Capstone: Hyperloop Control and Telemetry Architecture

Lab Members:

Rigoberto Orozco (1228362) _____

Anthony Grigore (1229767) _____

Brock Sorensen (1369027) _____

Josiah Lim (1127824) _____

Table of Contents

1 – ABSTRACT

This report documents the development of the Hyperloop Control and Telemetry Architecture that can control a Hyperloop prototype pod and report various telemetric measurements. The primary microcontroller used in this architecture is the PIC18F4685 which features ECAN module, 10-bit ADC, UART, and multiple I/O ports. This system can control the speed at which the pod is propelled, brake the pod during testing, run motors to spin Halbach arrays, and simultaneously communicate with all the subsystems through the CAN bus. The speed of this control system is very responsive and accurately reports telemetry data to a main hub which will soon implement ethernet capability to communicate to the main hub wirelessly in revisions to come.

2 – INTRODUCTION

The purpose of this capstone is to build a functioning prototype of the Hyperloop Control and Telemetry Architecture that can be translated into a pod used for real-world testing. The requirements of the system are the ability to monitor acceleration, roll, pitch, pressure, altitude, and temperature of a theoretical pod for real-time data acquisition. Furthermore, it uses a CAN bus for the master node to communicate to multiple slave nodes and up to eight bytes of data can be sent through the CAN bus at one time. Overall, the system will consist of a master node connected to four subsystems: motor, levitation, braking, and independent sensors. All measurements will be displayed in a terminal window on a local PC. For this system, the local PC used is a Raspberry Pi. Commands are sent through UART from the Raspberry Pi to the master node, which will output signals on the CAN bus.

3 – LAB DISCUSSION

3.1 - Design Specification

System Description

This specification describes and defines the detailed design requirements for the Hyperloop Control and Telemetry Architecture. This system will support measurements of acceleration, roll, pitch, pressure, altitude, and temperature. The system will utilize the I2C protocol or ADC to obtain measurements from sensors, and use the CAN bus protocol to communicate between the master node and the slave node subsystems.The Hyperloop Pod control architecture is to be able to control the Hyperloop pod propulsion mechanism (motors, actuators, and brakes) and provide telemetry readings of acceleration, roll, pitch, pressure, altitude, and temperature of various locations throughout the pod. Transmission of data will simulate communication over a local area network. At full scale, the control architecture is to be controlled remotely via a Network Access Panel (NAP) that is connected to a wireless LAN inside the length of the test-track tube. The user will send commands and retrieve real-time data through a terminal (for this scope of the capstone), but eventually control and display data on a GUI. The control architecture is to implement redundant safety mechanisms in case of an unexpected failure of a subsystem. There are four subsystems: Motors control, Actuator control, Braking control, and Independent Sensor monitoring. The Motor control subsystem is to variably control the speed of the motors as well as to provide real-time temperature readings of motors. The Actuator control subsystem is to control the position of the Halbach arrays. The Braking control is to control the braking actuators via the CAN control bus or through ADC readings of ultrasonic distance sensors (i.e. autonomous braking). The Independent Sensor system is to monitor the analog and digital readings of various sensors on board. Data will be acquired through ADC as well as I^2C serial communication. For the scope of the capstone project we will use sensors to acquire data and connect actuators and simple motors to the control architecture. This will allow the team to simulate operation of the pod as closely as possible without actually having a pod in the lab. The actual braking and propulsion mechanisms will not be tested for the capstone project, only a control system will be designed and tested.

Specification of External Environment

The control architecture is to operate in a basic, industrial environment. This environment must have commercial grade temperature and pressure as well as a normal lighting environment. The unit will support battery-powered operation via a 12 VDC power line. All system components will be powered via a 5 VDC reference step-down internal to the central hub of communication and control. Specific details are included in Section 2.6: Operating Specification. The system shall also be able to operate in an environment with low atmospheric pressure in a Hyperloop tube.

System Input and Output Specification

System Inputs

The system shall measure the following signals directly from independent sensors. These signals will be sent periodically through I2C or obtained through ADC in the PIC18F4685:

Motor Subsystem:

- Temperature
  - Low range -40C ± 2°C resolution
  - High range 125C ± 2°C resolution
- CAN bus commands
  - '0' - '9' indicate speed for motor rotation
  - '-' indicate maximum speed for motor rotation

Braking Subsystem

- Distance
  - Range of 30 cm to 500 cm for prototype
    - 1-mm reading to reading stability, 0.1% accuracy under standard temperature conditions
    - Analog output between 0V and 5V
    - Minimum sensing range of 30 cm
- CAN bus commands
  - Receives character 's' indicate brake ON
  - Receives character 'g' indicate brake OFF

Levitation Subsystem

- CAN bus commands
  - Receives character 'f' indicate pod forward motion
  - Receives character 'b' indicating pod backward motion
  - Receives character 'i' indicating pod idle state

Individual Sensors Subsystem

- Accelerometer + Gyroscope + Magnetometer
  - Provides XYZ direction acceleration data
  - Provides XYZ direction gyroscope data
  - Transmits data to master node via I2C protocol to be displayed on screen
- Temperature, Altitude, and Barometric Pressure Sensor via I2C
  - SoC barometric pressure and temperature sensor
  - Transmits data to master node via I2C protocol to be displayed on screen
- Analog Temperature Sensor
  - Analog data input into ADC pins of PIC18F4685

CAN bus Communication Subsystem

- CAN Low / CAN High
  - 1 Mb baud

Subsystem Control from User Console

- Motor/Braking Control
    - Motors receive duty cycle of PWM - also controlled internally through temperature feedback loop.
    - Linear actuator for brakes receives digital “10” for positive voltage across the actuator to allow extension. Receives “11” for no motion.
- Linear Actuators for Levitation Control
    - Linear actuator for extension receives digital “10” for positive voltage across the actuator.
    - Linear actuator for retraction receives digital “01” for negative voltage across the actuator.
    - Linear actuator for no movement receives digital “11” for no voltage across the actuator.

The system shall measure the following signals from a connected PC utilizing the EIA-232 protocol:

TXD

- Data transmission line from a PC to the control system
    - Send letter ‘f’, ‘b’ ‘i’, ‘0’ to ‘9’, ‘-’, ‘s’, ‘g’ to change control system

System Outputs

The system shall output the following signals to a PC utilizing the EIA-232 protocol:

Telemetry

- Display pitch, roll, yaw
- Temperature inside and outside pod (Celsius)
- Pressure of various locations in the pod
- Acceleration using accelerometer
- Distance sensors on front and back of pod

DSR

- Data Set Ready primary handshake

CTS

- Clear To Send secondary handshake

RXD

- Data transmission line from the monitoring system to a PC

For all EIA-232 outputs:

- Mark or logical ‘1’: -3 to -15 V
- Space or logical ‘0’: +3 to +15 V

Command Received

- System will print a string to describe the input command

Sensor Data

- System will display all telemetry as described above on the Local PC

User Interface

The user shall be able to monitor system components and view real-time sensor readings to console. Additionally, the user will have the ability to control the following subsystems of the Hyperloop pod:

A. Motor System Control

The user will have the ability to control motor speed. There will be two DC motors that power the levitation system of the pod. These motors will be controlled using Pulse Width Modulation (PWM) over an

H-Bridge module to the motor terminals. The user will have the ability to control the duty cycle of the PWM to allow for motor speed control over the console. Additionally, a temperature feedback loop will be implemented to allow for automatic motor control.

B. Braking System Control

The braking mechanism of the pod will be controlled by one linear actuator. This actuator will extend under braking to allow compression pads to clamp on the track of the tube. The track and the tube will be theoretical in the scope of this project. The user will have the ability to extend the braking actuator by specifying a two-bit signal, 10, to allow a positive voltage drop across the terminals of the actuator. This positive voltage extends the actuator for compression braking.

C. Levitation System Actuator Control

The user will have to ability to control the levitation system. The levitation system is comprised of two plates below the pod. Each plate will have four linear actuators placed on the corners. An inward configuration of these plates will result in acceleration of the pod. An outward configuration of these plates results in deceleration. For the scope of this project the plates will be theoretical. This capstone project will simulate the configuration control of these plates with bare, linear actuators. Control of these actuators is previously described in Section 4A.

D. Independent Sensors/Telemetry Readings

The user will have the ability to monitor the various telemetric readings on the pod through console. Analog data will be collected through ADCs and transmitted through I2C or RS232 serial protocol to the PIC slave. All communication between subsystems will communicate via CAN to a Master node. The Master node will transmit data to the console via UART.

At full scale, the user interface will receive data via a Local Area Network (LAN). Serial data under RS232 protocol will be sent through WiFi to the PC to allow real-time data acquisition and control transmission. A terminal in the system will allow for serial RS232 data to be transferred via a transparent WiFi network. Data will be received and transmitted through console. A graphic user interface will not be provided for the scope of this project.

The user will be able to select the following measurement modes using character commands. Additionally the user will be able to monitor system components and view real-time sensor readings to console:

Character Input:

- ‘f’: Moves actuators to forward propulsion state (outer actuators extended, inner actuators retracted)
- ‘b’: Moves actuators to create braking force (outer actuators retracted, inner actuators extended)
- ‘i’: Moves actuators to idle state (outer actuators and inner actuators are at same length)
- ‘0’: Motor is at speed configuration 0, no rotation
- ‘1’ to ‘9’: Motor is at speed configuration as according to character input, increasing rotation as number increases
- ‘-’: Motor is at speed configuration 10, maximum rotation speed
- ‘s’: Braking force for friction brakes is activated, actuator for brake extends
- ‘g’: Friction brakes deactivated, actuator for brake retracts

Measurement:

- Acceleration and gyroscope data in XYZ direction, temperature of motor, temperature of surrounding, commands recently pressed.

Range/Resolution:

- Gyroscope and Acceleration resolution = ± 0.001 degrees or g respectively
- Temperature sensor (I2C) = ± 0.1 Celsius
- Temperature sensor (ADC) = ± 2 Celsius
- Ultrasonic sensor (ADC) = ± 1 mm

Use Cases

The use cases for the Hyperloop Pod Control system are given in the following diagrams. The operation of the system will be through a remote connection in which the user can send commands to the system to obtain desired measurements.

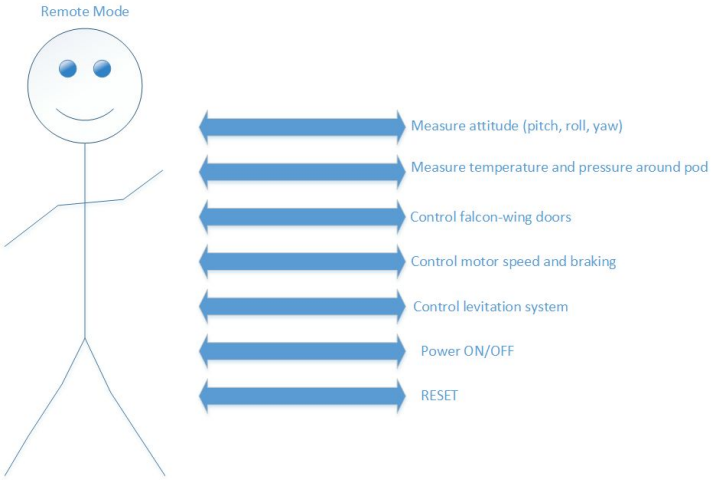


Figure 3.1.1. High-Level Use Case Diagram

Execution of the selected measurement function will be through the remote user to the system and no other commands will be supported other than through this manner.

At power ON, the default mode is Measure Attitude. All ranges will default to their highest value.

Measure Attitude The system will measure and display the attitude of the pod based on input signals from the gyroscope. The following use case is defined for measure attitude mode:

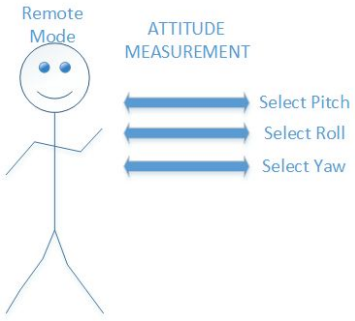


Figure 3.1.2. High-Level Attitude Measurement

Independent Measurements The system will measure and display telemetric data from various locations around the pod. The following use case is defined for measure temperature mode:

If the a temperature or pressure measurement is invalid or exceeds expected values by a significant amount, the display will present an “invalid” reading. The mode may be changed any time by a command from the remote user to the system.

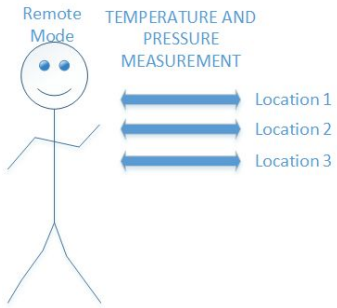


Figure 3.1.3. High-Level Temperature/Pressure Measurement

Control Motor Speed and Braking The remote user can control the speed and braking of the pod. This speed and braking system will only be ON/OFF and not variable by the user. The following use case is defined for controlling this mode:

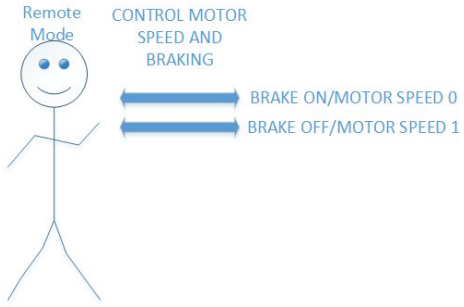


Figure 3.1.4. High-Level Motor/Braking Control

Control Levitation System The remote user can control the acceleration and deceleration of the pod through controlling the state of the actuators connected to the maglev plates.

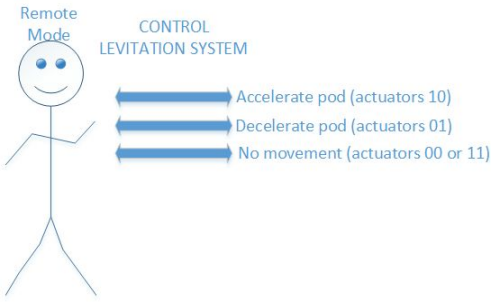


Figure 3.1.5. High-Level Levitation Control Measurement

(Also See Section 3.1 - User Interface)

Motor System Control

The user will have the ability to control motor speed. There will be two DC motors that power the levitation system of the pod. These motors will be controlled using Pulse Width Modulation (PWM) over an H-Bridge module to the motor terminals. The user will have the ability to control the duty cycle of the PWM to allow for motor speed control over the console. Additionally, a temperature feedback loop will be implemented to allow for automatic motor control.

Braking System Control

The braking mechanism of the pod will be controlled by one linear actuator. This actuator will extend under braking to allow compression pads to clamp on the track of the tube. The track and the tube will be theoretical in the scope of this project. The user will have the ability to extend the braking actuator by specifying a two-bit signal, 10, to allow a positive voltage drop across the terminals of the actuator. This positive voltage extends the actuator for compression braking.

Levitation System Actuator Control

The user will have to ability to control the levitation system. The levitation system is comprised of two plates below the pod. Each plate will have four linear actuators placed on the corners. An inward configuration of these plates will result in acceleration of the pod. An outward configuration of these plates results in deceleration. For the scope of this project the plates will be theoretical. This capstone project will simulate the configuration control of these plates with bare, linear actuators. Control of these actuators is previously described in Section 4A.

System Functional Specification

The control architecture is to control three different subsystems and provide real-time telemetry from each. Each subsystem is to take in readings of sensors located throughout the pod as well as to provide outputs that control subsystem peripherals (i.e. motors and/or actuators). Each of the subsystems will communicate to a Master node (telemetry module/data logger) that will send and receive information out onto RS232. A CAN bus will be implemented to allow for communication throughout the pod.

A. Master Node Subsystem

The Master node will be the central hub for communication and control for the pod. As the master module, the Master node will transmit real-time data and also receive information from a remote user via RS232. The Master node will communicate with each of the subsystems via an automotive-industry standard CAN bus. Through the CAN bus, commands will be sent to each of the subsystems in order to control peripherals. Data retrieved from sensors (located within different subsystems) will also be received through the CAN bus. This data logger uses a real-time operating system to allow for immediate data acquisition and subsystem control. The CAN bus will be implemented using the ISO 11898-1 standard using +5V twisted pair busses.

1) Attitude, Pressure, Temperature: The Master node will receive data from a BMP180 Barometer using I2C serial protocol that will be used to monitor the pod's attitude in real time. These measurements will be sent out to the remote user via RS232.

2) Acceleration: The Master node will take values from an included LSM9DSO accelerometer that will be utilized to monitor acceleration in the XYZ plane.

B. Motor Subsystem

The Motor Subsystem will be responsible for providing controls to motors driving the Halbach arrays. The subsystem will also be responsible for sending real-time temperature readings of the motors out to the Master node via the CAN bus. This subsystem will be implemented using a separate microcontroller that will communicate to its peripherals through GPIOs, pulse width modulated signals, and analog inputs. This subsystem will also implement a feedback loop that is to maintain motors within operating temperature by changing the duty cycle of the DC motors.

1) *Controls:* Motors will be controlled with an H-bridge in order to variably control their speed. This will require that the microcontroller used for this subsystem be able to provide pulse width modulated signals. Actuators used to control braking will require the use of GPIOs. GPIOs on the microcontroller will be connected to relay circuits that will then directly control the actuators.
2) *Data Acquisition:* Temperature sensors will be continuously taking measurements of motor temperatures. The microcontroller will require analog-to-digital converters to take the measurements from the sensors and produce digital measurements. The digital measurements will then be sent to the Master node via the CAN bus to be sent to the remote user via RS232.

C. Braking Subsystem

The Braking Subsystem will be responsible for providing controls to two electric linear actuators used to brake the pod. The subsystem will also be responsible for monitoring the analog values of two ultrasonic distance sensors. Should an object in front of the Hyperloop pod trigger the distance sensors, the actuators are automatically extended to brake the system. This subsystem will be implemented using a separate microcontroller that will communicate to its peripherals through GPIOs, digital outputs, and analog inputs. This subsystem will also implement a feedback loop that is to maintain motors within operating temperature by changing the duty cycle of the DC motors.

3) *Controls:* Brakes will be controlled with two digital signals to digital relays respectively in order to extend (brake) or retract (release) hypothetical braking pads. This will require that the microcontroller used for this subsystem be able to provide four digital output signals. Actuators used to control braking will require the use of GPIOs. GPIOs on the microcontroller will be connected to relay circuits that will then directly control the actuators.
4) *Data Acquisition:* Ultrasonic distance sensors will be continuously taking measurements of obstructive objects in the direction of travel. The microcontroller will require analog-to-digital converters to take the measurements from the sensors and produce digital measurements. The digital measurements will then be used by the braking system to determine the need to extend the braking actuators.

D. Halbach Array Position (Levitation) Subsystem

The Halbach Array Subsystem will be responsible for controlling actuators. The actuators will adjust the position and angle of the Halbach Arrays. The subsystem will also include distance sensors placed at the front and the back of the pod that monitor how close the pod is to the track. This control subsystem will implement a feedback loop that is responsible for maintaining pitch stability in relation to the track. Additionally, the gyroscope built into the Master node will be used as a redundant mechanism for monitoring the pitch of the pod.

1) *Controls:* Actuators will require that the microcontroller used for this subsystem include enough GPIOs to interface with relay circuits. The relay circuits will then take the control signals from the microcontroller and control the actuators directly. The microcontroller will continuously monitor the distance of the pod from the track and adjust its position to maintain stability.

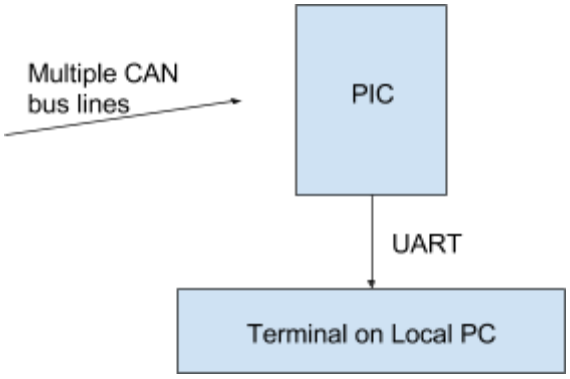
2) *Data Acquisition:* This subsystem will require analog-to-digital converters that provide distance of the pod to the track at different locations of the pod. The microcontroller will interpret these measurements and adjust the position of the Halbach Arrays directly. The measurements made by the distance sensors will also be sent to the Master node to be sent out to the remote user via RS232.

E. Display

For the purpose of this capstone, information sent to the PC will not be viewed on a graphical user interface. Data streamed to the PC will be viewed on the console. Data sent to the controls system will be sent via the console as well. The system is designed to take four different categories of measurements: acceleration, gyroscope, temperature, and distance. The user will be able to view all the different measurements on a local PC.

To send commands to any subsystem, the user sends valid characters as listed in the System Inputs section. The commands will be communicated to the subsystems through the CAN bus and the subsystem will execute according to the command. A confirmation that a command was sent to the subsystem will display on the terminal to show that a command was successfully sent.

Output Subsystem:



Timebase - The timebase system is a 32MHz internal oscillator in the PIC18F4685. This system will drive the internal control of the microcontroller. For the motor control subsystem, there are two PICs running at different clock speeds to accommodate the PWM control.

Measurement Subsystem - The measurement subsystem will measure the frequency of the input signal and display it with corresponding timer measurement. The high resolution will be in 10 ms range and the low resolution will be in 1 second range.

Power Supply Subsystem - The power subsystem will provide 5VDC to the instrument at a current level of 0.10A per subsystem. Actuators and DC brushless motors are powered by 12VDC at a maximum of 5A at full speed extension/retraction.

Display System - The display system of the prototype will provide the ability for the user to view supported data measurements and also to send commands through the CAN bus. These commands route through the RS-232 interface to the master node communication PIC18F4685 and is then sent through all CAN bus links.

Operating Specification

For the scope of this project, the controls architecture of the Hyperloop pod will operate in a standard commercial/industrial environment.

-
- The temperature should range between 0-85C.
- Humidity must be non-condensing and can range up to 85% RH.
- Power provided by 12 VDC battery to the system. The system shall operate for a minimum of 20 minutes on a fully charged battery.
- Net weight will not exceed 20lbs.
- Dimensions of the controls architecture are negligible. The controls system will be implemented across a full-scale pod in theory.

The system time base follows a temperature stability of 0-50C as well as the following Aging Rate:
Aging Rate:

90 days	$< 4 \times 10^{-8}$
6 months	$< 7 \times 10^{-7}$
1 year	$< 26 \times 10^{-6}$

Reliability and Safety Specification

The control architecture incorporates many variations of safety mechanisms for the Hyperloop pod. The submodules within the system, including motor control, braking control, and levitation control, will have redundant microcontrollers monitoring the functionality of each subsystem. Should a primary microcontroller fail, a dormant/redundant MC will be able take primary role. Additionally, the use of separate subsystems allows for the control system as a whole to continue functioning in the event of a single subsystem failure. For example, should the braking control fail, the levitation control and motor control will still be functioning.
Additionally, the motor/braking subsystem will implement two safety mechanisms. First, should the motors overheat and exceed a temperature threshold, the duty cycle of the PWM will be reduced to allow the motors to reduce RPM and cool. Secondly, an active-low braking mechanism will allow the Hyperloop pod to brake under complete power loss. A backup, 12 VDC battery will allow the braking actuator to extend and all the controls system to maintain functionality. This is implemented for scalability. Should the Hyperloop pod experience power loss in industry, communication with the pod will be maintained.

The system shall comply with the appropriate standards

- Safety: UL-3111-1, IEC-1010, CSA 1010.1
- EMC: CISPR-11, IEC 801-2, -3, -4, EN50082-1

MTBF: Minimum of 10,000 hours

3.2 - Design Procedure

Initially, the design of the system began with understanding the high-level requirements of the Hyperloop Pod system. Various components of the Hyperloop pod needed to be addressed including subsystem controllers and data acquisition of telemetry readings. A high-level block diagram was created and revised once the major subsystems of the pod were determined: Motor control, Braking control, Levitation control, and Independent Sensor monitoring.

The Motor controller needed to variable change the speed of two brushless DC motors that will eventually provide rotation to the magnetic, Halbach Array rotors found in the Levitation system. Motor speed was determined via the duty cycle of the Pulse Width Modulation (PWM) signal sent from the motor PIC.

The Braking controller needed to extend or retract two electric linear actuators that act as a compression braking mechanism about the I-beam on which the Hyperloop pod travels. During a braking sequence, the electric actuators needed to extend to allow compression pads to clamp onto the I-beam. Otherwise, the braking actuators needed to stay retracted during movement of the Hyperloop pod. Additionally, autonomous braking was implemented through use of ultrasonic distance sensors. The design implemented automatic braking should an object reach < 2 meters from the pod.

The Levitation controller needed to actuate two subframes below the mainframe of the pod. These subframes each contain a Halbach array of rotors that allow for acceleration or deceleration of the pod (*See Section 3.5 Hardware Implementation)*. Therefore, the actuation distances needed to be exactly calibrated to match the requirements outlined for the proper acceleration forces to act on the Hyperloop pod.

The Independent Sensors needed to monitor various characteristics of the Hyperloop pod to be displayed for the user through serial UART communication to console. These sensors needed to monitor acceleration, roll, pitch, pressure, altitude, and temperature of various locations throughout the pod. Therefore, the design procedure included researching and utilizing various digital and analog sensors to accommodate the telemetry requirements of the Hyperloop pod.

3.3 - System Description

The system shall be able to control various subsystems of the Hyperloop pod as well as receive and display telemetric data from various sensors placed across the pod. All control and data signals will be placed on a systemwide CAN network. The CAN network will allow all subsystems to receive and act on control commands given from the user as well as allow the user to receive data packets from all subsystems. Individual subsystems will implement various closed-loop feedback lines to allow for responsive, automatic control independent of user commands. Closed-loop feedback include real-time temperature monitoring of the two DC brushless motors for autonomous thermal control as well as ultrasonic distance monitoring for autonomous braking. The high-level block diagram of the various systems of the Hyperloop pod can be seen below in Figure 3.3.1.

FINAL DETAILED COMPLETE FULL SCALE SYSTEM BLOCK DIAGRAM

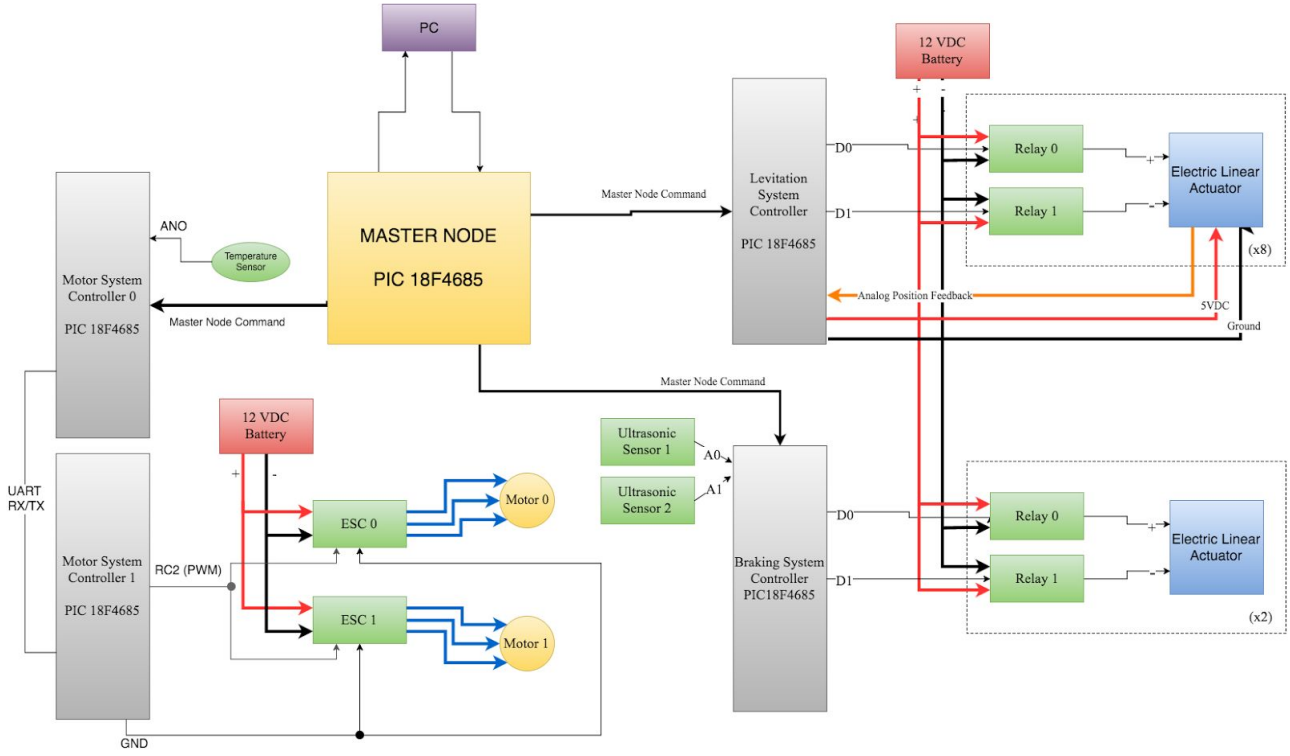


Figure 3.3.1 High-Level Block Diagram

Additionally, the Hyperloop Controls Architecture constructed for this capstone will be implemented in the full-scale Hyperloop pod being built by the UWashington Hyperloop team. The current high-level block diagram of this system can be seen below in Figure 3.3.2.

FULL SCALE SYSTEM BLOCK DIAGRAM

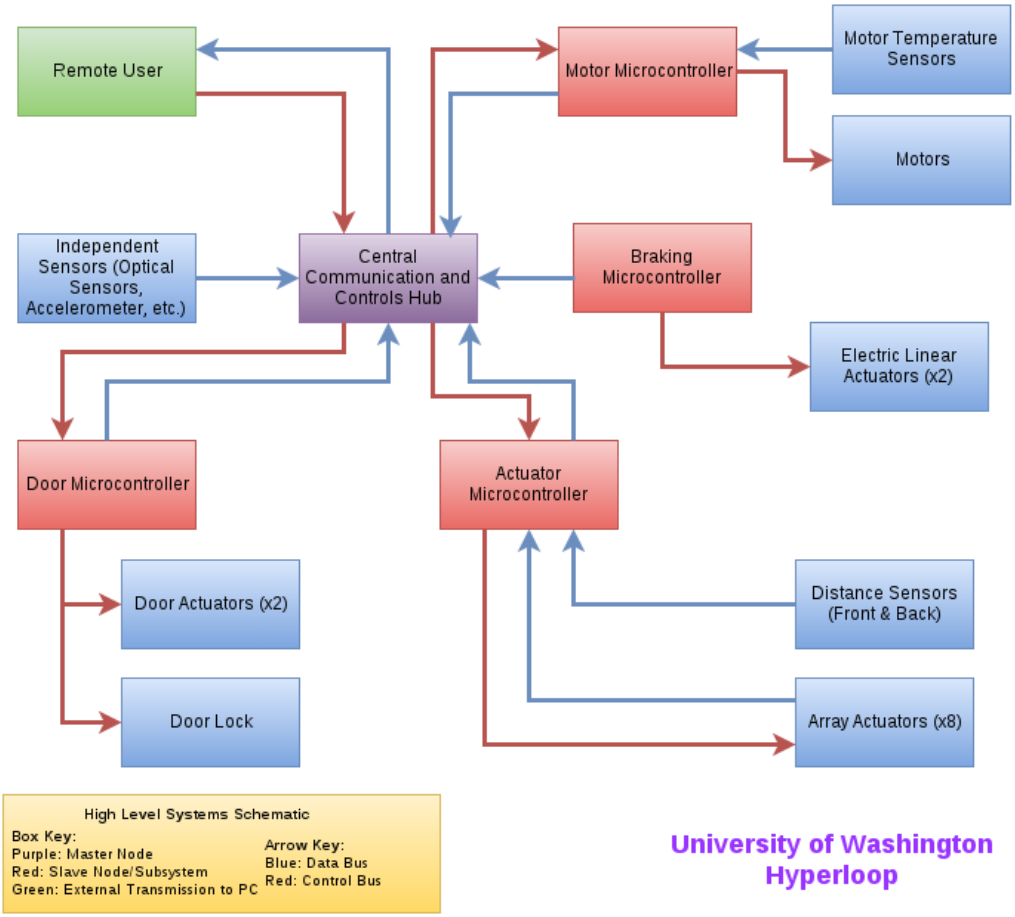


Figure 3.3.2 Full Scale System Block Diagram.

HIGH LEVEL SYSTEM TIMING DIAGRAM:

A standard CANA message will a proper frame. Below in Figure 3.3.3 is an example of a message. The frame includes a start bit which bring the bus lines to their dominant state. The start bit is followed by an 11-bit service identifier field. The frame include the length of the data in bytes, the payload of the frame which can be one to eight bytes. A cyclic redundancy check (CRC) field may also be included to provide error detection. Note that the bus in its recessive state corresponds to a 1 and in its dominant state it corresponds to a 0.

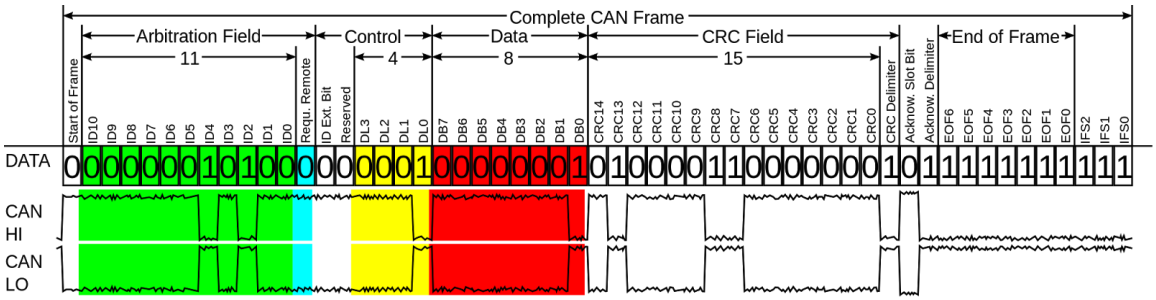


Figure 3.3.3 Standard CANA Frame

3.4 - Software Implementation

The software implementation done in this project was split among four different group members for four subsystems. Most of the software implementation was done in MPLAB X IDE using the XC8 compiler. The C programming language is used to program the PIC18F4685 in each subsystem. For the Raspberry Pi and independent sensors, the software implementation was to use several libraries for obtaining data from the sensors, as well as a Python script to display sensor data on the terminal. The following sections will document the software implementation for each subsystem and navigate through each subsystem's initialization and execution process.

Master Node/Central Hub Subsystem and Independent Sensors

The Master Node Subsystem is responsible for maintaining a connection to a PC or a Raspberry Pi over UART/RS232. The software for this subsystem was written in C and programmed onto a PIC18F4685 using MPLab X. Initially software is to initialize all modules within the PIC that will be needed for operation and configure them with the proper settings, namely the system clock, CAN controller, and UART module.

Once the needed peripherals were configured, software written to utilize the CAN controller and UART module. With the functionality fully enabled, the PIC controlling used in the master node needed to poll the UART to receive commands from the PC/Raspberry Pi. Software was written to interpret the UART commands (received as bytes) and send out new commands to the CAN bus with corresponding SIDs and encoded information. The master node was configured to receive three sets of commands: levitation commands, motor commands, and braking commands. Levitation commands were set up as follows: If ASCII characters 'f', 'b' or 'i' were received, CAN messages with an SID of 0x00A were sent out to the CAN bus with a payload of one byte of 0x01, 0x02 and 0x00, in that respective order.
Motor controls were set up as follows: If ASCII characters '0' through '9' or '-' were received, CAN messages with an SID of 0x007 were sent out to the CAN bus with a payload of one byte of 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09 and 0x0A, in that respective order.
Brake controls were set up as follows: If ASCII characters 'g' or 's' were received, CAN messages with an SID of 0x00B were sent out to the CAN bus with a payload of 0x00 and 0x01, respectively.

The master node is to send feedback to the serial terminal when commands are sent.
A levitation command will print to serial a message corresponding to the command sent. If an 'i' is received, "Command Sent (Array Idle)" will be printed to the serial terminal. If a 'f' is received, "Command Sent (Array Forward)" is printed. If a 'b' is received, "Command Sent (Array Backward)" is printed" Similar feedback will be printed for motor and brake commands. For motor commands feedback will look like "Command Sent (Motor Speed <speed setting>" where "<speed setting>" is a number 0 through 10. A brake command will provide feedback of either "Command Sent (Brake On)" or "Command Sent (Brake Off)" depending on the command received.

Finally, the master is also is able to receive CAN messages of any SID that are sent out on the bus. When a CAN message is received an interrupt is triggered. The interrupt handler for receiving CAN messages is to read the value of the message received, which is a package with an SID indicating its origin and contents, as well a payload of four bytes that can be a 32-bit integer or double float measurement. Once the message is decomposed, the master node sends out feedback to the PC/Raspberry Pi indication what the message was and its content. This way the master node can relay information from the CAN bus to the user.

Independent sensors were enabled within the Raspberry Pi system. The Raspberry Pi was included because there are many libraries available for the sensors that are used in the system. The independent sensors require Python scripts to read data from an I^2C bus that is connected to the Raspberry Pi. The libraries/scripts can then be utilized together to report measurements periodically. This periodic reporting of measurements was not implemented in this project as it requires more knowledge and understanding of Python, but the measurements were successfully made including temperature in degree Celsius, pressure in Pascals, altitude in meters, and various accelerometer and gyroscope measurements.

Levitation Subsystem

The levitation subsystem is responsible for controlling the orientation of the Active EDS subframes below the mainframe of the Hyperloop pod. Software begins by initializing the ECAN network (including the specific SID value). Additionally, the Analog-to-Digital converter is initialized as well as the UART for substitutional bluetooth control. Once the tools are initialized, the system monitors the status of the ECAN receive buffer on the CAN bus. CAN bus networking also uses a CAN transceiver that allows for multiple nodes to be placed across the CAN network. The end nodes of the CAN bus must include a series resistor across the HIGH and LOW Tx/Rx lines. The levitation system receives byte characters from the master node that represent three orientations: idle - “i”, forward acceleration - “f”, and backward acceleration - “b”. Data is received by the PIC controller through the CAN network placed across the pod. The levitation system recognizes all data on the bus but only acts upon commands that reference the levitation subsystem Slave ID (SID). The SID for this subsystem is 0x00A. Based on the commands of the master node, this subsystem extends or retracts eight linear actuators to properly accelerate or decelerate the Hyperloop pod. Based on the tangential velocity or “tip” velocity of the magnetic rotors below the frames, the pod will either be repelled in the forward direction or backward direction. An example of the subframe, actuators, and magnetic rotors in various orientations can be seen below in Figure 3.4.1.

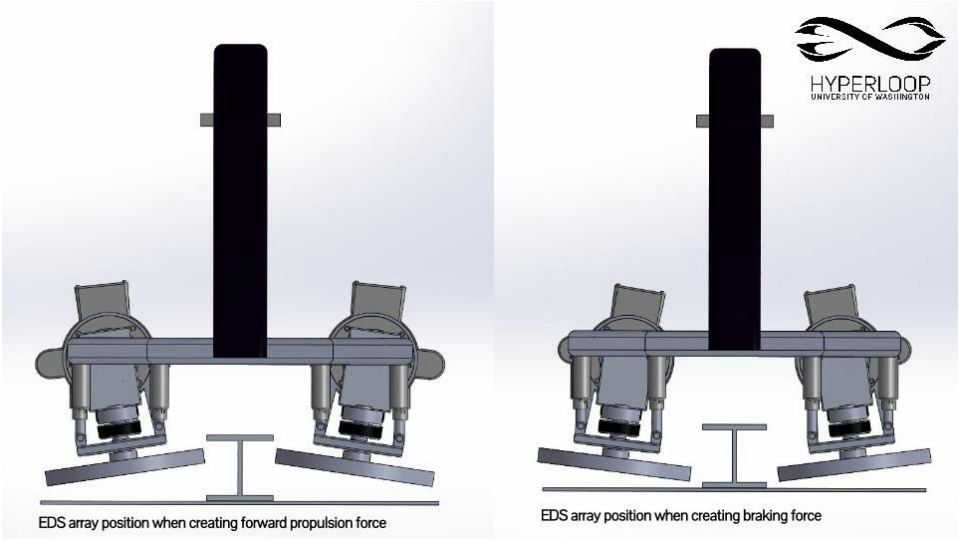
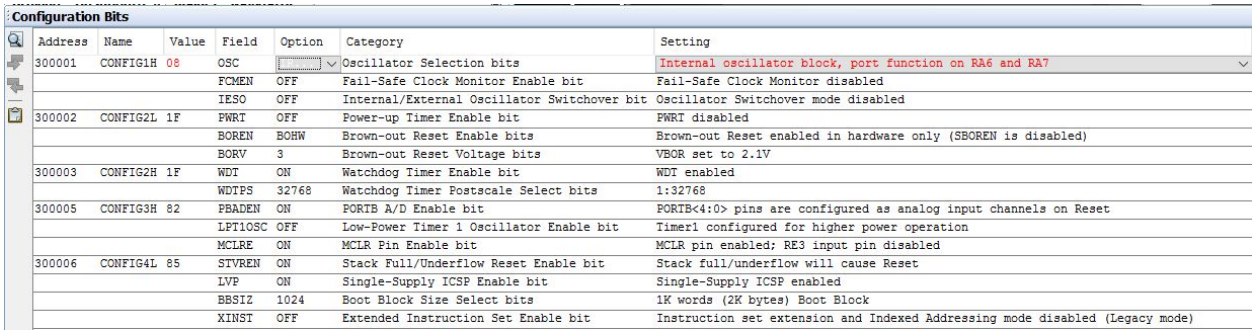


Figure 3.4.1

Once a control byte is properly recognized by the subsystem, the 10-bit Analog-to-Digital converter (ADC) within the PIC monitors the current position of each of the eight actuators and sets the direction of the individual actuators based on where the current position is and where it needs to be to achieve the correct orientation. Pins AN0-AN7 on the PIC are used to delegate specific analog input channels for each actuator. The specific channels are set in software one-by-one for the ADC to read. Once the "GO_NOT_DONE" flag built into the control register of the ADC is set to LOW, the digital value on the ADC is stored and the next analog position feedback channel is sent to the ADC. The analog values representing the positions of each actuators are converted to digital byte values. A 0 value from the ADC represented a completely retracted actuator position while 1023 represented an actuator in the fully extended position. The direction of motion of the actuators are set through use of digital relays. Once the direction of the individual actuators is determined, the digital relays receive either a LOW or HIGH signal to allow a positive or negative 12 VDC across the individual actuators. 12 VDC power is supplied across each pair of relays - sixteen relays are implemented for the eight actuators used in the levitation subsystem.

Braking Subsystem

To begin, the software initializes the system components for use. This involves initializing the PIC18F4685 digital input and output pins, disabling features of the PIC that conflicted with others that were needed, initializing ECAN, and setting analog pins. In the Figure 3.4.2, the configuration bits for the PIC18F4685 in the braking system is shown. Note that the internal oscillator is used followed by a PLLEN x4 feature that multiples the 8MHz internal oscillator by 4 to produce a 32MHz internal clock.



Configuration Bits

Address	Name	Value	Field	Option	Category	Setting
300001	CONFIG1H	08	OSC		Oscillator Selection bits	Internal oscillator block, port function on RA6 and RA7
			FCMEN	OFF	Fail-Safe Clock Monitor Enable bit	Fail-Safe Clock Monitor disabled
			IESO	OFF	Internal/External Oscillator Switchover bit	Oscillator Switchover mode disabled
300002	CONFIG2L	1F	PWRT	OFF	Power-up Timer Enable bit	PWRT disabled
			BOREN	BOHW	Brown-out Reset Enable bits	Brown-out Reset enabled in hardware only (SBOREN is disabled)
			BORV	3	Brown-out Reset Voltage bits	VBOR set to 2.1V
300003	CONFIG2H	1F	WDT	ON	Watchdog Timer Enable bit	WDT enabled
			WDTPS	32768	Watchdog Timer Postscale Select bits	1:32768
300005	CONFIG3H	82	PBADEN	ON	PORTB A/D Enable bit	PORTB<4:0> pins are configured as analog input channels on Reset
			LPT1OSC	OFF	Low-Power Timer 1 Oscillator Enable bit	Timer1 configured for higher power operation
			MCLRE	ON	MCLR Pin Enable bit	MCLR pin enabled; RE3 input pin disabled
300006	CONFIG4L	85	STVREN	ON	Stack Full/Underflow Reset Enable bit	Stack full/underflow will cause Reset
			LVP	ON	Single-Supply ICSP Enable bit	Single-Supply ICSP enabled
			BBSIZ	1024	Boot Block Size Select bits	1K words (2K bytes) Boot Block
			XINST	OFF	Extended Instruction Set Enable bit	Instruction set extension and Indexed Addressing mode disabled (Legacy mode)

Figure 3.4.2 Braking System Configuration Bits

The software behind the braking subsystem consists of several helper functions, an interrupt handler, and code to handle incoming messages from the CAN bus. From the braking subsystem's point of view, the system repeats its while(1) loop checking certain boolean flags each time. These boolean flags are set by the interrupt handler which checks if there is a message from the CAN bus to activate the braking system. For example, when a byte message comes through the CAN bus, an interrupt is triggered and the code in the braking subsystem first checks if the SID matches the braking system's SID, which is 0x0B. This is a unique identifier which tells the braking system that the message it received is a message for braking.

Next, after the interrupt is triggered and the message is valid, the system will set the boolean flag FORW to true or false depending if the byte message sent is equal to 0x00 or 0x01. If the byte message is 0x00, the FORW flag will be set to false, meaning the brakes will retract and that brakes are no longer needed. If the byte message is 0x01, the FORW flag will be set to true, indicating that the brakes should immediately activate. After the receive buffer is cleared and the interrupt is cleared, the system goes back into its repeating while loop with the condition now that FORW is set accordingly. If the FORW flag is true, the instructions for extending the brakes will be executed. Else, the instruction for retracting the brakes will be executed. The control logic is output through Port D0 and D1 on the PIC18F4685 which are fed into relays. The relay switches between two terminals depending on the control logic input to the relay.

Additionally, the analog to digital converter in the PIC18F4685 is also used for distance sensors. The distance sensors are set up and initialized in the beginning of the program. In this system, Ports A0 and A1 are used which are capable of converting analog to digital signals. The same exact logic is applied for the converted digital values. If the digital value dips below a certain threshold (set to 60 for this prototype), Ports D0 and D1 will set their outputs accordingly to extend the actuators. The ADC process takes up the most time as there is a period where the program is waiting for a GO_NOT_DONE flag to be cleared before retrieving the ADC values. Therefore, there is a small amount of delay between the time when an object is close to the distances sensors and the time when the actuators extend. This is unavoidable as this is the nature of how the ADC is implemented in the PIC18F4685.

Motor Subsystem

Software in the motor subsystem supports two main functions: handling the temperature sensor and converting commands from the CAN bus into the proper PWM signal. The standard CAN software module seen previously in the other subsystems is also included here. An ADC software module is included for use with the temperature sensor, and a UART module is included to facilitate communication between the two PIC18F4685 chips used in the motor subsystem.

On motor PIC 0 (see Figures 3.5.10 and 3.5.11 to see the hardware configuration of these PICs), the main program loop reads the ADC. This integer value is then interpreted to a voltage using the equation: voltage = (reading * 5) / 1024. This voltage is then converted to a temperature using the equation: temperature = (voltage - 0.5) * 100. If the temperature exceeds 24 C, a stop motor command is sent to motor PIC 1. The ECAN interrupt for this PIC simply forwards the ECAN command data to PIC 1 via UART if temperature is below 24C.

The primary software function of PIC 1 is to properly configure the duty cycle of the PWM signal. Incoming commands via UART are simply binary numbers ranging from 0 to 10. These numbers are interpreted as 0% to 100% motor power. The integer commands are used in a lookup array to determine the proper values to write to the various PWM control registers.

3.5 - Hardware Implementation

The hardware implementation required the use of various ICs to meet design specifications. For this prototype, the PIC18F4685 was chosen because it has all the features needed for our prototype such as CAN capability, 10-bit ADC, I2C capability, RS232, and multiple digital pins for GPIO functions. As shown in Figure 3.5.1, the pinout for the PIC microcontroller is a 40 pin chip and consumes low power which is ideal for our prototype since we are running on a small number of power supply systems. Additionally, it features a 32MHz internal oscillator which allows us to minimize necessary hardware for clock control. THe PIC18F4685 has 11 channels of 10-bit A/D which supports our levitation system as we have eight actuators with position feedback that each require a A/D channel.

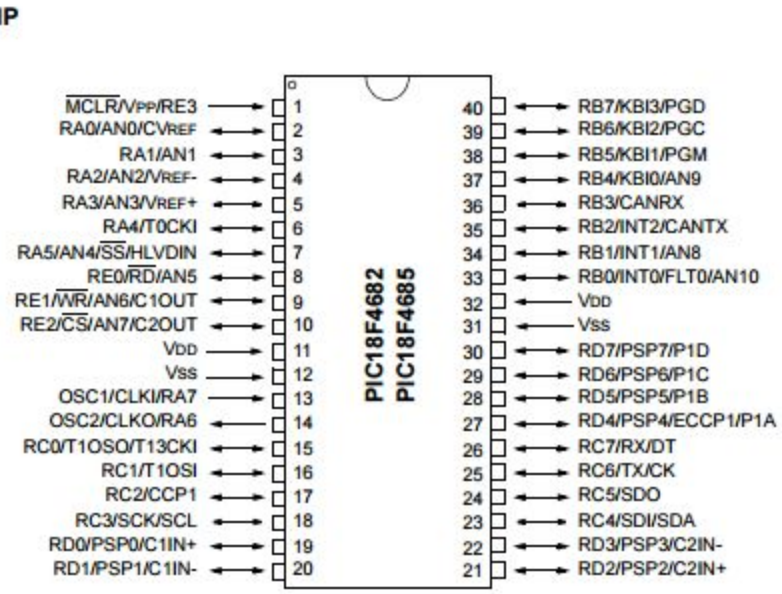


Figure 3.5.1 PIC18F4685 40-Pin PDIP Pinout

Based on lab measurements, the master node consumes about 0.10A of current at 5VDC power supply. In total, the prototype consists of six PIC18F4685 since each subsystem requires one or two PICs depending on the application. In the following sections, a layout of the hardware implementation of each subsystem will be shown.

Levitation Subsystem

The levitation controller subsystem implemented a variety of various pieces of hardware. A PIC 18f4685 was utilized as the system controller, allowing byte signals from the master node to control the orientation of the levitation arrays. Digital signals from the PIC were sent to sixteen relays that control the direction of movement of eight actuators - two relays per actuator. The electric linear actuators used allowed a closed-loop analog feedback line to represent the current position of the actuator based on a variable resistance across a 10 kOhm potentiometer. Based on the current position of the actuators, the controller set the direction of motion of the actuators to properly represent the control command received from the master node. A block diagram of the levitation system can be seen below in Figure 3.5.2.

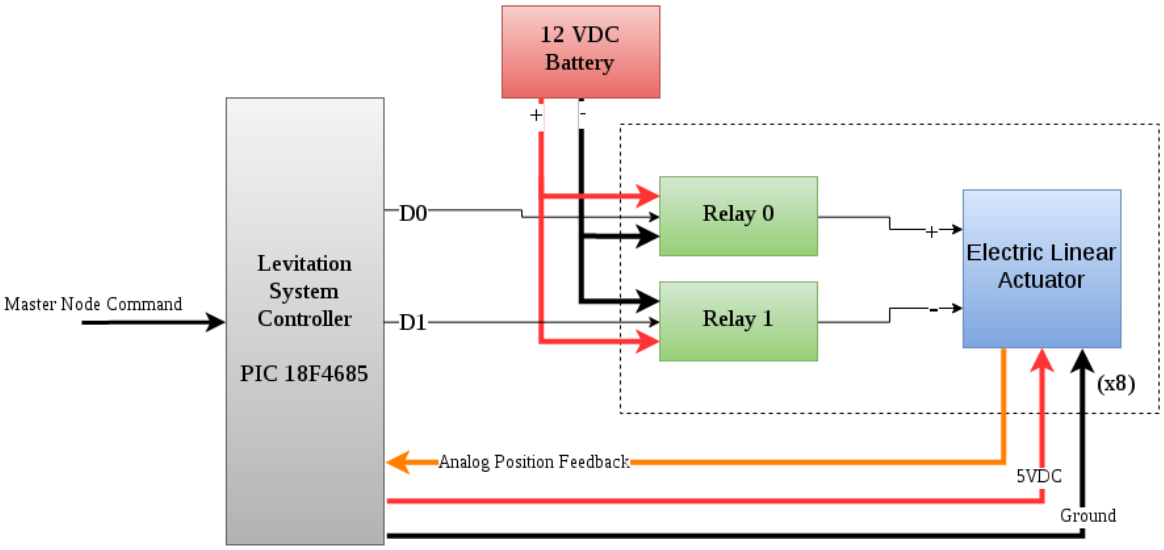


Figure 3.5.2 Block Diagram of Levitation Control

Each actuator utilized two digital relays that controlled the polarity of the 12 VDC power supply to the respective actuator. Given eight actuators are included in this system, sixteen relays were used. Each relay accepted a one-bit digital signal from the subsystem PIC that determined whether to set the voltage to the terminal or ground the terminal. An image of these relays and their respective digital signals can be seen below in Figure 3.5.3.

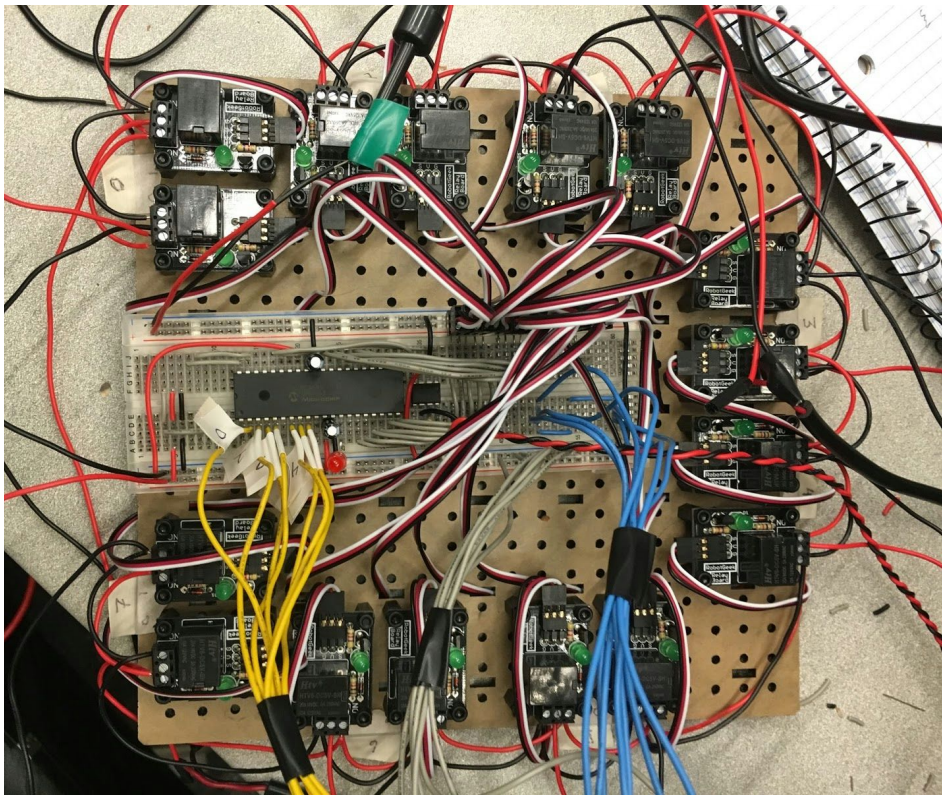


Figure 3.5.3 Actuation Controlling Digital Relays

Images of the levitation system implemented for this project can be seen below in Figure 3.5.4. One subsystem PIC was used as well as eight 12” actuators. Sixteen digital relays set the direction of motion of the actuators through use of a 12VDC battery.

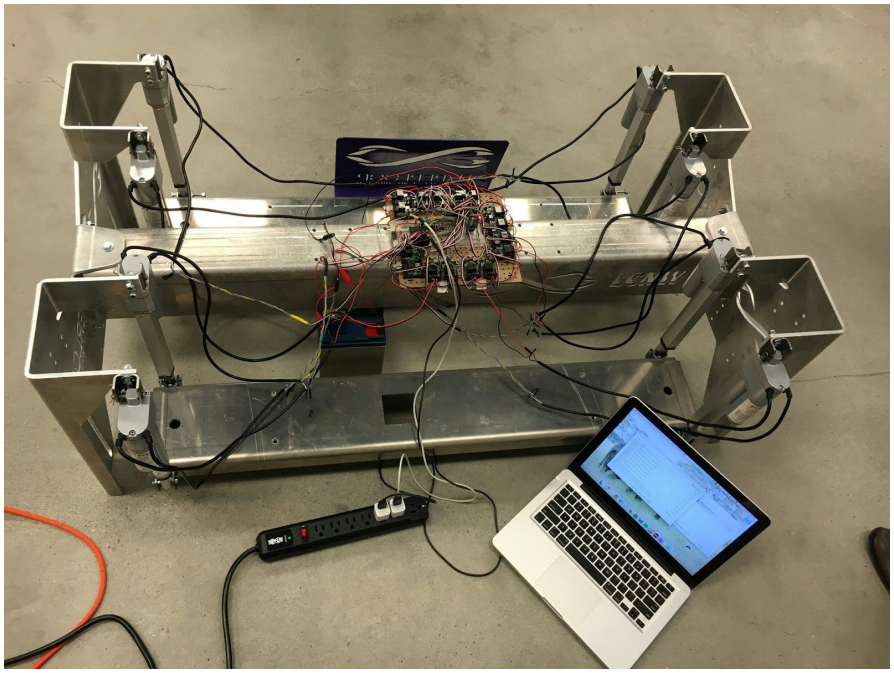


Figure 3.5.4 Levitation Control Hardware

Braking Subsystem

For the braking subsystem, the hardware needed are (1) PIC18F4685, (1) 4 Channel Relay Board, (2) Ultrasonic Sensors, (1) 3” Stroke Actuator, and (1) CAN Transceiver, some of which are shown below.

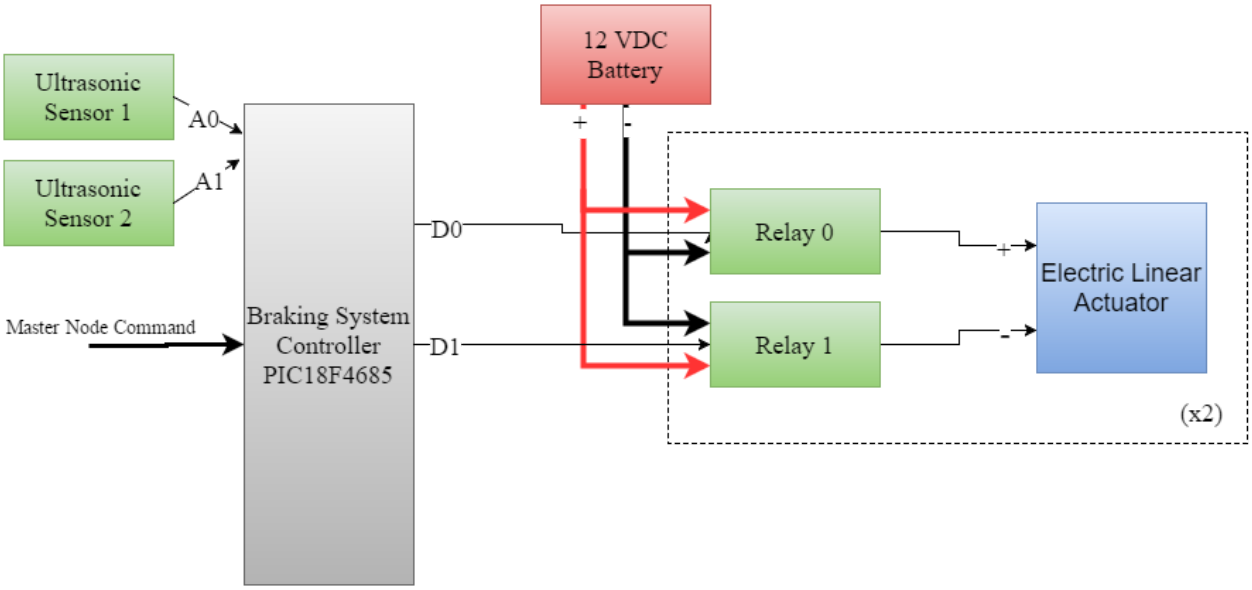


Figure 3.5.5 Block Diagram of Braking Control

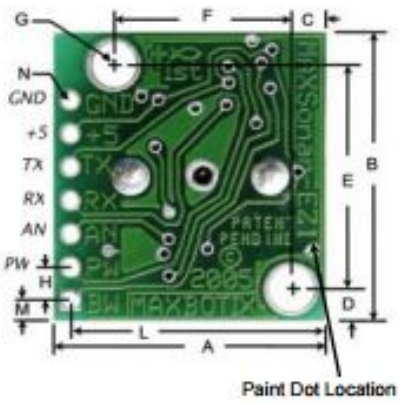


Figure 3.5.6 Ultrasonic Sensor Pinout

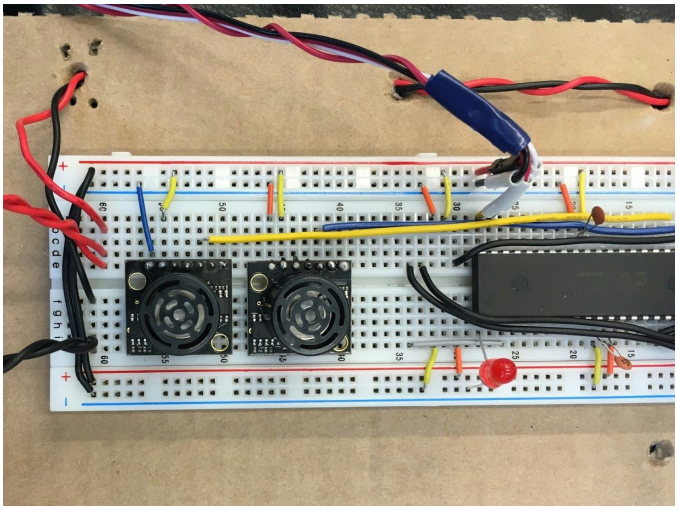


Figure 3.5.7 Ultrasonic Distance Sensors

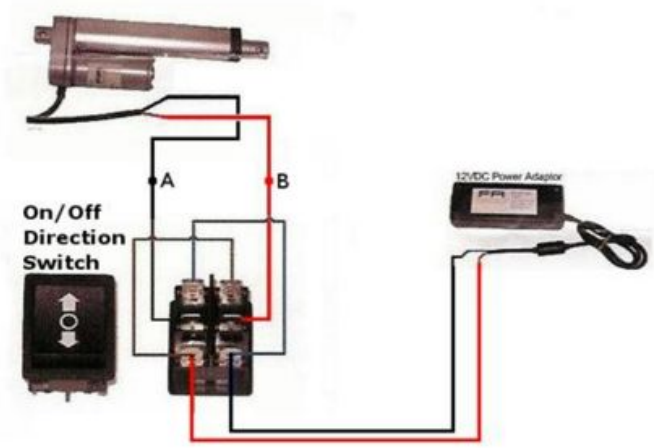


Figure 3.5.8 Actuator Wiring

Figure 3.5.9 JBTek 4 Channel Relay Board

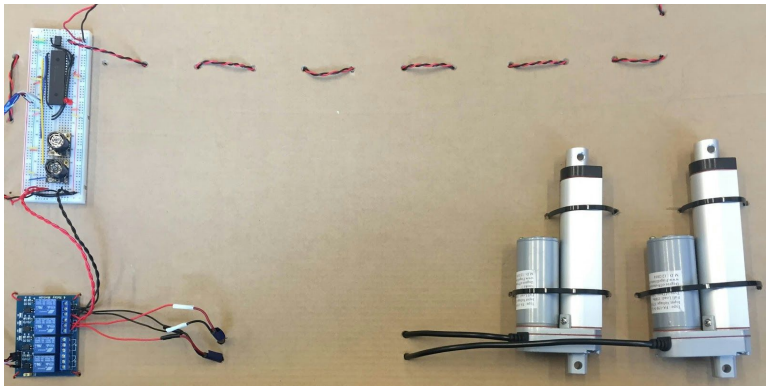


Figure 3.5.10 Braking Subsystem Final Hardware Implementation

For the actuator wiring, the on/off control is implemented in software instead of a hardware implementation. Since the actuators require a 12VDC power at 5 amperes of current, a relay board which is capable of 30VDC and 10A was used. This allowed the control of actuator retraction and extension from different power supplies. The control signal for the relays are output by the braking subsystem PIC18F4685 microcontroller. Since this relay board works on active low logic, the switch closes when IN1 or IN2 are set to low respectively. Thus, the output signal from port D0 and D1 of the PIC18F4685 were used to control the IN1 and IN2 logic. For extension, IN1 is required to be high and IN2 in required to be low; for retraction, IN1 is required to be low and IN2 is required to be high. This implements a safety system for the braking in the event of power loss, where a logical 0 to both IN1 and IN2 pins will automatically extend the brakes, thus providing a safety net for a power outage.

The ultrasonic sensors in Figure 3.5.7 were wired to the PIC18F4685 in the AN0 and AN1 (pins 2 and 3). The third pin of the ultrasonic sensor provides analog output which can be converted in the PIC18F4685. Originally, the implementation of the ultrasonic sensor used RS232 to communicate data to the master node. However, there were issues with this method and we changed the implementation to use analog values. The ultrasonic sensor is wired on the same breadboard as the braking subsystem PIC18F4685, but will be mounted on the front of a pod during the building of the prototype. Due to the wide beam pattern at far distances, this sensor picks up signals from objects to its side as well. A timing diagram for the data transmission is shown in Figure 3.5.11.

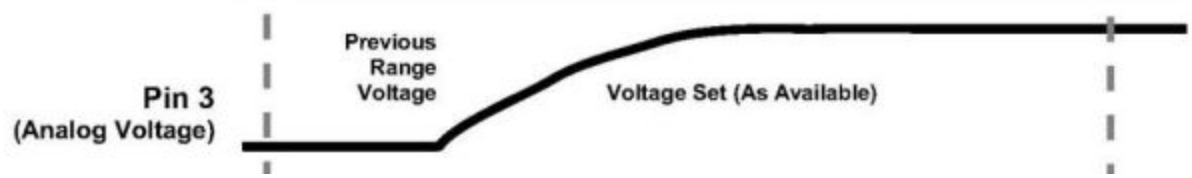


Figure 3.5.11 Ultrasonic Sensor Timing Diagram

Motor Subsystem

The motor subsystem consists of (2) PIC18F4685, (1) CAN Transceiver, (2) 30A brushless electronic speed controllers (ESCs), and (2) A2212/13T quadcopter brushless motors.

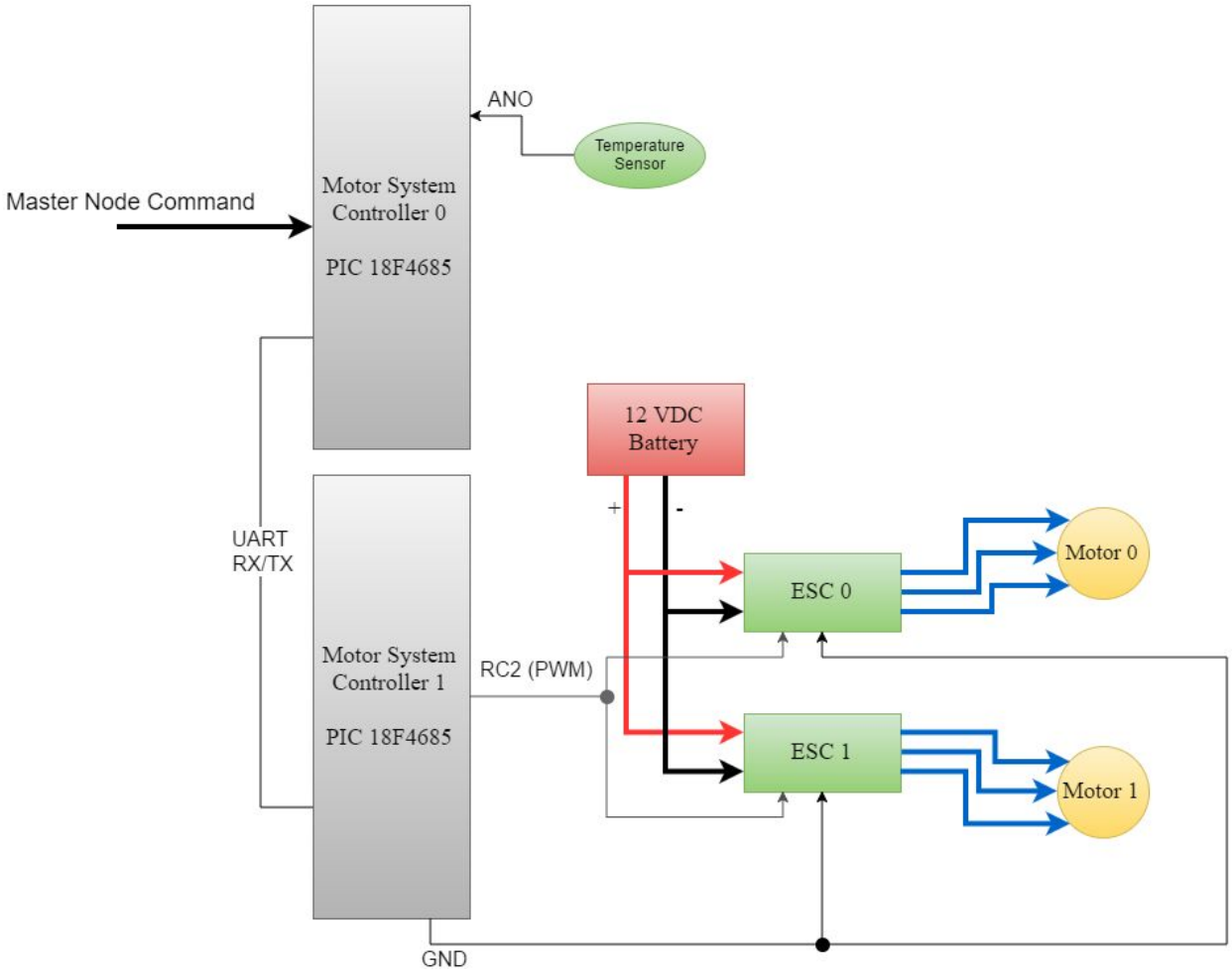


Figure 3.5.12 Motor Subsystem Block Diagram

One of the PICs is clocked at 32MHz to facilitate communication on the CAN bus. The CAN transceiver is connected to this PIC. The second PIC is clocked at 2MHz to facilitate a 200Hz PWM signal output to the two ESCs. The second PIC receives motor speed commands from the CAN PIC via UART. The UART baud rate for communications between the PICs is 2400.

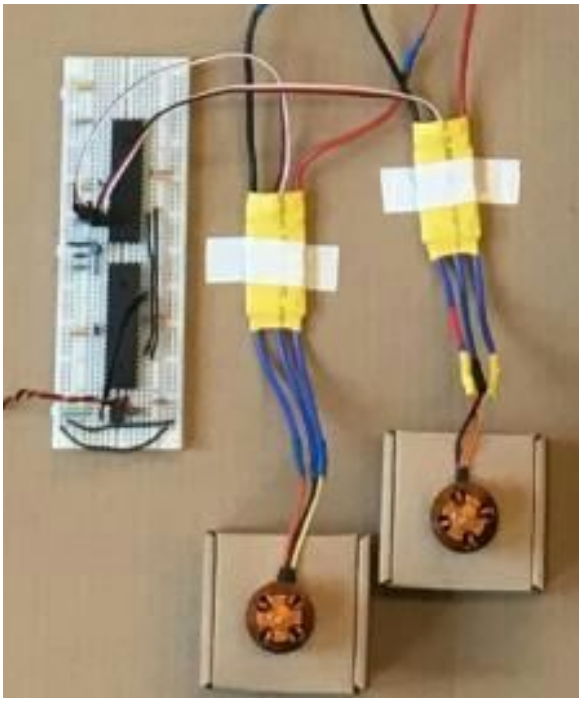


Figure 3.5.13 Motor Subsystem Layout

The 200Hz PWM signal is output from the second PIC to the inputs of the two brushless ESCs. These ESCs are powered by 12V DC and generate three-phase AC power to run the brushless motors. Motor speed is controlled by varying the phase of the AC signal. The ESC modules measure the positive pulse width of the PWM signal and interpret it a percentage of full power to run the motor. The ESC interprets a PWM positive pulse width of 1.0 ms as 0% power and a 2.0 ms positive pulse width at 100% power. Any value between these pulse widths is linearly interpreted as a percentage of full power. If the ESCs receive a 2.0 ms pulse width upon power up, they will enter calibration mode by beeping twice. The 2.0 ms pulse width is recorded as maximum power. The ESCs beep twice again indicating it is ready for the 0% power signal. Once the 1.0ms pulse width is acquired, the ESCs will beep once more and enter normal operating mode. To start in normal operating, power up the ESCs while a 1.0 ms pulse width PWM is being input. If the PWM signal is ever lost, the motors will be powered off and the ESCs will emit a short beep about every 30 seconds.

Motor PIC 0 also reads a temperature sensor via ADC. If the motor temperature ever reaches near body temperature (>32.0 C), the PIC will send a command via UART to power down the motors. The motors will not be powered on until a motor speed command is received via the CAN bus AND the temperature is below the temperature threshold.

CAN Bus Communication

Each PIC18F4685 is equipped with a CAN controller which includes several transmit and receive buffers that can be configured to receive and transmit messages. The controller on the PIC cannot, however, drive the CAN bus on its own. A CAN bus operates on a twisted pair of wires where the two carry a differential signal. Using a differential pair allows the signal to be carried longer distances without being interfered or interfering other signals. One of the wires is designated a CANH and the other CANL. The CAN bus has two states, a recessive state where no node drives the bus and a dominant state. In the recessive state the voltage of the differential falls two zero because of the two termating 120Ω at each end. In its dominant state, CANH is driven high and the CANL is driven low/negative. The PIC has a CAN Tx and Rx that function at the TTL, since a differential pair is used for this project a transceiver

is required to transmit data to the bus. On the PIC side (TTL logic), a logic 0 is represented with the dominant state of the bus. A logic 1 is represented with the recessive state of the bus. The transceiver used for this project is the MCP2551 a commonly used transceiver that works with a variety of different voltages; ±5V, ±12V and ±24V. For this project the CAN bus is driven at 3.3V to 5V recessive state. Below is a diagram of the the MCP2551 and how it is set up.

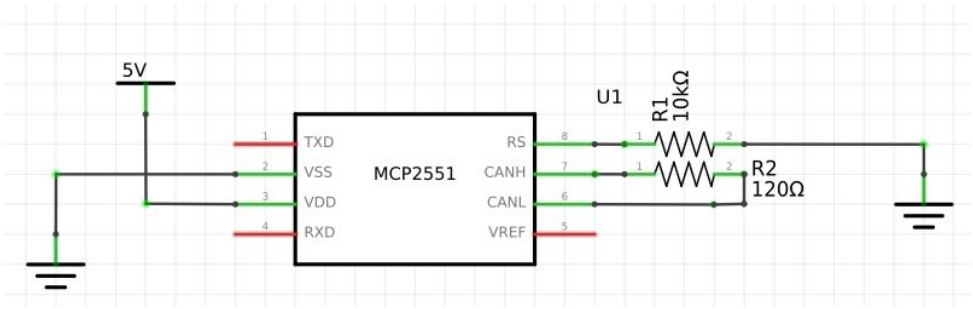


Figure 3.5.14 MCP2551 Transceiver

TXD and RXD go to a PIC's CAN Rx and CAN Tx (pins 35 and 36, respectively). CANH and CANL are to be connected to a twisted pair bus where the stubs connecting the two are to be kept as short as possible. Figure 3.5.12 above also shows a 10kΩ resistor that is used to set a current on pin 8 that dictates the slew rate of the transceiver. Also shown above is a 120Ω termination resistor that is include only when a transceiver is located at the end of a CAN bus.

4 – TEST PLAN

The proper functionality of each subsystem will be tested. These subsystems include main control, motor, braking, acceleration, and measurement subsystems. These tests will ensure the functionality of the following components:

1. CAN bus interface
2. UART interface
3. Distance sensors
4. Braking actuators and relays
5. Brushless motors and ESC
6. Temperature sensors
7. Acceleration actuators and relays

To implement this plan, each subsystem will first be tested individually. After each subsystem is determined to function properly on its own, the subsystems will be tested simultaneously to simulate the demands of a moving hyperloop pod. These tests will be repeated with various temperature and distance measurements to ensure proper functionality under all conditions.

5 – TEST SPECIFICATION

Subsystem Tested	**Input**	**Output**	**Components Tested**
Control, Motor	Various motor speed commands entered as ASCII characters via the UART interface to the control module	Various motor speeds observed with corresponding PWM duty cycle	UART, control microchips, CAN bus, motor microchips, ESC, brushless motors
Braking,	Distance between the ultrasonic	Braking actuators either	Braking microchip,

Measurements	distance sensor and a solid object is varied	deployed released	relays, ultrasonic sensors
Control, Acceleration	Acceleration subsystem commands entered as ASCII characters via UART interface to the control module	Acceleration actuators move to the specified position	Control microchips, acceleration microchips, relays, acceleration actuators
Control, Measurements, Motor	Motor set to various nonzero speeds by entering commands as ASCII characters via the UART interface. Temperature sensor heated to near body temperature (>32.0 C)	Motors deactivate when temperature sensor is heated to near body temperature (>32.0 C)	Control microchips, UART, temperature sensors, motor microchips, ESC, brushless motors

6 – TEST CASES

Control/Motor System Test Cases

Motor Control Command Character	**Expected PWM Duty Cycle (ms)**	**Expected Output**
1	1.10	10% motor speed
2	1.20	20% motor speed
3	1.30	30% motor speed
4	1.40	40% motor speed
5	1.50	50% motor speed
6	1.60	60% motor speed
7	1.70	70% motor speed
8	1.80	80% motor speed
9	1.90	90% motor speed
-	2.00	100% motor speed
8	1.80	80% motor speed
6	1.60	60% motor speed
4	1.40	40% motor speed
2	1.20	20% motor speed

1	1.10	10% motor speed
0	1.00	0% motor speed

Breaking Measurement System Test Cases

Distance from Ultrasonic Sensor (m)	**Braking Actuators**
3.00	Retracted
2.75	Retracted
2.50	Retracted
2.25	Retracted
2.00	Retracted
1.75	Retracted
1.50	Retracted
1.25	Retracted
1.00	Retracted
0.75	Retracted
0.30	Extended
0.25	Extended
0.01	Extended

Control/Acceleration System Test Cases

Motor Control Command Character	**Expected Acceleration Actuator Position (act. 1 pos. - act. 2 pos. - act. 3 pos. - act. 4 pos. - act. 5 pos. - act. 6 pos. - act. 7 pos. - act. 8 pos.)**
i	Extended - extended - extended - extended - extended - extended - extended - extended
f	Extended - extended - retracted - retracted - retracted - retracted - extended - extended
b	Retracted - retracted - extended - extended - extended - extended - retracted - retracted

Control/Measurements/Motor System Test Cases

Motor Control Command Character	Approximate Sensor Temperature	Expected PWM Duty Cycle (ms)	Motor Speed
1	Room temp	1.10	10%
1	Body temp	1.00	0%
2	Room temp	1.20	20%
2	Body temp	1.00	0%
3	Room temp	1.30	30%
3	Body temp	1.00	0%
4	Room temp	1.40	40%
4	Body temp	1.00	0%
5	Room temp	1.50	50%
5	Body temp	1.00	0%
6	Room temp	1.60	60%
6	Body temp	1.00	0%
7	Room temp	1.70	70%
7	Body temp	1.00	0%
8	Room temp	1.80	80%
8	Body temp	1.00	0%
9	Room temp	1.90	90%
9	Body temp	1.00	0%
-	Room temp	2.00	100%
-	Body temp	1.00	0%

Concurrent Testing

- Repeat braking test cases with motor at various speeds
- Repeat braking test cases with temp sensor near body temperature
- Repeat braking test cases at each acceleration state
- Repeat acceleration test cases with motor at various speeds
- Repeat acceleration test cases with temp sensor near body temperature
- Repeat acceleration test cases at each braking state
- Repeat motor test cases at each braking state

- Repeat motor test cases at each acceleration state

7 – PRESENTATION AND DISCUSSION OF RESULTS

Our group was able to demonstrate working functionality of the instrument for all operations of the prototype control system. The following sample of pictures will show different functionalities of the system. All the functional parts of our system are not shown in the report as the group demonstrated the functionality in the demonstration.

In Figure 7.1, the terminal shows various commands being sent to all the subsystems. The echo back to the terminal confirms that the command was sent through the CAN bus. Additionally, all telemetric values required by the design specification were met including monitoring of acceleration, pitch, roll, temperature, pressure, and altitude.

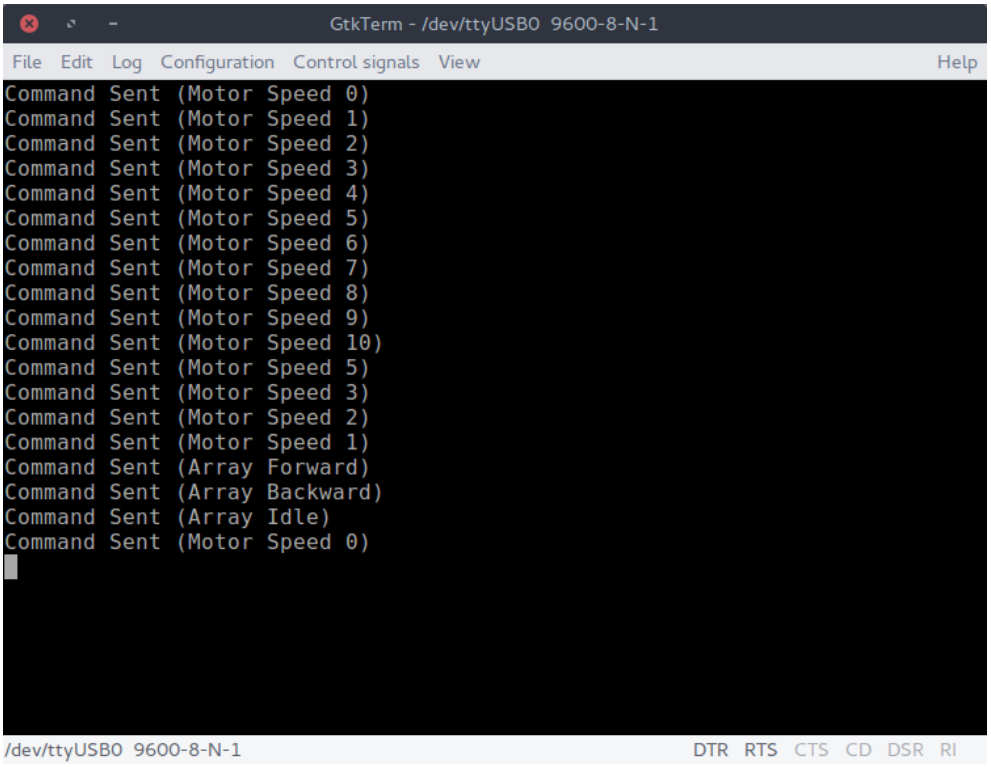


Figure 7.1 User Terminal Communication Through Console

In Figure 7.2, a stop command was sent from the Raspberry Pi to all the subsystems. The braking subsystem responded by extending the actuators to full length. Additionally, placement of an object in proximity to the ultrasonic sensors resulted in the extension of the braking actuators as expected.

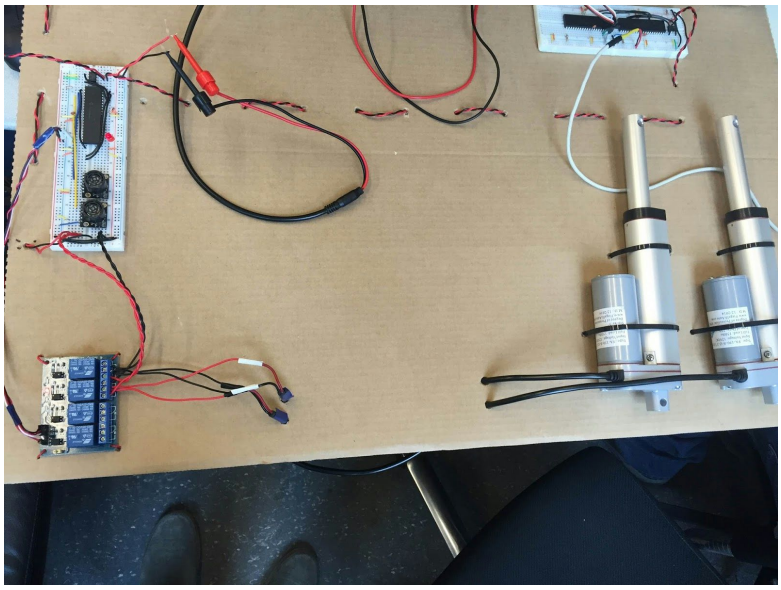


Figure 7.2 Stop Command or Object Near Sensors

Figure 7.3, a go command was sent from the Raspberry Pi to all the subsystems. The braking subsystem responded by retracting the actuators. This allows the Hyperloop pod to continue movement once the braking mechanism is released.

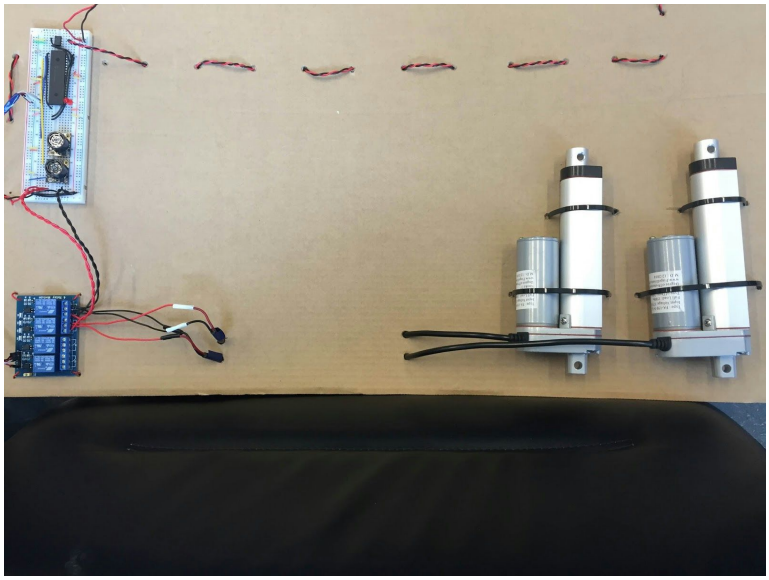


Figure 7.3 Stop Command or Object Near Sensors

Figure 7.4, a full scale system with levitation system activated. The actuators successfully mounted the mainframe to the two subframes that will hold the Halbach array magnetic rotors.

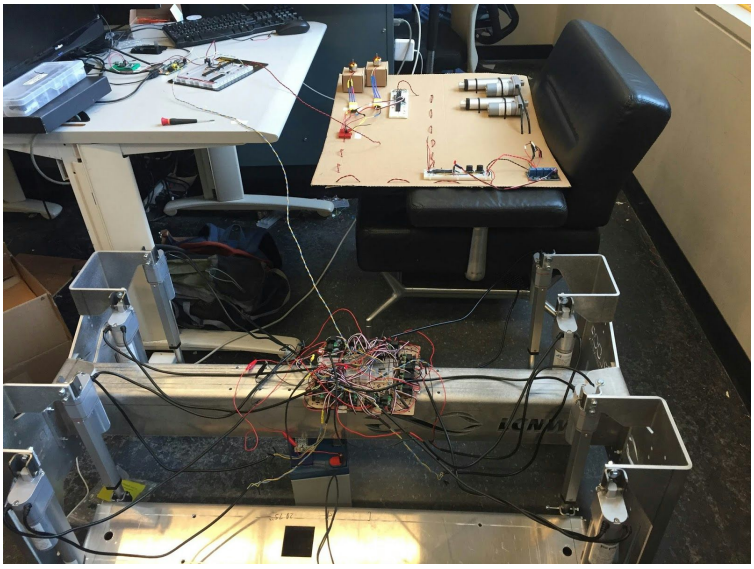


Figure 7.4 Full Scale Levitation System

Figure 7.5, the motor system successfully controlled via user commands based on the duty cycle of the PWM signals.

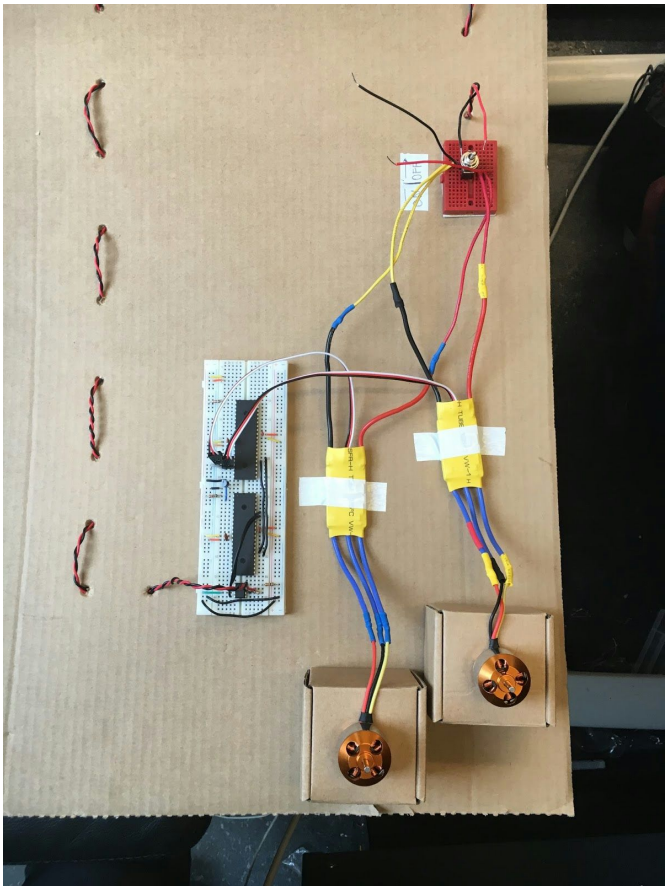


Figure 7.5 Motor Controller System

Figure 7.6 and 7.7 show the final results of the capstone project - a full-scale block diagram and the image of the hardware.

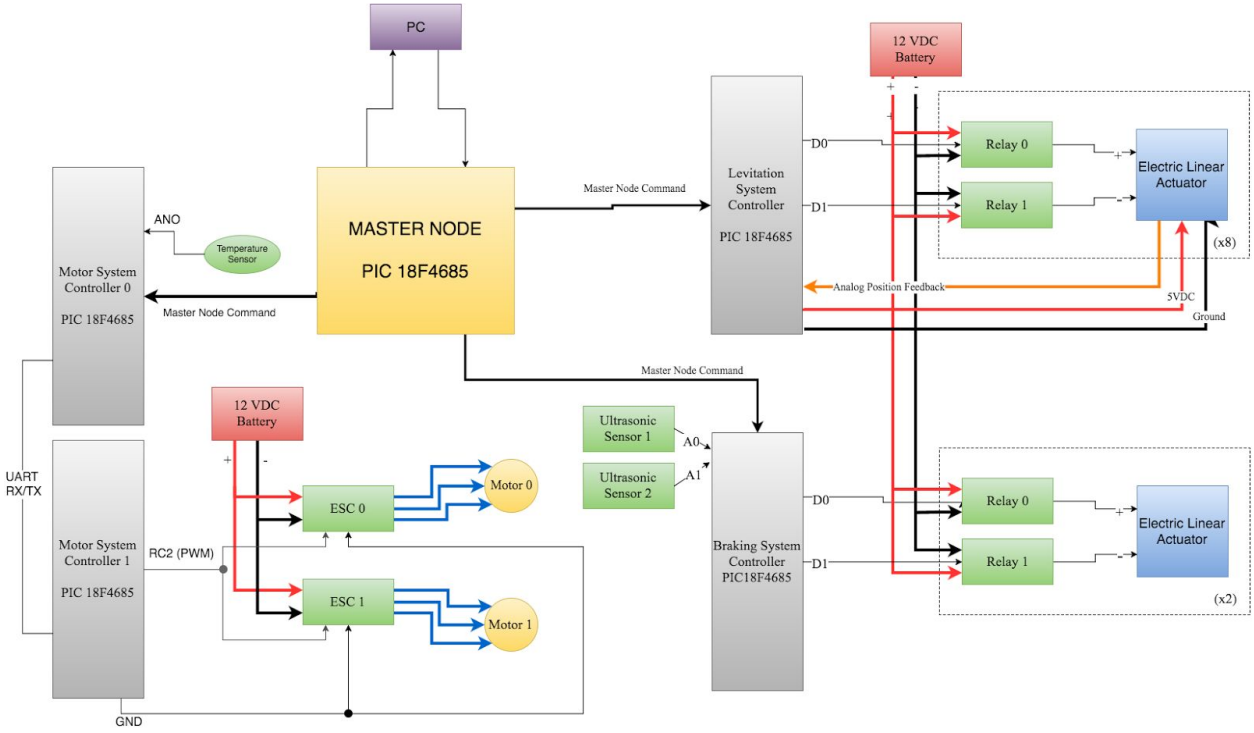


Figure 7.6 Full Scale Block Diagram

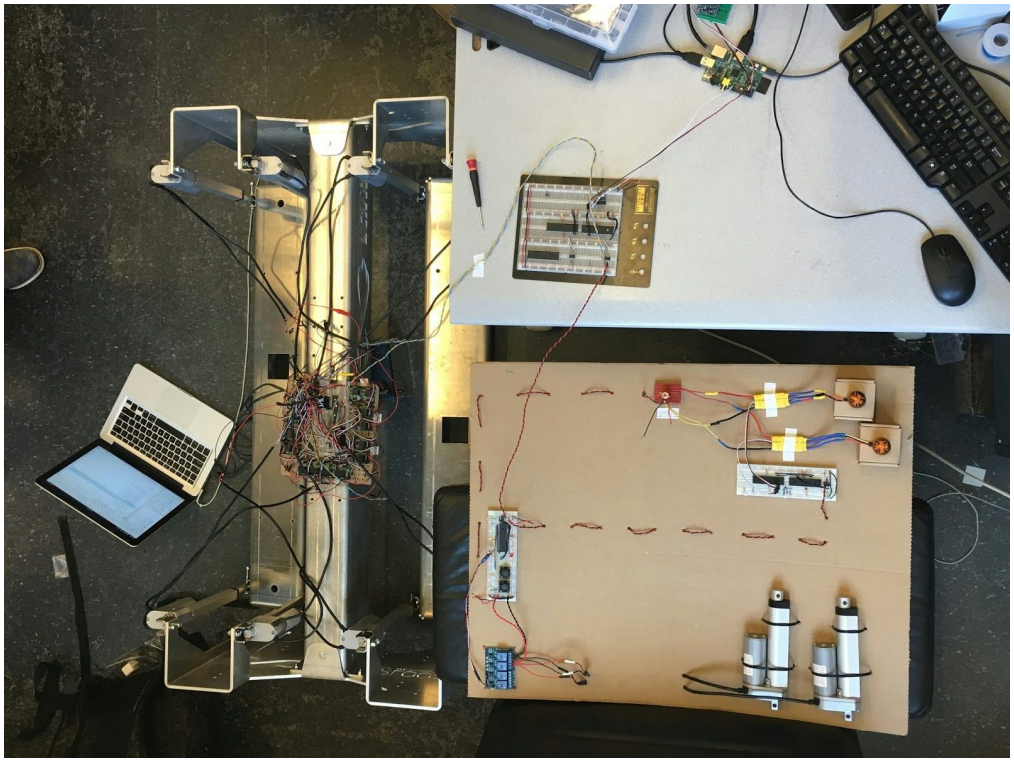


Figure 7.7 Full Scale System Image

8 – ANALYSIS OF ERRORS

There were multiple setbacks that occurred during the development and implementation of this instrument. In the software implementation, one setback occurred with the levitation system where the code was structured in a way that caused an error whenever an interrupt would occur at a certain point. This error had a low likelihood of detection yet an obvious effect on the system. As the error only occurred spontaneously without pattern, it was difficult to pinpoint the place in software that needed to be modified.

Another setback was in the braking system where the 5V and Ground connectors from the power supply were accidentally switched while powering the circuit. This resulted in a burnt relay, PIC, and a CAN transceiver.

For the independent sensors system, there were multiple setbacks for the temperature and pressure data. The pressure in the lab would constantly read 0.5atm which is half of what is expected. After a few days of trial and error, the group decided to use built in libraries with the Raspberry Pi instead to read from the sensors. The output data was much more consistent and accurate with what was expected. However, learning how to use the PIC to read from the sensors taught us how to implement I2C from scratch.

As advice for future groups, ordering parts from unreliable sources is not suggested. This cost a lot of wasted time waiting for a shipment of microcontrollers to arrive.

9 – FAILURE MODE ANALYSIS

The expected failure modes for any signal are stuck at zero errors (SA0) and stuck at one errors (SA1). Bridge errors where two signals are unintentionally connected are outside the scope of this report. The possible failure modes, their effects, and solutions are summarized in the table below.

Failure Mode	**Signal**	**Failure Causes**	**Failure Effects**	**Likelihood of Occurrence**	**Likelihood of Detection**	**Recommended Action**
SA0	UART output (from control PIC to PC)	AtMega chip failure	Results not displayed in terminal on PC	Low	High	Replace AtMega chip
	UART input (from PC to control PIC)	PC serial output failure	Commands not sent to command module	Low	High	Try a different PC
	CAN Tx	Broken connection, faulty transceiver	CAN commands not sent from control module	Medium	High	Check CAN bus connection, check digital Tx and Rx to PIC, Replace transceiver

	CAN Rx	Broken connection, faulty transceiver	CAN commands not sent from slave systems to control module	Medium	High	Check CAN bus connection, check digital Tx and Rx to PIC, Replace transceiver
	UART output (from high clocked PIC to low clocked PIC)	PIC UART module failure or code error	Brushless motors stuck at last set speed	Low	High	Replace high clocked motor PIC, debug code on this PIC
	UART input (from low clocked PIC to high clocked PIC)	PIC UART module failure	No effect	Low	Low	N/A
	Temp sensor output (from temp sensor to ADC)	Sensor failure, PIC ADC failure	High temps fail to disable motors	Medium	High	Replace sensor, replace PIC
	Braking signal	Braking PIC Failure, possible communication failure	Braking actuators constantly retracted even when commanded	Low	High	Debug communications first using trigger LED for interrupt, then debug braking PIC
	Acceleration signals ?					
	Control system signals ?					
SA1	**Signal**	**Failure Causes**	**Failure Effects**	**Likelihood of Occurrence**	**Likelihood of Detection**	**Recommended Action**
	UART output (to PC)	AtMega chip failure	Results not displayed in terminal on PC	Low	High	Replace AtMega chip
	UART input (from PC)	PC serial output failure	Commands not sent to command	Low	High	Try a different PC

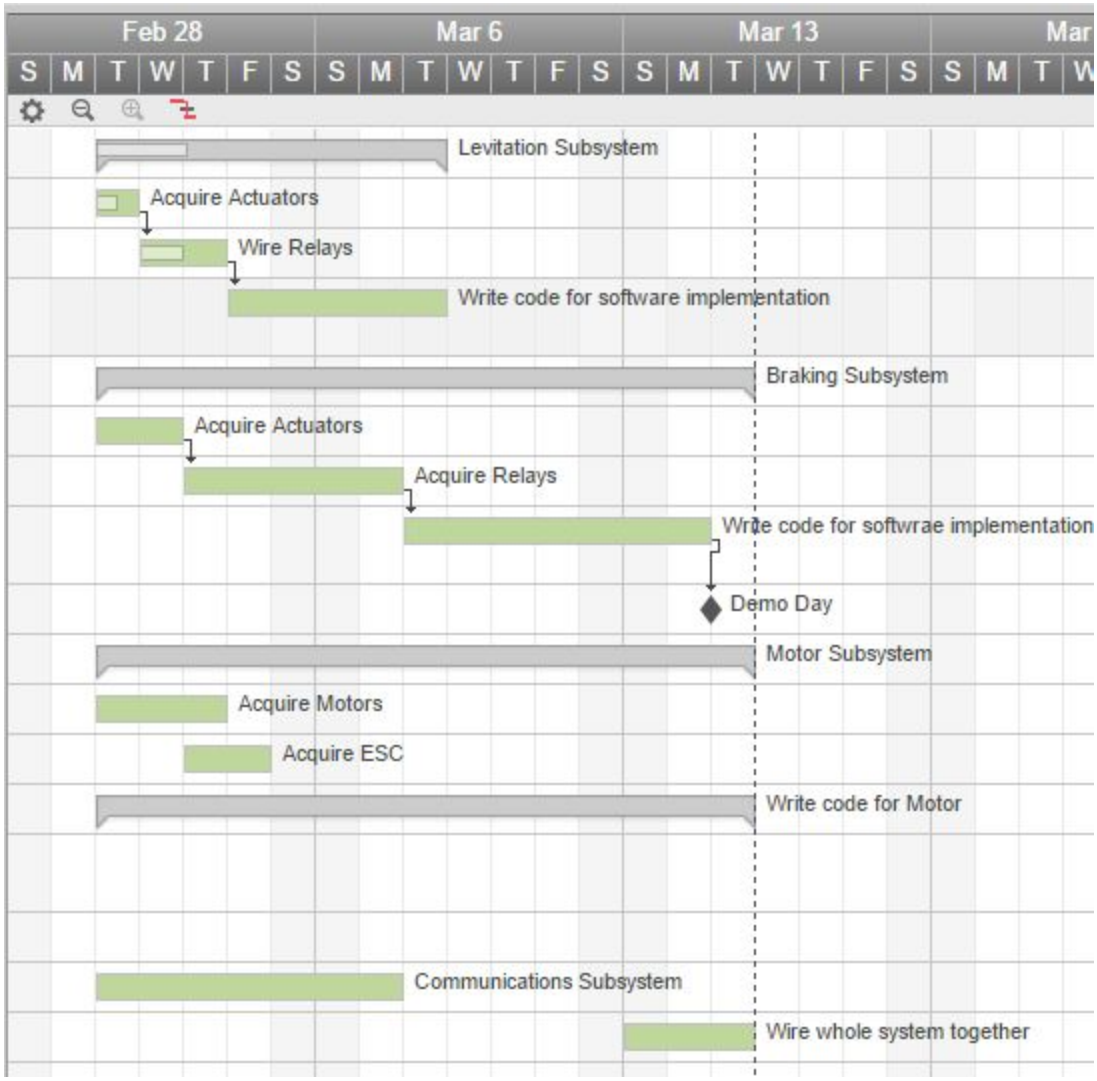
			module			
	CAN Tx	Broken connection, faulty transceiver	CAN commands not sent from control module	Medium	High	Check CAN bus connection, check digital Tx and Rx to PIC, Replace transceiver
	CAN Rx	Broken connection, faulty transceiver	CAN commands not sent from slave systems to control module	Medium	High	Check CAN bus connection, check digital Tx and Rx to PIC, Replace transceiver
	UART output (from high clocked PIC to low clocked PIC)	PIC UART module failure or code error	Brushless motors stuck at last set speed	Low	High	Replace high clocked motor PIC, debug code on this PIC
	UART input (from low clocked PIC to high clocked PIC)	PIC UART module failure	No effect	Low	Low	N/A
	Temp sensor output (from temp sensor to ADC)	Sensor failure, PIC ADC failure	Motor disabled at all temperatures	Medium	High	Replace sensor, replace PIC
	Braking signals	Braking PIC Failure, possible communicati on failure	Braking actuators constantly extended even when commanded	Low	High	Debug communication s first using trigger LED for interrupt, then debug braking PIC
	Levitation signals	Levitation PIC Failure, wiring to relays failure, poor current supply	Actuators extend to full extension, actuators retract to full retraction, actuators fail to move	Low	High	Check all wiring first, verify CAN control commands properly received by PIC.
	Control system signals	CAN twisted pair wiring failure,	Messages not received by subsystem or	Low	High	Check all wiring, verify transceiver is

		transceiver failure, SID setting incorrect	incorrect message received by subsystem			functioning, verify message contents are correct - including SID for specific subsystem

10 – FINAL BOM

Component	Details	Price	Quantity	Cost
PIC Microcontroller	PIC18F4685 I/P	$5.00	20	$100.00
Accel + Gyro 9-DOF Breakout	LSM9DS0	$24.95	1	$24.95
Barometric Pressure + Temperature Sensor	BMP180	$9.95	1	$9.95
Ultrasonic Sensor	MB1010 MAX-LV	$25.95	2	$51.90
Linear Actuator	DSTL-0418-06	N/A	10	N/A
Brushless Quadcopter Motors	A2212-13T	$7.95	2	$15.90
ESCs for Brushless Motors	ESC_30A	incl.	2	$0.00
Relays	ASM-RG-RELAY	$4.95	18	$89.10
4 Channel Relay Board	JBTEK-4CHAN RELAY	$6.99	1	$6.99
Dual EIA-232 Driver/Rx	MAX232-N	$1.60	1	$1.60
CAN Transceivers	MCP2551	$1.95	10	$19.50
Temperature Sensor (ADC)	MCP9700T-E/TT	$0.25	2	$0.50
Total				$338.37

11 – FINAL UPDATED SCHEDULE



12 – SUMMARY

This capstone project contains four individual subsystems: motor, braking, levitation, and individual sensors. These subsystems communicate through a CAN bus which is the backbone behind this prototype system. The CAN bus allow data transmission to all subsystems simultaneously and makes the system function seamlessly. For the motor system, the prototype used two brushless DC motors, two PIC18F4685, and an analog temperature sensor for a closed loop feedback motor temperature monitoring system. The braking system utilized two actuators, a relay board, and two ultrasonic sensors for distance monitoring and compressing friction pads onto the I-beam. The levitation system used eight actuators and two relays per actuator to control the configuration of the Halbach arrays which will be implemented on the pod in the future. Overall, the system was fully functional as designed in the specifications and will be ready for a handoff to the UW Hyperloop team to implement on the full scale design.

13 – CONCLUSION

Overall, the development of the Hyperloop Control and Telemetry architecture was successful and the system is able to perform all desired measurements and commands. One difficulty in implementing this system was calculating or deriving velocity and position from raw accelerometer data. To obtain velocity and position, mathematical double integration was needed in software but would cause significant error and noise due to accumulating errors. Designing this system was a difficult but rewarding task because our group learned many different components of embedded electronics such as I2C, CAN bus, electronic speed controllers, and also had to consider power systems and distribution. Implementing all the subsystems together presented a big challenge in distributing power to all the required components, but the group was eventually successful in meeting the specifications for the project.

14 – APPENDICES

The C code for this lab is uploaded onto Catalyst Dropbox.

14.1 - Requirements Specification

System Description

This specification describes and defines the requirements for the Hyperloop telemetry system. The system is required to monitor position, velocity, acceleration, and temperature of a theoretical pod for real-time data acquisition. The system is to be operated through WiFi via Ethernet and report measurements onto a console. The hardware will be placed on a 3' by 3' piece of plywood to simulate wiring on a mainframe similar to what is seen on the Hyperloop pod. Additionally, wooden subframes will be added to allow simulation of an Active Electrodynamic Suspension (EDS).

Specification of External Environment

The Hyperloop telemetry system shall be able to operate fully within the the low pressure environment inside the SpaceX's 1 mile long tube. The electrical equipment will also need to be installed on a shock and vibration proof chassis to dampen any turbulence during travel.

System Input and Output Specification

System Inputs

The system shall support the measurement of the following kinds of signals:

- Temperature measurements (-3 to 85°C)
  - Analog Sensor
  - System on Chip (SoC) interface through I^2C serial protocol
- Acceleration/Velocity/Position and Attitude measurements via accelerometer (up to 300 MPH)
  - SoC interface through I^2C serial protocol
- Pressure measurements via pressure sensors (0 to 1 atm)
  - SoC interface through I^2C serial protocol
- Distance sensors for pitch stability (up to 3 inches)
  - Distance Sensor
- 12 VDC 5A (60W) Power supply to eight linear actuators

System Outputs

The system shall display the following kinds of telemetry signals on the console of a PC. Communication between PIC will be done through CAN 2.0A serial protocol. The Maser CAN will transmit data to PC via UART

- Acceleration (-4g to +4g)
- Attitude (pitch, roll, yaw)
- Temperature (-3 to 85°C) @ at least 1Hz
- Pressure (0 to 1 atm) @ at least 1Hz
- Distance (1mm to 5m) @ at least 1Hz
- Actuation control of Active EDS

User Interface

The user shall be able to view data collected from the Master CAN module on the PC.

Mode:

- Acceleration
- Attitude (pitch, roll, yaw)
- Pressure

- Distance
- Power ON/OFF
- Reset

The user shall be able to command these actions through Gtkterm:

Motor start and stop
Emergency Braking

The measurement results will be presented on the PC.

System Functional Specification

The system is intended to make four different kinds of telemetric measurements in multiple domains. These measurements comprise of acceleration, velocity, position, attitude, pressure, and distance. The measurements will be implemented for data analysis for users to determine how to control the Hyperloop pod. There will be a 0.1 resolution range tolerance for all measurements for the telemetry because the accuracy of the measurements are important to the operation of the pod. Additionally, commands sent to slave via CAN will allow for linear actuator control to simulate an Active EDS system.

For controllable functions, the user shall be able to power up the system, start the motor, vary the configuration of linear actuators, and activate the emergency brake of the pod.

Operating Specification

The system shall operate in a standard commercial / industrial environment

- Temperature Range 0-85℃
- Humidity up to 90% RH non-condensing
- Power 5V DC and 12 VDC
    - The system shall be able to operate for a minimum of 2 hours continuously to mimic the travel time from city to city.
- Linear actuation will be powered via a 12 VDC/60W power supply.

Reliability and Safety Specification

The Hyperloop system shall comply with the following safety standards:
NFPA 1600, Standard on Disaster/Emergency Management and Continuity Programs
NFPA 130, Standard for Fixed Guideway Transit and Passenger Rail Systems, 2015ed
NFPA 101, Life Safety Code, 2015ed
NFPA 70, National Electric Code
NFPA 75, Standard for the Fire Protection of Information Technology Equipment

The MTBF will be a minimum of 1,000 hours.

14.2 - Design Specification

This section is in Section 3 of the lab report.

14.3 – Circuit/Schematics Diagrams

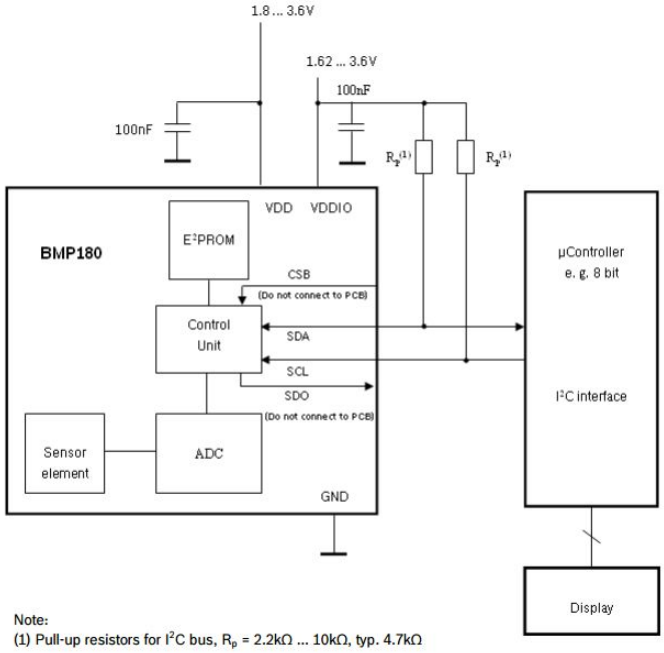


Figure 14.3.1 BMP180 Schematic to PIC

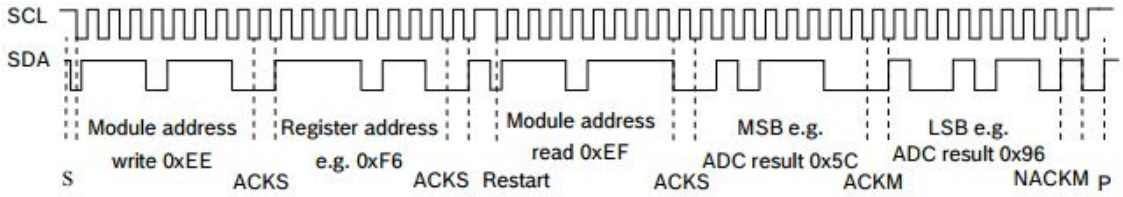


Figure 14.3.2 BMP180 Timing Diagram Read 16 bit A/D Conversion Result

14.3 – Logic Analyzer Printouts and Timing Analysis

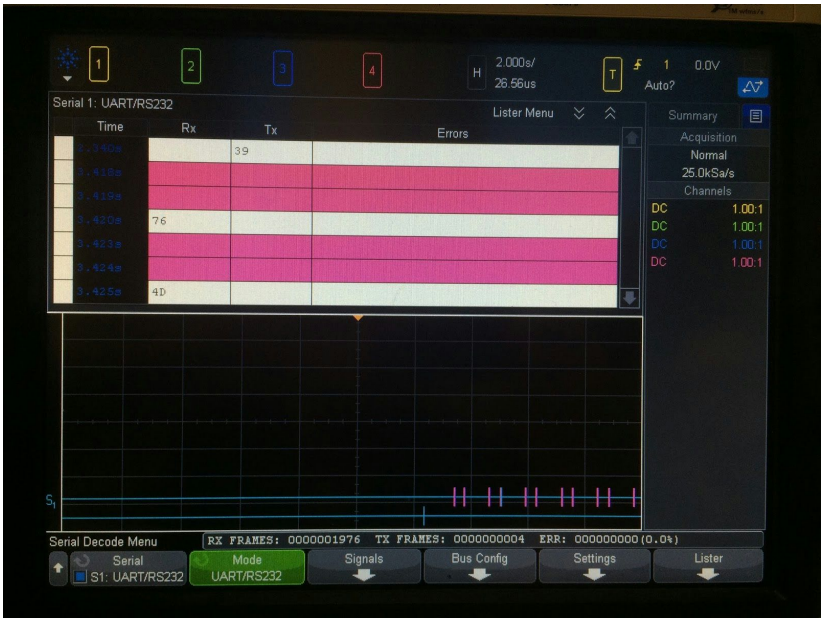


Figure 14.4.2 UART Transmit and Receive