

    // Initialize Console
    int BaudRate  = 115200;
    uint32_t loggerFreq = 1000; //1 KHz
    Logger_Init(loggerFreq, BaudRate);
```

```
    // Initialize motor
    uint32_t pwmFreq = 20000; // 20KHz
    Motor_Init(pwmFreq);

    // Initialize Load Cell
    int hardwareAveraging, ADCsampleFreq;
    hardwareAveraging = 8;
    ADCsampleFreq = 1000; //1 KHz
    LoadCell_init(hardwareAveraging, ADCsampleFreq);

    // Initialize controller
    setGoalForce(ref_input);
    uint32_t controllerFreq = 2000; //2kHZ
    Controller_Init(controllerFreq);
    ControllerEnable();

    // Enable all interrupts and channels
    EnableInterrupts();

    // For debugging
    RGBled_Init(0, 0, 1); //Blue

    while(1){
        // Do nothing
    }
}
```

The only required change in the interrupt handler for the controller which now calls the adaptive control function as shown below.

```
void ControllerIntHandler(void){
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    Adaptive_control();
}
```

The Adaptive control algorithm is implemented as follows:

```
void Adaptive_control(){
    //Plant output
    x = measuredLoad();

    //Error
    ERROR = x - x_ref_prev;

    //Theta update
    theta_x_dot = -gamma_x*ERROR*x;
    theta_r_dot = -gamma_r*ERROR*getGoalForce();

    theta_x_final = theta_x_dot*delta_t + theta_x_prev;
    theta_r_final = theta_r_dot*delta_t + theta_r_prev;

    theta_x_prev = theta_x_final;
    theta_r_prev = theta_r_final;

    // Controller output
    MRAC_OUT = theta_x_final*x + theta_r_final*getGoalForce();

    // Ref system output
    x_ref_dot = -a_ref*x_ref_prev + b_ref*getGoalForce();
    x_ref_final = x_ref_dot*delta_t + x_ref_prev;

    x_ref_prev = x_ref_final;

    // Send output
    checkLimits(MRAC_OUT);
}
```

Figure 6.15 shows the behavior of the controller when the reference system is very slow with $b_m = 0.1$ and $a_m = 0.1$. Figure 6.16 shows the results for a faster reference system with $b_m = 0.3$ and $a_m = 0.3$. The faster system allows for faster convergence which can be easy seen in the experimental video shown in [56].

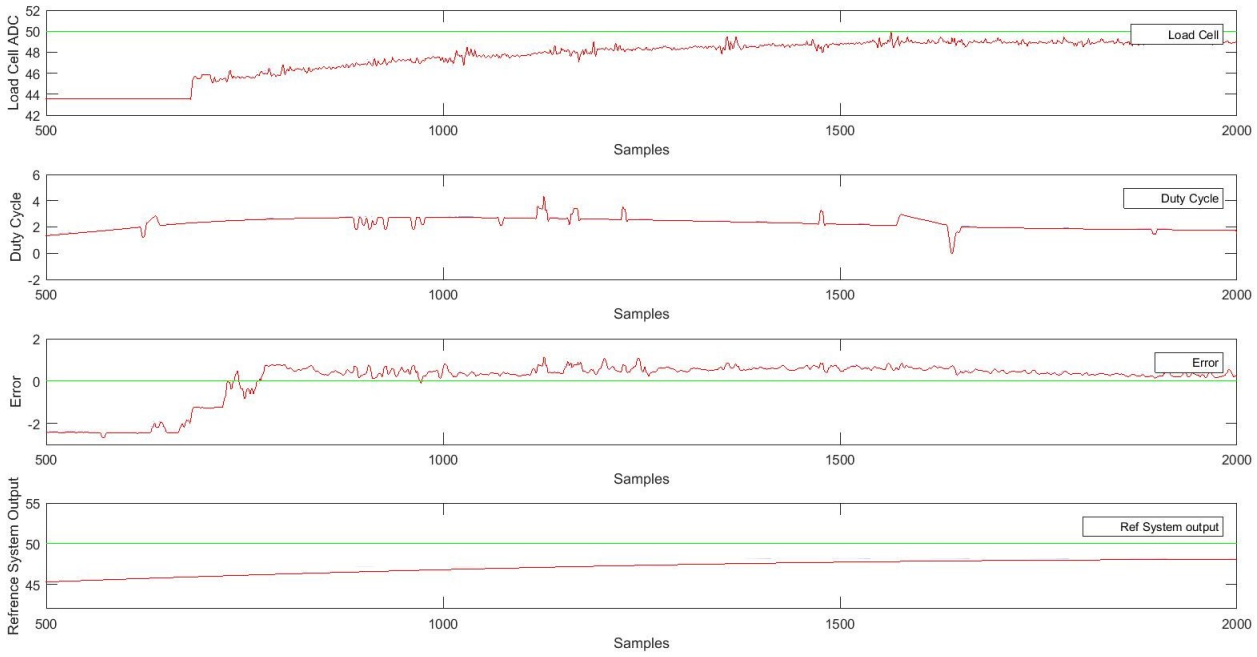


Figure 6.15: System performance with slow first order reference model

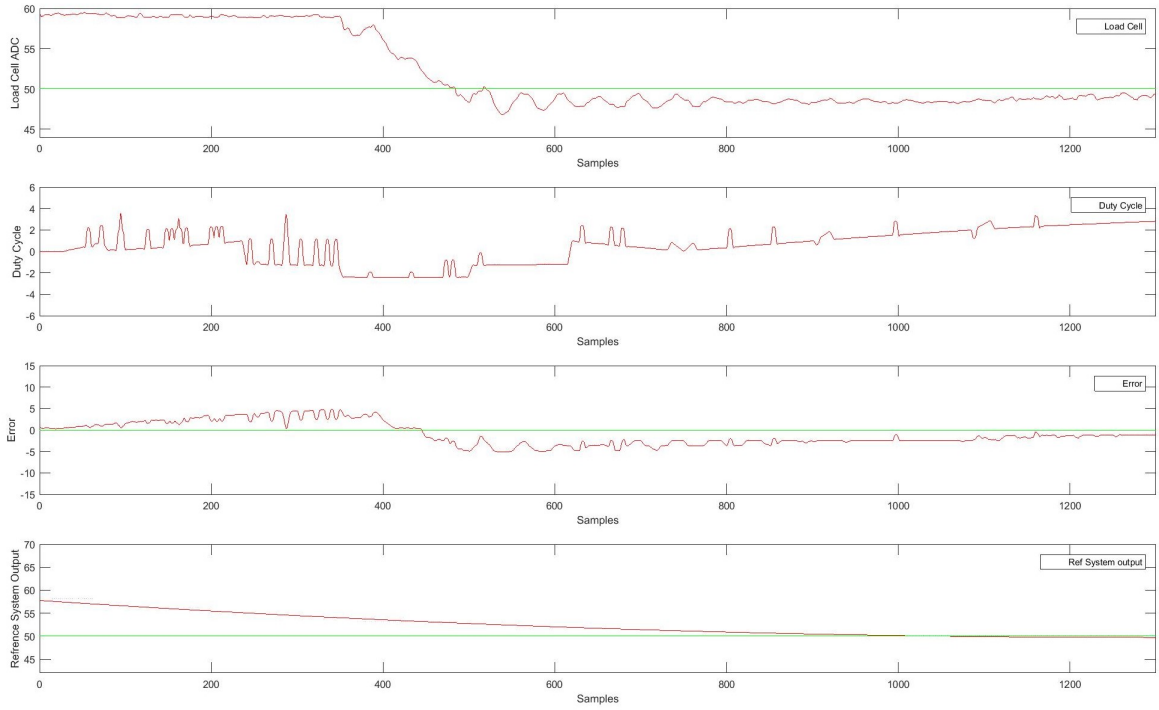


Figure 6.16: System performance with fast first order reference model

Chapter 7

Conclusion

The work presents the design and implementation of a custom motor controller architecture for control of Series Elastic Actuators. These motor controllers are designed specifically for the lower body exoskeleton. However, one of the design requirement was to make a multi-purpose board which can be used to replace the existing motor controllers on the humanoid Robots, THOR and ESCHER. These boards can also be used for quick prototyping of other exoskeleton projects such as prosthetic knee.

The motivation for this effort came from the exiting gap in the field of semi-custom electronic architectures which can make prototyping as well as development fast, easy and cheap. The use of existing off-the-shelf solutions for control of SEA was shown to be inadequate. The major requirements such as robustness, portability, small size and low cost posed severe restriction to the design of the motor controller. To solve these challenges, a simple and scalable design was proposed and implemented. The proposed design uses several necessary off-the-shelf components.

The software for the controller was written to be modular. The Board Support Package acts as an abstraction layer and makes programming control loops on the board significantly easier. The flexibility of the software allows it to be used in any application requiring data acquisition and data processing.

Sample code and test routines for several peripherals was developed to show the usage of the hardware and the software. The software design was followed by thorough testing of each and every peripheral. Two force control applications were discussed. These applications demonstrate the ease of use and the flexibility of the hardware and the software.

Finally, the capability to add an RTOS layer to any application was provided. The flexibility and the ability to log and monitor performance due to use of RTOS was discussed.

Test results were obtained for each and every sensor and peripheral. The performance of the boards were tested and verified using hardware testing techniques. Finally the complete

system was tested by deployment on the existing humanoid robot THOR.

7.1 Future Work

There is always scope of improvement on any robotics project and this work is no exception. While the boards already meet the current requirements, given below are some possible future changes on the TIVA SEA Shield.

1. Addition of hardware low pass filter circuit on output of the load cell amplifier. This would further help remove the noise.
2. Addition of a few more additional peripherals such as SPI and ADC for more choices on selection of sensors.
3. Addition of an on board SD card based logger.

The SD card based logger is an important improvement to the existing design. Often times, the serial logger cannot log the data in high speed due to which some debugging and data collection tasks become challenging. Having a on-board data logger hardware would certainly solve this problem.

The daughter board for the servo drivers can be upgraded by adding a connector to connect to the TIVA shield. A jumper switch can be used to select whether the board is connected directly as a stacked board or is connected externally to utilize the available space.

The software is modular and can keep growing. Some possible extensions would be to add optimal control routines such as LQR, software low pass filtering algorithms, different control architectures etc. An impedance controller has to be written on top of the existing force controller.

Bibliography

[1] Nicholas Paine et al. "Actuator Control for the NASA-JSC Valkyrie Humanoid Robot: A Decoupled Dynamics Approach for Torque Control of Series Elastic Robots". In: *Journal of Field Robotics* 32.3 (2015), pp. 378–396. ISSN: 1556-4967. DOI: 10.1002/rob.21556. URL: http://dx.doi.org/10.1002/rob.21556.

[2] J. Englsberger et al. "Overview of the torque-controlled humanoid robot TORO". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. Nov. 2014, pp. 916–923. DOI: 10.1109/HUMANOIDS.2014.7041473.

[3] N. G. Tsagarakis et al. "COMpliant huMANoid COMAN: Optimal joint stiffness tuning for modal frequency control". In: *2013 IEEE International Conference on Robotics and Automation*. May 2013, pp. 673–678. DOI: 10.1109/ICRA.2013.6630645.

[4] Coleman Knabe et al. "Team VALOR's ESCHER: A Novel Electromechanical Biped for the DARPA Robotics Challenge". In: *Journal of Field Robotics* (2017), n/a–n/a. ISSN: 1556-4967. DOI: 10.1002/rob.21697. URL: http://dx.doi.org/10.1002/rob.21697.

[5] Rewalk. *Rewalk Rehabilitation Exoskeletons*. 2016. URL: http://rewalk.com/.

[6] Magdo Bortole et al. "The H2 robotic exoskeleton for gait rehabilitation after stroke: early findings from a clinical study". In: *Journal of NeuroEngineering and Rehabilitation* 12.1 (2015), p. 54. ISSN: 1743-0003. DOI: 10.1186/s12984-015-0048-y. URL: http://dx.doi.org/10.1186/s12984-015-0048-y.

[7] A. J. Young and D. P. Ferris. "State of the Art and Future Directions for Lower Limb Robotic Exoskeletons". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25.2 (Feb. 2017), pp. 171–182. ISSN: 1534-4320. DOI: 10.1109/TNSRE.2016.2521160.

[8] G. Pratt and M. Williamson. "Series elastic actuators". In: 1 (1995), pp. 399–406.

[9] C. Knabe et al. "Design of a series elastic humanoid for the DARPA Robotics Challenge". In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. Nov. 2015, pp. 738–743. DOI: 10.1109/HUMANOIDS.2015.7363452.

[10] M. A. Hopkins et al. "Embedded joint-space control of a series elastic humanoid". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, pp. 3358–3365. DOI: 10.1109/IROS.2015.7353845.

[11] Kazuo Hirai et al. "The Development of Honda Humanoid Robot - Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on". In: 2004.

[12] J. Chestnutt et al. "Footstep Planning for the Honda ASIMO Humanoid". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. Apr. 2005, pp. 629–634. DOI: 10.1109/ROBOT.2005.1570188.

[13] S. Kajita et al. "Biped walking pattern generation by using preview control of zero-moment point". In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 2. Sept. 2003, 1620–1626 vol.2. DOI: 10.1109/ROBOT.2003.1241826.

[14] Khairul Anam and Adel Ali Al-Jumaily. "Active Exoskeleton Control Systems: State of the Art". In: *Procedia Engineering* 41 (2012). International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), pp. 988–994. ISSN: 1877-7058. DOI: http://dx.doi.org/10.1016/j.proeng.2012.07.273. URL: http://www.sciencedirect.com/science/article/pii/S1877705812026732.

[15] R. V. Ham et al. "Compliant actuator designs". In: *IEEE Robotics Automation Magazine* 16.3 (Sept. 2009), pp. 81–94. ISSN: 1070-9932. DOI: 10.1109/MRA.2009.933629.

[16] Boston Dynamics. *Atlas*. URL: https://www.bostondynamics.com/atlas.

[17] D. C. Bentivegna, C. G. Atkeson, and Jung-Yup Kim. "Compliant control of a hydraulic humanoid joint". In: *2007 7th IEEE-RAS International Conference on Humanoid Robots*. Nov. 2007, pp. 483–489. DOI: 10.1109/ICHR.2007.4813914.

[18] G. A. Pratt et al. "Late motor processing in low-impedance robots: impedance control of series-elastic actuators". In: *Proceedings of the 2004 American Control Conference*. Vol. 4. June 2004, 3245–3251 vol.4.

[19] Stephen A. Ressler. "Design and Implementation of a dual axis motor controller for parallel and serial series elastic actuators". An optional note. MA thesis. Blacksburg, Virginia: Virginia tech, Mar. 2014.

[20] Derek Lahr, Viktor Orekhov, and Bryce Lee. "Early Developments of a Parallelly Actuated Humanoid, Saffir". In: 2013.

[21] Robert J. Griffin et al. "Design and Approach of Team IHMC in the 2016 Cybathlon". In: *CoRR* abs/1702.08656 (2017). URL: http://arxiv.org/abs/1702.08656.

[22] Elmo. *Twitter Gold*. URL: http://www.elmomc.com/products/gold-twitter-servo-drive.htm.

[23] P. D. Neuhaus et al. "Design and evaluation of Mina: A robotic orthosis for paraplegics". In: *2011 IEEE International Conference on Rehabilitation Robotics*. June 2011, pp. 1–8. DOI: 10.1109/ICORR.2011.5975468.

[24] C. Beck et al. R. Rea. "X1: A Robotic Exoskeleton for In-Space Countermeasures and Dynamometry". In: *AIAA Space Conference and Exposition* (2013).

[25] E. J. Rouse et al. "Clutchable series-elastic actuator: Design of a robotic knee prosthesis for minimum energy consumption". In: (June 2013), pp. 1–6. ISSN: 1945-7898. DOI: 10.1109/ICORR.2013.6650383.

[26] E. D. Ledoux and M. Goldfarb. "Control and Evaluation of a Powered Transfemoral Prosthesis for Stair Ascent". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* PP.99 (2017), pp. 1–1. ISSN: 1534-4320. DOI: 10.1109/TNSRE.2017.2656467.

[27] Joshua M. Caputo and Steven H. Collins. "An experimental robotic testbed for accelerated development of ankle prostheses". In: *2013 IEEE International Conference on Robotics and Automation* (2013), pp. 2645–2650.

[28] F. Sup et al. "Design and control of an active electrical knee and ankle prosthesis". In: *2008 2nd IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics*. Oct. 2008, pp. 523–528. DOI: 10.1109/BIOROB.2008.4762811.

[29] J. F. Duval and H. M. Herr. "FlexSEA-Execute: Advanced motion controller for wearable robotic applications". In: *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. June 2016, pp. 1056–1061. DOI: 10.1109/BIOROB.2016.7523771.

[30] Institute of Human and Machine Cognition. *Ubuntu RTOS*. URL: https://github.com/ihmcrobotics/ihmc-realtime.

[31] Scott Kuindersma et al. "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot". In: *Autonomous Robots* 40.3 (2016), pp. 429–455. ISSN: 1573-7527. DOI: 10.1007/s10514-015-9479-3. URL: http://dx.doi.org/10.1007/s10514-015-9479-3.

[32] Peter Neuhaus. *Cyblathon*. 2016. URL: http://robots.ihmc.us/cybathlon/.

[33] Beagle Board. *beagleBoneBlack*. URL: https://beagleboard.org/black.

[34] Texas Instruments. *TM4C123GH6PM*. URL: http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf.

[35] Texas Instruments. *Launchpad Manual*. URL: http://www.ti.com/lit/ug/spmu296/spmu296.pdf.

[36] Futek. *Load Cell*. URL: https://www.futek.com/files/pdf/Product%20Drawings/lcm200.pdf.

[37] Texas Instruments. *INA125*. URL: http://www.ti.com/lit/ds/symlink/ina125.pdf.

[38] Intersil. *Low Pass Filter*. URL: http://www.intersil.com/content/dam/Intersil/documents/an96/an9603.pdf.

[39] EDN. *Low Pass Filter*. URL: http://www.edn.com/design/systems-design/4320010/A-simple-software-lowpass-filter-suits-embedded-system-applications.

[40] Adafruit. *K-Type Thermocouple*. URL: https://goo.gl/QC44Lj.

[41] Adafruit. *Thermocouple Amplifier*. URL: https://www.adafruit.com/product/1778.

[42] Gurley Precision. *Absolute Encoder*. URL: http://www.gurley.com/Encoders/PDF/A19.pdf.

[43] Exar Corporation. *SP490*. URL: https://www.exar.com/content/document.ashx?id=1337.

[44] RLS. *Absolute Encoder*. URL: https://www.rls.si/orbis-true-absolute-rotary-encoder.

[45] RLS. *Absolute Encoder*. URL: https://www.rls.si/aksim-rotary-absolute-encoder-module.

[46] Maxon. *Incremental Encoder*. URL: http://storkdrives.com/wp-content/uploads/2013/10/maxon-4-Pole-catalog-data1.pdf.

[47] Texas Instruments. *Line Driver*. URL: http://www.ti.com/lit/ds/symlink/sn75175.pdf.

[48] Advanced Motion Controls. *Servo Driver*. URL: https://www.servo2go.com/support/downloads/Advanced_Motion_Controls_azbdc60a8.pdf.

[49] Bosch Semiconductors. *CAN Bus*. URL: http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf.

[50] Microchip. *CAN transceiver*. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf.

[51] Texas Instruments. *RTOS for MCU*. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf.

[52] Texas instruments. *RTOS User Manual*. URL: http://www.ti.com/lit/ug/spruhd4m/spruhd4m.pdf.

[53] Shriya Shah. *PID Application demonstration*. URL: https://youtu.be/r0okXl7lLqg.

[54] Shriya Shah. *Need for an RTOS*. URL: https://youtu.be/kw-apyquG0E.

[55] J. Lehoczky, L. Sha, and Y. Ding. "The rate monotonic scheduling algorithm: exact characterization and average case behavior". In: *[1989] Proceedings. Real-Time Systems Symposium*. Dec. 1989, pp. 166–171. DOI: 10.1109/REAL.1989.63567.

[56] Shriya Shah. *MRAC Application demonstration*. URL: https://youtu.be/mzJ0H9qPpB0.

Appendices

Appendix A

New RTOS Project in Code Composer Studio

This section describes the step by step procedure of creating a TI-RTOS project. The instructions assume that TI SYS/BIOS is installed.

1. Select a new Ti-RTOS minimal project from the new project selection window sh shown in figure A.1.

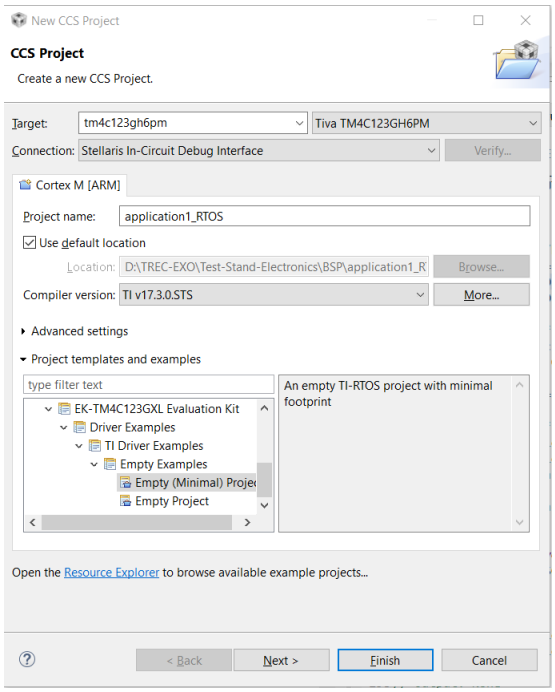


Figure A.1: Select a new project

2. Select the latest XDC tool version and the compiler as shown in figure A.3.

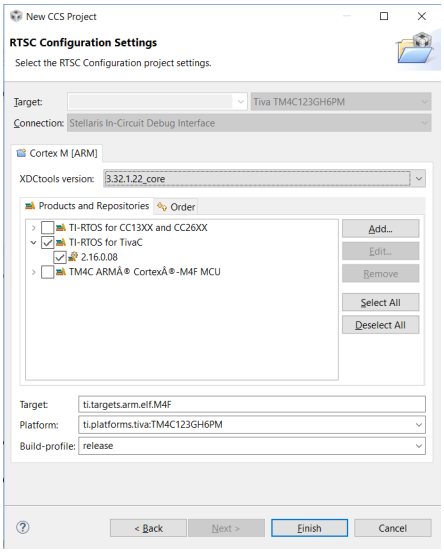


Figure A.2: Select the lasted XDC tool and compiler

3. Open the $empty_main.c$ file and run the green debug and run button.

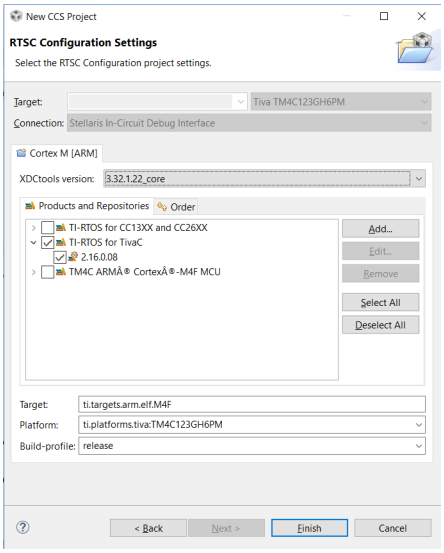


Figure A.3: Run the main program

Once the above steps are followed and reset button is pressed on the launchpad, the blue led starts blinking as a rate of one second. The project consists of a real time task that executes the toggle action.

The *.cfg* file in the project is the main configuration file where the real time threads, logging facilities, RTOS configuration settings are present.

Replace the content of the *emptymain.c* file with the program shown below. Also, add the board support package files *slug.c* and *slug.h* in the project.

The goal is to configure a timer interrupt and use it to toggle a LED. This can be done by using hardware and software interrupt. Another method is to use Tasks. However, in this work the Hardware and Software interrupt method is used.

A hardware timer can generate an interrupt based on the period specified. Time2 initialization routine from the BSP initializes the timer 2. The timer 2 is then added as a HWI using the configuration file as shown in figure A.4.

```
#include <xdc/std.h>
#include <ti/sysbios/BIOS.h>
#include <xdc/runtime/Log.h>
#include <xdc/cfg/global.h>

#include <stdint.h>
#include <stdbool.h>

#include "slug.h"

void main(void)
{
    // Setup system clock to 80 MHZ
   Clock_set_80MHz();

   // Initialize the LED
   RGBled_Init(0, 0, 1); // Blue

   // Initialize the timer 2
   uint32_t ui32Period;
   ui32Period = (SysCtlClockGet() /2);
   initTimer2(ui32Period);

   // Start BIOS
   BIOS_start();
}
```

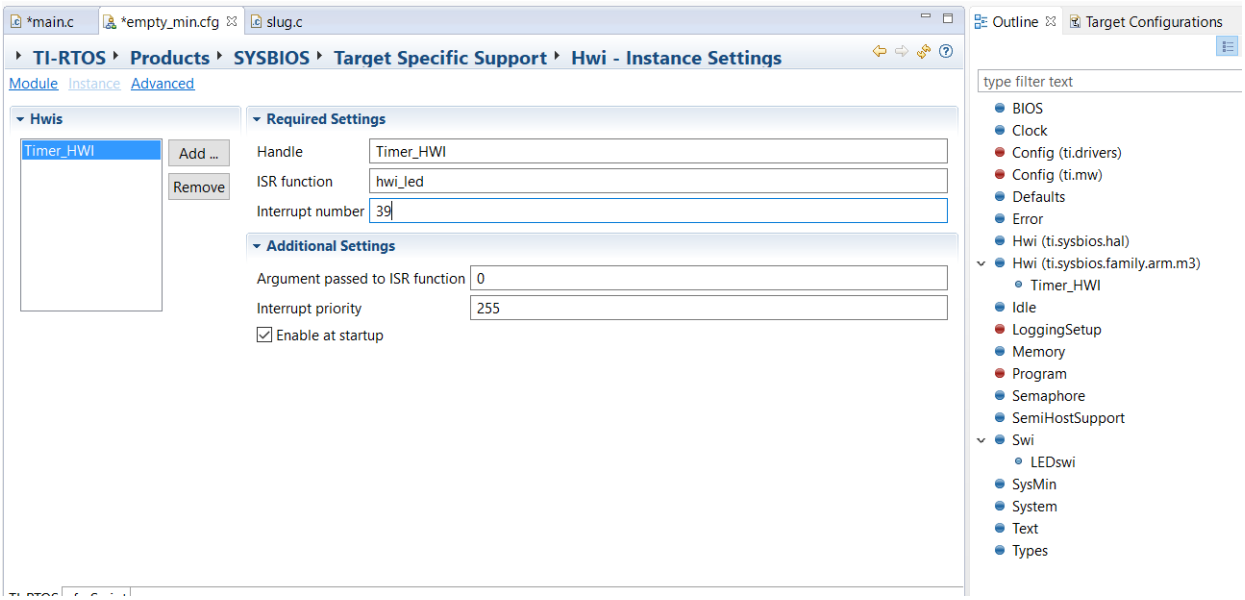


Figure A.4: Add a Hardware interrupt

The interrupt number for timer 2 is 39 which can be obtained from the TM4C123GH6PM data sheet [34]. The Handle is the name of the HWI and the ISR function is the function that is called when the HWI is executed. Here, the *hwi_led* function is being called which is shown below.

```
void hwi_led(void){
    // clear the interrupt flag
    Timer2IntHandler();

    // Post a software interrupt
    Swi_post(LEDswi);
}
```

The function calls the timer interrupt handler which is implemented in the BSP. This call clears the interrupt flag. The HWI then posts a software interrupt with name LEDswi. Figure A.5 shows the method of adding a SWI to the application.

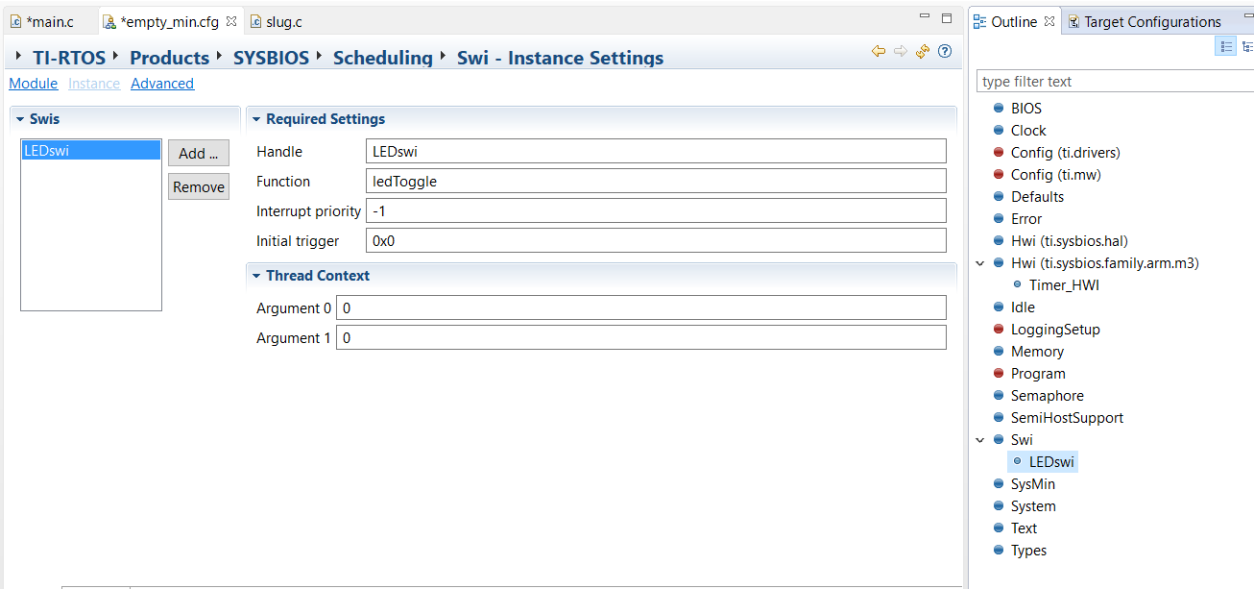


Figure A.5: Add a Software interrupt

The SWI handle is the name of the SWI which is used when the SWI is called inside the HWI. The function specifies the function which gets executed when the SWI is called. As the requirement here is to blink the LED, the following function is called. $RGBled_Toggle()$ is from the BSP. The software interrupt priority is 1 which s the lowest for this application.

```
void ledToggle(void)
{
    RGBled_Toggle(0, 0,1); //Toggle blue LED

}
```

An number of hardware or software interrupt can be added by following the described method.

Appendix B

TIVA SEA shield

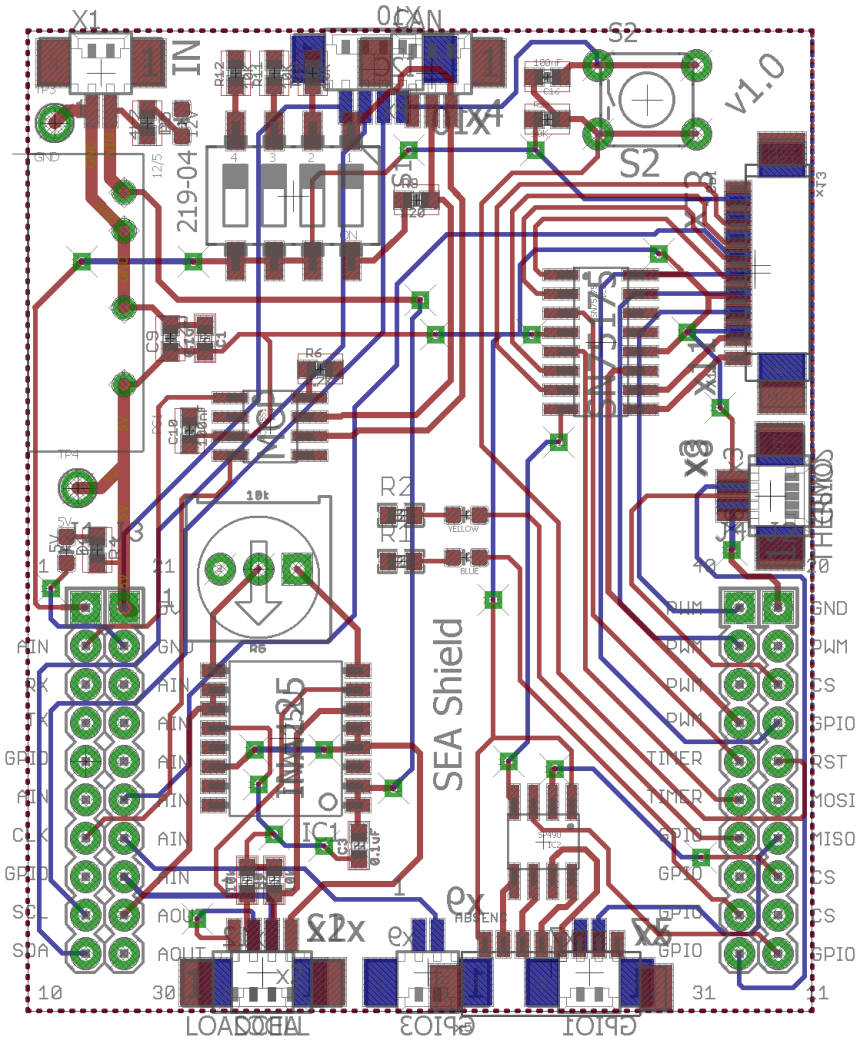
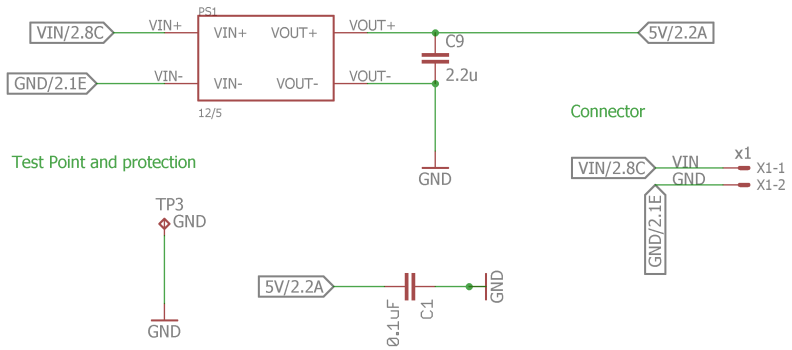


Figure B.1: Tiva SEA Shield

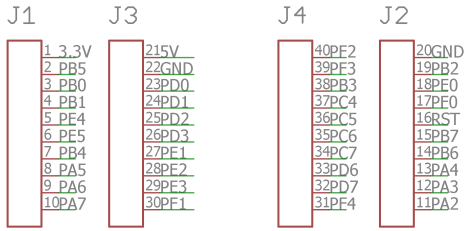
Voltage Conversion



Test Point and protection

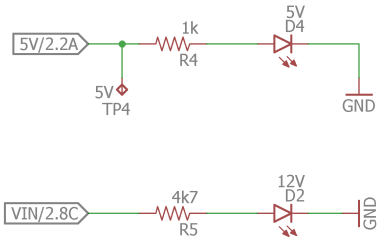
Connector

Tiva Connectors

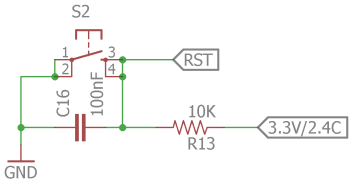


Indication LED

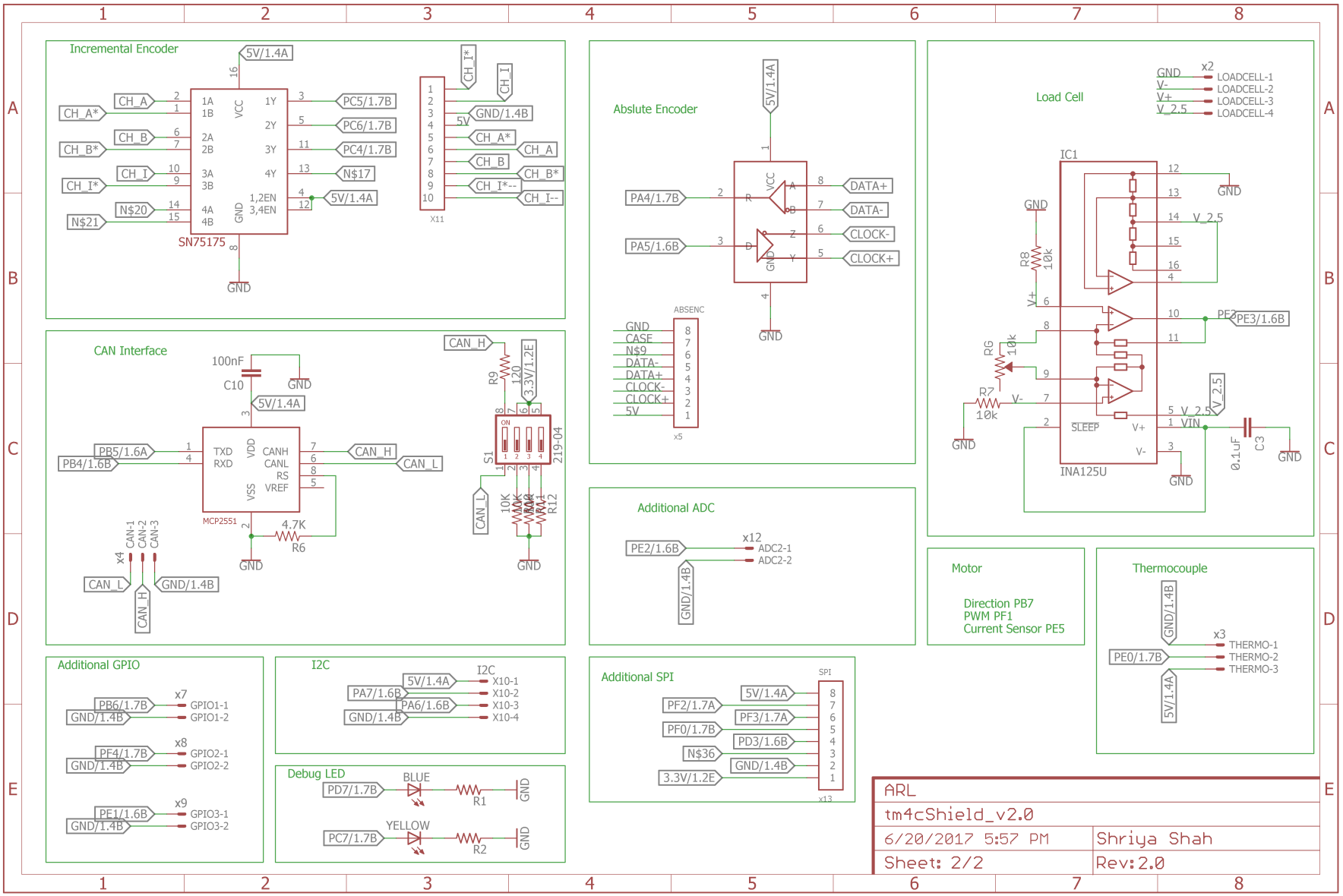
Red 12V
Green 5V



Reset



ARL	
tm4cShield_v2.0	
6/20/2017 5:35 PM	Shriya Shah
Sheet: 1/2	Rev:2.0


Incremental Encoder
5V/1.4A
CH_A
CH_A*
CH_B
CH_B*
CH_I
CH_I*
N$20
N$21
SN75175
PC5/1.7B
PC6/1.7B
PC4/1.7B
N$17
5V/1.4A
GND
CH_I*
CH_I
GND/1.4B
CH_A*
CH_A
CH_B
CH_B*
CH_I*--
CH_I--
X11
Absolute Encoder
PA4/1.7B
PA5/1.6B
DATA+
DATA-
CLOCK-
CLOCK+
ABSENC
GND
CASE
N$9
DATA-
DATA+
CLOCK-
CLOCK+
5V
x5
Load Cell
x2
LOADCELL-1
LOADCELL-2
LOADCELL-3
LOADCELL-4
V-
V+
V_2.5
IC1
INA125U
R8
10k
R6
10k
R7
10k
PE2
PE3/1.6B
SLEEP
VIN
0.1uF
C3
CAN Interface
100nF
C10
PB5/1.6A
PB4/1.6B
CAN_H
CAN_L
MCP2551
4.7K
R6
R9
120
3.3V/1.2E
S1
219-04
10K
R12
x4
CAN-1
CAN-2
CAN-3
GND/1.4B
Additional ADC
PE2/1.6B
x12
ADC2-1
ADC2-2
GND/1.4B
Motor
Direction PB7
PWM PF1
Current Sensor PE5
Thermocouple
GND/1.4B
PE0/1.7B
x3
THERMO-1
THERMO-2
THERMO-3
5V/1.4A
Additional GPIO
PB6/1.7B
GND/1.4B
x7
GPIO1-1
GPIO1-2
PF4/1.7B
GND/1.4B
x8
GPIO2-1
GPIO2-2
PE1/1.6B
GND/1.4B
x9
GPIO3-1
GPIO3-2
I2C
5V/1.4A
PA7/1.6B
PA6/1.6B
GND/1.4B
I2C
X10-1
X10-2
X10-3
X10-4
Debug LED
PD7/1.7B
BLUE
R1
GND
PC7/1.7B
YELLOW
R2
GND
Additional SPI
PF2/1.7A
PF0/1.7B
N$36
3.3V/1.2E
5V/1.4A
PF3/1.7A
PD3/1.6B
GND/1.4B
SPI
x13
ARL
tm4cShield_v2.0
6/20/2017 5:57 PM
Shriya Shah
Sheet: 2/2
Rev:2.0

Appendix C

Power Distribution Board

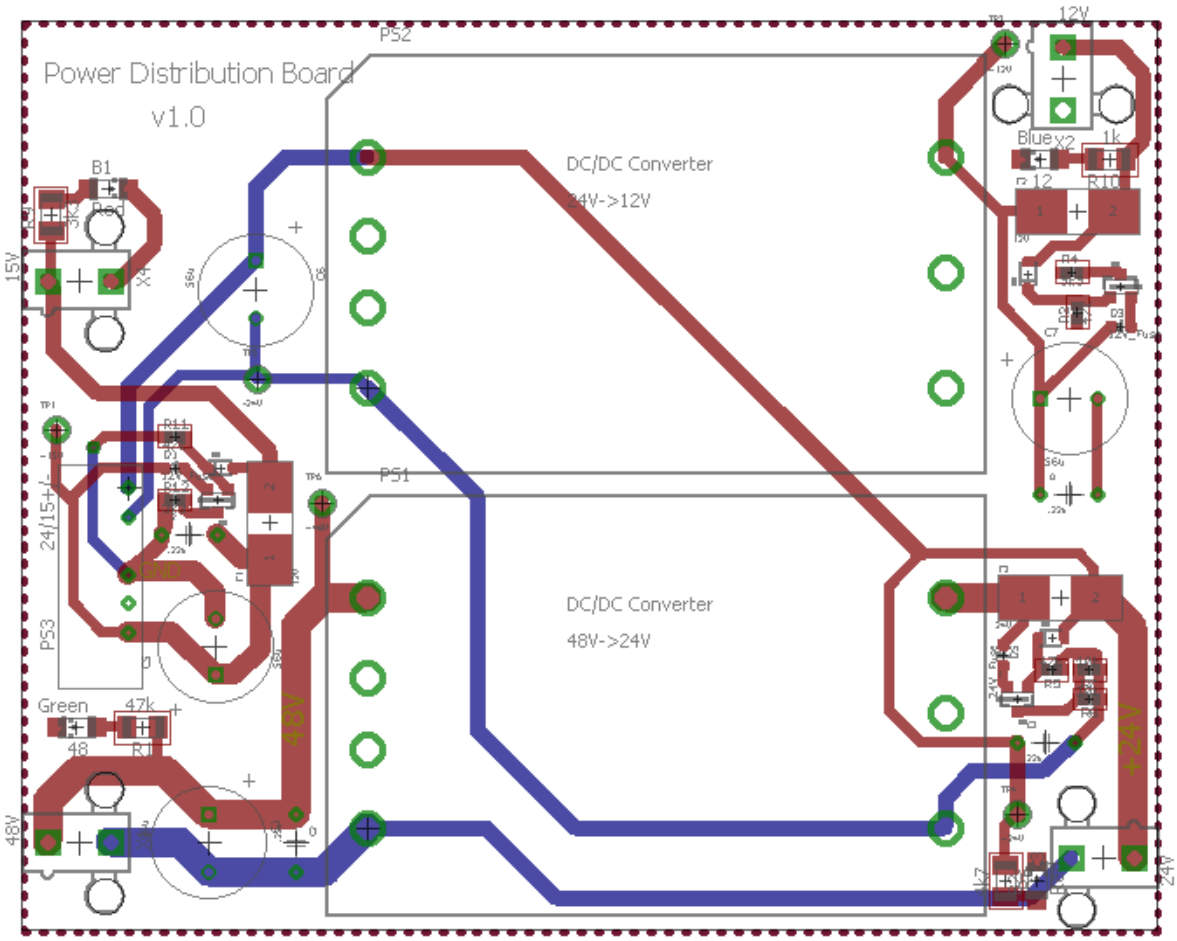
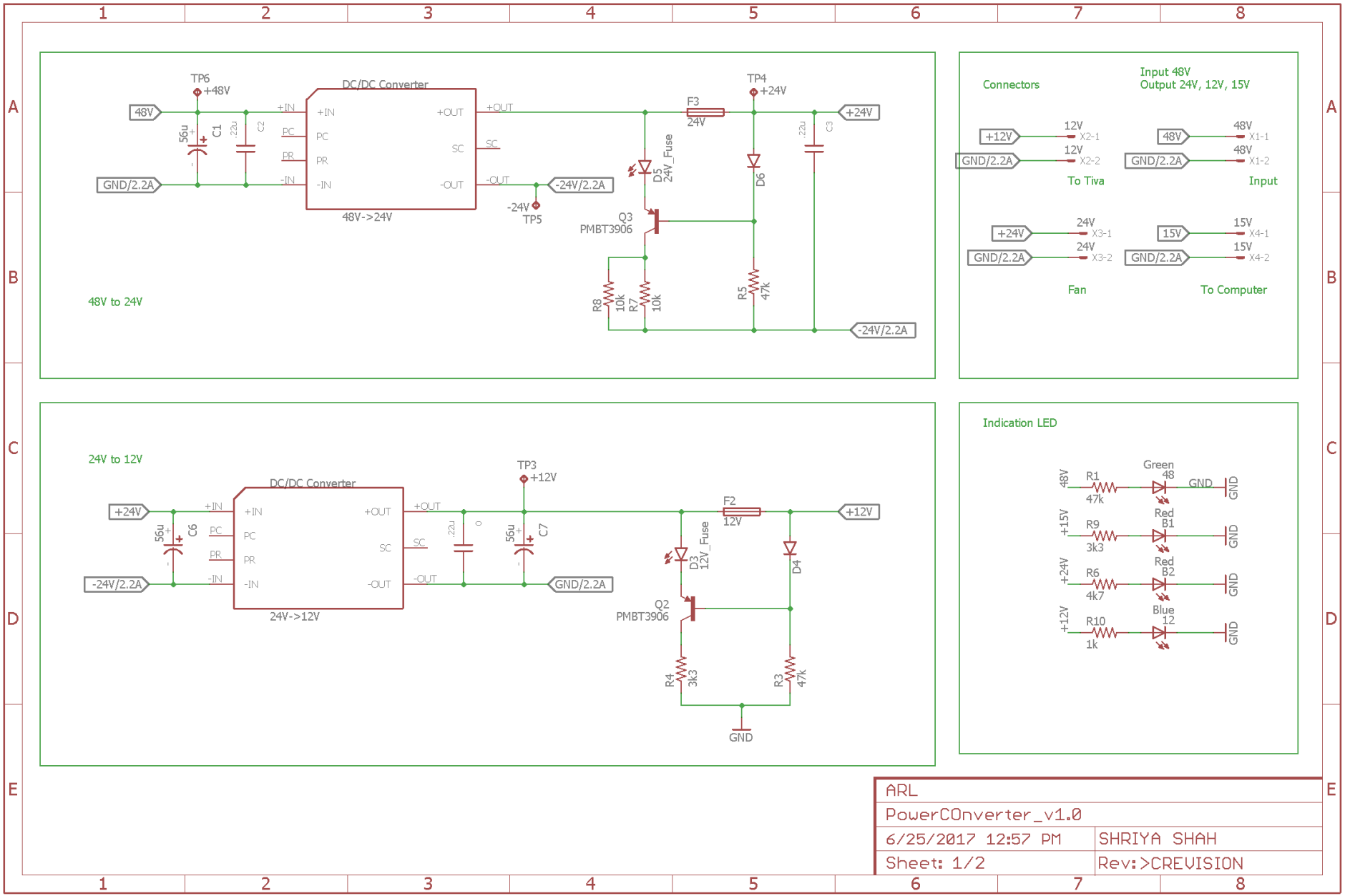
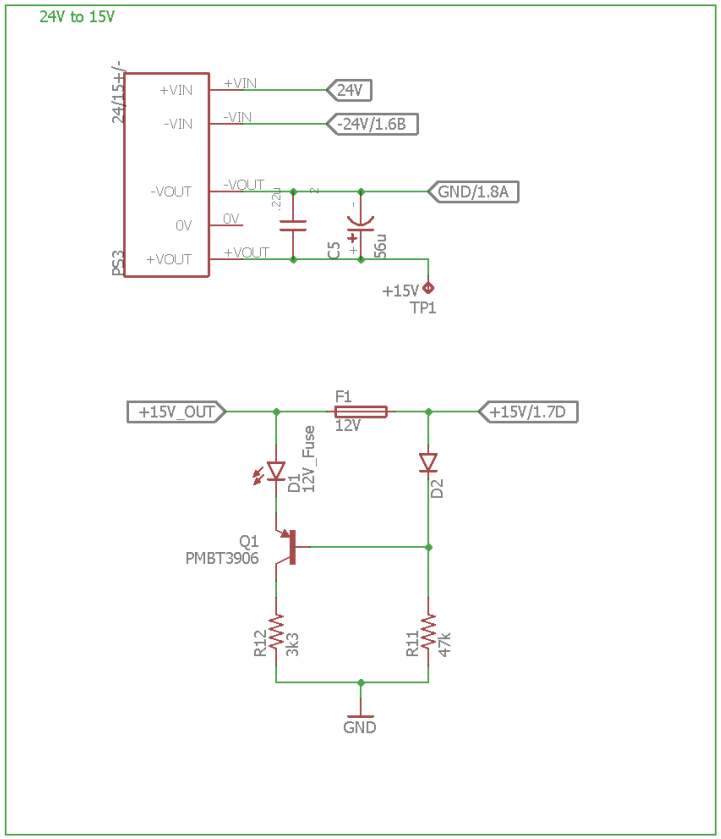


Figure C.1: Power Distribution Board



ARL	
PowerCOnverter_v1.0	
6/25/2017 12:57 PM	SHRIYA SHAH
Sheet: 1/2	Rev:>CREVISION



ARL	
PowerCOnverter_v1.0	
6/25/2017 12:57 PM	SHRIYA SHAH
Sheet: 2/2	Rev:>CREVISION

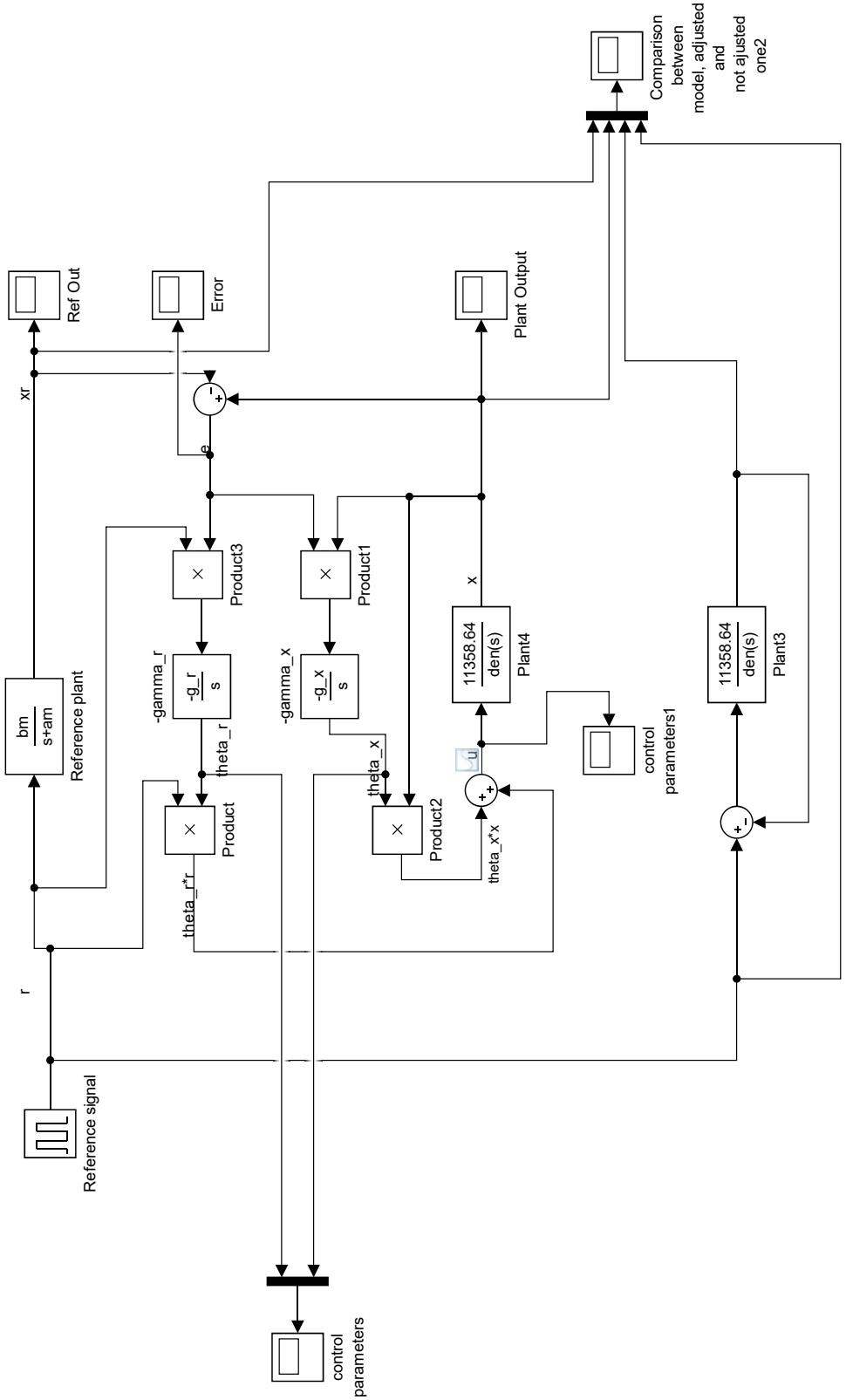
Appendix D

Adaptive Controller Design

Given below is the MatLab script used for the simulation. The next page shows the Simulink block diagram.

```
%% Design of Model Reference Adaptive controller for the SEA on THOR
clear all; close all; clc

%% 1st order ref

% Ref plant
am = 5;
bm = 1;

% gamma
g_x = -0.1;
g_r = 0.01;

%% Simulate the model and plot results

sim('Sea_firstOrder_simple_drL');
time = ContParam(:, 1);

% Control Action
h = figure(1);
plot(time, ScopeData1(:, 2), 'LineWidth', 1);
grid on
xlabel('Time (in Seconds)')
ylabel('u, Control Action')
title('Control Action Vs Time')
print(h,'-dpng','-r1000','Control_Action')
```

```
% Error
h = figure(2);
plot(time, u2(:, 2), 'LineWidth', 1);
grid on
xlabel('Time (in Seconds)')
ylabel('e = x - x_r')
title('Error between reference model and actual plant')
print(h,'-dpng','-r1000','Error')

% System Output
h = figure(3);
hold all
plot(time, output1(:, 2), 'LineWidth', 1);
plot(time, output1(:, 3), 'LineWidth', 1);
plot(time, output1(:, 4), 'LineWidth', 1);
plot(time, output1(:, 5), 'LineWidth', 1);
grid on
xlabel('Time (in Seconds)')
ylabel('Simulation Results')
title('Simulation Output Vs Time')
legend('Ref Model', 'Controlled Plant', 'Open Loop Plant', 'Reference Input')
print(h,'-dpng','-r1000','Output')

% Controller Parameters
h = figure(4);
hold all
plot(time, ContParam(:, 2), 'LineWidth', 1);
plot(time, ContParam(:, 3), 'LineWidth', 1);
grid on
xlabel('Time (in Seconds)')
ylabel('Controler Parameters')
title('Controller Parameters Vs Time')
legend('theta_r', 'theta_x')
print(h,'-dpng','-r1000','Control_Parameters')
```


Comparison between model, adjusted and not ajusted one2
Ref Out
Error
Plant Output
xr
e
Product3
Product1
x
-gamma_r
-gamma_x
11358.64 / den(s)
Plant4
bm / s+am
Reference plant
-g_r / s
-g_x / s
theta_r
theta_x
u
control parameters1
11358.64 / den(s)
Plant3
Product
Product2
theta_r*r
theta_x*x
r
Reference signal
control parameters

Appendix E

PID Controller Design

Given below is the simple MatLab simulation used to design PID controller for the given SEA.

```
%% Design of Force controller for SEA
clear all; close all; clc

%% Define Plant
s = tf('s');
SEA = 11358.64/(s^2 + 3.823*s + 50.126);

%Pole Zero map
figure
pzmap(SEA);
grid on

% Bode
figure
bode(SEA);
grid on


%% Open loop Step response
u1 = 1; % pound
u2 = 2;
u3 = 3;
u4 = 4;
u5 = 5;

[out1, T] = step(u1*SEA*(100/225));
```

```
[out2, T] = step(u2*SEA*(100/225));
[out3, T] = step(u3*SEA*(100/225));
[out4, T] = step(u4*SEA*(100/225));
[out5, T] = step(u5*SEA*(100/225));

figure;
plot(T, out1);
hold on
grid on
plot(T, out2);
plot(T, out3);
plot(T, out4);
plot(T, out5);
xlabel('Time in second');
ylabel('Force in N');
legend('100 N ', '200 N', '300 N', '400 N', '500 N');
title('Open Loop Step Resnponse');
axis([0 1 0 800]);

%% PID controller
Kp = .01;
Ki = 0.05;
Kd = 0.001;

C = Kp + Ki/s + Kd*s;

SEA_cl = feedback(C*SEA, 1);

figure
bode(SEA_cl);
grid on

t = 0:0.01:2;
[out_cl1, T2] = step(t, u1*SEA_cl*100);
[out_cl2, T2] = step(t, u2*SEA_cl*100);
[out_cl3, T2] = step(t, u3*SEA_cl*100);
[out_cl4, T2] = step(t, u4*SEA_cl*100);
[out_cl5, T2] = step(t, u5*SEA_cl*100);

figure;
plot(T2, out_cl1);
hold on
```

```
grid on
plot(T2, out_cl2);
plot(T2, out_cl3);
plot(T2, out_cl4);
plot(T2, out_cl5);
xlabel('Time in second');
ylabel('Force in N');
legend('100 N ', '200 N', '300 N', '400 N', '500 N');
title('Closed Loop Step Resnponse');
axis([0 1 0 600]);
```