

Spring 2019

Autonomous Combat Robot

Andrew J. Szabo II

The University of Akron, ajs262@zips.uakron.edu

Chris Heldman

The University of Akron, crh99@zips.uakron.edu

Tristin Weber

The University of Akron, trw48@zips.uakron.edu

Tanya Tebcherani

The University of Akron, tt50@zips.uakron.edu

Holden LeBlanc

The University of Akron, hcl9@zips.uakron.edu

See next page for additional authors

Please take a moment to share how this work helps you [through this survey](#). Your feedback will be important as we plan further development of our repository.

Follow this and additional works at: https://ideaexchange.uakron.edu/honors_research_projects

Part of the [Electrical and Electronics Commons](#), [Power and Energy Commons](#), [Robotics Commons](#), [Systems and Communications Commons](#), and the [VLSI and Circuits, Embedded and Hardware Systems Commons](#)

Recommended Citation

Szabo, Andrew J. II; Heldman, Chris; Weber, Tristin; Tebcherani, Tanya; LeBlanc, Holden; and Ardeljan, Fabian, "Autonomous Combat Robot" (2019). *Williams Honors College, Honors Research Projects*. 887.

https://ideaexchange.uakron.edu/honors_research_projects/887

This Honors Research Project is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.

Author

Andrew J. Szabo II, Chris Heldman, Tristin Weber, Tanya Tebcherani, Holden LeBlanc, and Fabian Ardeljan

Honors Project Final Design Report

Design Project: Autonomous Combat Robot

DT07A

Fabian Ardeljan

Chris Heldman

Holden LeBlanc

Stephen Veillette

Dr. French

April 25, 2019

Table of Contents

1. Problem Statement	6
1.1 Need	6
1.2 Objective	7
1.3 Background	9
1.4 Proposed Design Overview	13
1.5 Marketing Requirements	18
1.6 Objective Tree	19
3.0 Accepted Technical Design	21
3.1.1 System Level 0 Block Diagram with Functional Requirement Table	22
3.1.2 Hardware Level 1 Block Diagram with Functional Requirement Table	23
3.1.3 Hardware Level 2 Block Diagram with Functional Requirement Table	25
3.1.4 Hardware Level 3 Diagram (Schematic Design)	29
3.1.5 Hardware Simulation	34
3.1.6 Hardware Testbench	38
3.1.7 Software Level 0 Block Diagram and Functional Requirements Table	40
3.1.8 Software Level 1 Block Diagram and Functional Requirement Tables	40
3.1.9 Level 2 Software Block Diagram	42
3.1.10 Level 3 Software Block Diagram	44
3.2.0 Sensor Block Diagrams and Pseudo Code	45
3.2.1 Main Program Pseudocode	45
3.2.2 LiDAR Sensor	48
3.2.3 Ultrasonic Sensor	50
3.2.4 Encoder	54
3.2.5 Gyroscope/Accelerometer	55
3.2.6 Communication Protocols Software	57
3.3.0 Bill of Material as Standing	59
3.3.1 Parts List	59
3.3.2 Eagle BOM	60
3.4 Mechanical Sketch of System	62
3.5 Calculations	67
3.5.1 LiDAR Calculations	67
3.5.2 Main Algorithm Simulation	70

3.5.3 LiDAR Simulation & Data Processing	71
3.5.3 Data Resolution Calculations	76
3.5.4 Computing Calculations	77
3.5.5 Power, Voltage and Current Calculations	79
3.5.6 Mechanical Calculations	80
4.0 Schedule	81
4.1 Calendar View	81
4.2 Gantt Chart Fall Semester	82
4.3 Gantt Chart Spring Semester	84
5. Design Team Information	85
6. Conclusion and Recommendations	86
7. References	87
Appendix	89

List Of Tables

Table 1: Comparison of Combat Robot Types	10
Table 2: Design Requirements	20
Table 3: Level 0 Functional Requirement Table	22
Table 4: Hardware Level 1 Sensor Array Functional Requirement Table.	24
Table 5: Hardware Level 1 Robot Control Center Functional Requirement Table.	24
Table 6: Ultrasonic Sensor Functional Requirement Table.	26
Table 7: Gyroscope Functional Requirement Table.	26
Table 8: Lidar Sensor Functional Requirement Table.	26
Table 9: Sensor System Functional Requirement Table.	27
Table 10: Overcurrent Protectors Functional Requirement Table.	27
Table 11: Level 2 Voltage Regulators Functional Requirement Table.	27
Table 12: Level 2 LED Lights Functional Requirement Table.	28
Table 13: Sensor System Functional Requirement Table.	28
Table 14: Software System Functional Requirement Table	40
Table 15: Read Sensor Functional Requirement Table	41
Table 16: Locate Opponent and Walls Functional Requirement Table	41
Table 17: Determine Course of Action Functional Requirement Table	41
Table 18: Initialize Sensors Requirement Table	42

Table 19: Read Data Functional Requirement Table	42
Table 20: Interpret Data Functional Requirement Table	43
Table 21: Calculate Fight or Flight Functional Requirement Table	43
Table 22: Update Encoded Location/Speed Functional Requirement Table	43
Table 23:LiDAR Sensor Requirements	48
Table 24: Ultrasonic Sensor Requirements	50
Table 25: Parts List/Materials Budget List	59
Table 26: Mechanical Components of System.	64
Table 27: Baud Rate for Processor Calculations	77
Table 28: Power Calculation Table	79

List Of Figures

Figure 1: Autonomy Locomotion Algorithm.	17
Figure 2: Combat Robot Autonomy Objective Tree.	19
Figure 3: DT7A Autonomous System Block Diagram	22
Figure 4: Level 1 Combat Robot Block Diagram.	23
Figure 5 - Hardware Level 2 Block Diagram	25
Figure 6: Minimum Hardware Schematic	32
Figure 7: Schematic Diagram of the Sensor And Navigation System	33
Figure 8: Optoisolator simulation	35
Figure 9: Optical Isolator Design	36
Figure 10: Harmony GUI for PIC32MZ2048EFH064	37
Figure 11: 100 pin PIM for PIC32MZE	38
Figure 12: 60% duty cycle PWM output, 25kHz	39
Figure 13: Software Level 0 Block Diagram	40
Figure 14: Software Level 1 Block Diagram	40
Figure 15: Software Level 2 Block Diagram	42
Figure 16: Level 3 Software Block Diagram	44
Figure 17: RPLiDAR Interface Flowchart	49
Figure 18: Ultrasonic Pulses (From Datasheet)	51
Figure 19: Ultrasonic Sensor	51
Figure 20: Gyroscope Diagram	55
Figure 21: Mechanical Sketch of System.	63

Figure 22: Updated Mechanical Design - Isometric View.	64
Figure 23: Updated Mechanical Design - Planar View	65
Figure 24: Representation of Lidar Field of View	66
Figure 25: Lidar FOV With Enemy at Max Distance	68
Figure 26: Simulated Output of the Main Algorithm	70
Figure 27: Ideal Orientation, Full FOV	71
Figure 28 (left) Empty Room, Figure 29 (right) Opponent On Opposite Wall	72
Figure 30: Non-ideal Orientation, Half FOV	73
Figure 31 (left) Non-ideal Orientation, Empty Room. Figure 32 (right) Neighboring Sample Delta Array	74
Figure 33: Enemies Isolated From Walls, Non-ideal Orientation	75
Figure 34: RPLiDAR A3 Measurement Data	76
Figure 35: Distance Traveled Between Samples	78

Abstract - The objective of the project is to design and build the electrical and software systems for the autonomy system of a 60 lb. combat robot. The system should allow the robot to function autonomously. The autonomous combat robot will outperform its opponents by following a variety of combat algorithms. The autonomous system will follow an intercept or escape locomotion pattern to outperform human operators. The system will also attempt to keep the robot pointed in the correct orientation, facing the opponent at all times. While operating autonomously, the robot will use LiDAR and ultrasonic sensors to detect and attack opponent robots.

[CH]

1. Problem Statement

The following sections define the problem being solved with the autonomous combat robot.

[AS]

1.1 Need

Combat robotics is a discipline that requires much skill and a quick response time. It is often the case that the winner is not the best robot, but rather the best operator. Human operators inherently lack a consistent, fast reaction time when using a remote controlled system for combat robots. Human operators also have difficulty keeping up with the fast decision making necessary to maneuver their combat robots. It would be much faster and more effective for a combat robot to operate independently of human controls. Autonomous control of the operation and locomotion of a combat robot would outperform a manual operator.

A fully autonomous system has the ability to make algorithmic decisions, follow a locomotion algorithm, and attack with more precision than a manual operator. Therefore, an autonomous combat robot is needed to outperform opponents in movement and weapon reaction time.

[AS, FA, CH]

1.2 Objective

The objective of Design Team 07A is to design a system to enable autonomous functioning of the combat robot. The autonomous system will outperform the manually driven robots during competition.

Sensor data is used to determine whether the robot should attack or run from the enemy and which direction the robot needs to move to. This autonomous system should integrate with the robot controller system from Design The Motion and Actualization Team and the mechanical robot. The robot will indicate whether it is in full autonomous or manual override mode. The system will also be able to be armed and disarmed remotely, even while in autonomous mode. Lastly, the robot will incorporate an emergency shut off and braking system.

The autonomous combat robot will outperform its human driven opponents by following a variety of combat algorithms. While running autonomously, it will use sensors to detect its environment and the opponent. While the weapon system is reaching full speed, the combat robot will follow an avoidance and escape algorithm. When the weapon system is ready, the robot will follow an intercept algorithm to attack the opponent. The autonomous system will also attempt to keep the robot pointed in the correct orientation, facing the opponent when possible.

This project is a robotic system. It will have heavy reliance on electrical, software, and mechanical subsystems. The overall team for the combat robot consists of five electrical engineering students, two computer engineering students, and three mechanical engineering students. This team is a campus organization through Source, titled The Autonomous Combat Robotics Team.

[FA, HL, CH]

Role of Electrical/Computer Teams

- Create the control, feedback, and sensing system to control the combat robot
- Implement opponent facing tracking algorithm
- Implement control intercept or escape algorithm
- Create LiDAR sensing data interpretation and detection programming and algorithm
- Control of electric motor system
- Control of weapon system
- Autonomous sensing and control
- Create the power system to run the motors and robotic system

Role of Mechanical Team¹

- Create the robot's chassis
- Create the mechanical weapon system
- Create a mechanical drive system

[CH]

¹The mechanical engineering team has obtained preliminary approval to work with the electrical and computer engineering team from the Mechanical Engineering Department, and is in the process of receiving full approval upon the approval of the ECE's final proposal. The mechanical engineering team will synchronize its deadlines with the ECE team's deadlines in completion of their requirements with the project.

1.3 Background

The following sections provide a background and general overview of the combat robot design approach.

[TT]

1.3.1 Research Survey

Currently, the vast majority of combat robots operate by being remote controlled by an operator. Autonomy requires additional financial investments and far more work. However, the benefits of these investments are well worth the cost for first time contenders facing operators with many years of experience.

The explicit goal of combat robotics is to immobilize the opponent robot before it can do the same to one's own robot. There are many ways to accomplish this, from attacking with blunt force, to trying to impale key mechanisms of the robot, to lifting and getting the opponent robot stuck in an immobile position. The key to a successful combat robot is having both powerful offensive and defensive strategies.

Table 1 below shows a trade off analysis of the typical combat robotics weapon systems. This was used as a research tool to determine the best combat weapon system for the autonomous robot.

Table 1: Comparison of Combat Robot Types

Style	Pros	Cons
Wedge	<ul style="list-style-type: none"> - Structural integrity provides an excellent defense - Simple design - Able to get under an opponent to drive them into the wall or other hazards - Can incorporate other design features 	<ul style="list-style-type: none"> - Must have a skilled operator - Weak offense - Weak matchup against other wedges - Some competitions have banned combat bots that only use wedges
Spinner	<ul style="list-style-type: none"> - Weapon serves as both offense and defense - Low skill floor for operator 	<ul style="list-style-type: none"> - Potential to damage itself - Difficult to design - Difficult to upgrade with additional features
Drum	<ul style="list-style-type: none"> - High destructive potential - Allows for a sturdy frame - Room for additional features 	<ul style="list-style-type: none"> - Difficult to control
Crusher	<ul style="list-style-type: none"> - Potential to cause structural damage via blunt force - Allows for a sturdy frame 	<ul style="list-style-type: none"> - Requires a skilled operator to operate manually - Hard to stay within weight limits
Flipper	<ul style="list-style-type: none"> - Potential to flip other robots over for damage or immobilization - Room for additional features 	<ul style="list-style-type: none"> - Requires a skilled operator to operate manually - Weapon presents a vulnerable spot - Pneumatics limited by air tank size
Hybrid	<ul style="list-style-type: none"> - Flexibility and ability to have multiple weapon systems 	<ul style="list-style-type: none"> - Complex design - Hard to keep within weight restrictions

[TT]

The proposed combat robot will be operated autonomously, with an option to be controlled manually if desired. A major component in creating autonomous robots is sensing their surrounding environment. The sensors used for this project will be a LiDAR sensor, two ultrasonic sensors, an encoder, and a gyroscope.

LiDAR (light detection and ranging) sensors use light in the form of pulsed lasers to detect an object's distance from the sensor. When it emits a laser, the laser will hit an object and that object will reflect the laser back to the sensor. The sensor measures the time it takes to receive the reflection, and uses the speed of light to calculate the distance of the object from the sensor. LiDAR sensors emit approximately 150,000 pulses per second, so they can quickly build a "map" of their surroundings [2]. This sensor is a good fit for the combat robot, as it provides a point cloud from which an algorithm can identify both the opponent and the walls of the arena. The only downside to the LiDAR is that it does not return data at short distances, so the ultrasonic sensors must be used to complement it.

The ultrasonic sensors are similar to the LiDAR sensor in the sense that they measure their distance from an object, but they use sound instead of light. They send out sound waves at a specific frequency and wait for the waves to hit an object and bounce back. Based on the time the sound waves take to bounce back and the speed of sound, the ultrasound sensors can determine their distance from the object that the sound waves hit. These sensors may not work if they hit an object that deflects or absorbs sound [3]. Additionally, the sound waves may not return to the sensors if they are reflected at an odd angle. As a result, the ultrasonic sensors will be used by the combat bot only for short range detection in cases where the LiDAR fails to return data.

The encoder will be used to determine how fast the weapon system is spinning. It is made up

of a wheel with markers on it and a laser that returns a pulse each time a marker spins past it. As the weapon is spinning, it spins the encoder wheel with it and produces pulses at a rate proportional to the weapon rotations per minute. From there, it can be determined what the full speed of the weapon is in terms of pulses and from there the half capacity speed can be calculated. This information will later be used in the autonomous algorithm to decide whether the robot is ready to attack.

The gyroscope will be used to determine if the robot is right side up. If the robot is upside down, the autonomy is not guaranteed to function properly. If this happens, the robot will give a signal to the driver to switch to manual mode until it is right side up again. Instructions from the controller are automatically inverted as long as the robot is upside down.

[FA, TT]

Almost all combat robots at RoboGames are manually-controlled, which means their effectiveness in competition is severely limited by the skill level and reaction time of their operator. This flaw in current designs exists because the algorithms and sensors needed to have a competitive combat robot are quite complex. If one were to write such an algorithm that could effectively perform combat maneuvers and pair it with an adequate sensor array and a robust, reliable combat robot, the resulting combination could yield a very competitive end product.

This bot is similar to existing designs because it still has all of the components of a traditional combat bot. It will have a weapon as well as all of the required electrical engineering components. For example, it will have motors, actuators, controls, power supplies, programming, wireless communications, etc. It is also similar to other self-driving (autonomous) vehicles because it will use LiDAR sensors to accomplish autonomy. Note that, although LiDAR is constantly used for self-driven vehicles, autonomous combat bots are not the standard. Most combat bots have an

operator who controls the bot during a fight. Thus, making the bot autonomous is different from existing technologies.

[TT]

There are current patents on robotic systems that are similar to the one in which the team will create. Some interesting ideas can be drawn from these innovative patents.

One of these was a patent on a combat robot which used a walking system instead of the conventional wheels. This was proposed because many combat robots fail because they lose mobility due to their fragile rubber wheels being destroyed. The weapon system on this robot was of the flipping kind [7].

Another patent of interest is a combat robot which uses infrared emitters to sense its surroundings and autonomously detect and attack the opponent robot. This robot was a wedge combat design. The design mitigates the weakness of external vulnerable wheels by having them enclosed within its chassis [8].

A further patent found was on a flipping- wedge hybrid robotic combat system. This patent only discussed the weapon system design. The robot was of a wedge shape, and could function as a wedge combat robot to attack other bots by flipping them over and preventing their mobility. (If the wheel system was on the bottom of the robot). The more interesting weapon of this system was a flipping arm. The combat robot would have an arm which could reach under other robots and then lift at high velocity to send the opponent in the air. This system was driven by pneumatic circuits. It would use a compressed gas tank to drive the arm with a very large torque to flip the other robot [9].

1.4 Proposed Design Overview

For offense, the combat robot will use the drum method, which involves an upward spinning horizontal tube with extensions used to hit and possibly throw the opponent robot. This drum will be in combination with a wedge. Hybrid combat robots are often not used because of complexity of design but having a mechanical subteam will allow the team to make a more complex and effective weapon system. Most notably, the drum method is also used by Touro Maximus, a long time contender and multiple time finalist in combat robotics tournaments. The wedge method was used by the winning RoboGames combat robot Original Sin. By combining these two weapon systems and having a fully autonomous robot, it can outperform its opponents. By using autonomy, this method can be further enhanced to make sure the weapon system turns to face the opponent as fast as possible, and can follow ideal intercept and escape paths.

On the defensive side, it is a clear advantage to have a robot that can withstand being flipped. Robots that can operate on their back usually recover more effectively from being tossed as well. However, the design should also focus on a hard outer shell that resists damage to both blunt and sharp attacks. Autonomy can also help defensively, by allowing the robot to sense in all directions where the opponent is and turn to face the opponent with the weapon system before the opponent can strike from behind or from the side.

[FA]

The team has found that LiDAR is the best sensing technology for automated driving of combat robots. In 3D applications, it uses a laser beam to scan the environment with very high accuracy, and as a result is highly suited for estimating shapes of objects [10]. The data returned by

LiDAR can be interpreted as a 3D point cloud, where clusters of detected points form objects. In the arena, this 3D point cloud will show the outline of each wall, along with an outlying cluster of points where the opponent is. The closest point of any wall as well as the closest point of the opponent will be used to autonomously decide navigation.

Internally, LiDAR works by using a rotating mirror system to take panoramic picture data. It is controlled by motors and rotary encoders that determine the tilt of the mirrors used to take the pictures [12]. This mechanical complexity along with the large amount of data processing make this very expensive. Smaller, cheaper versions are used in mobile robots such as Aethon's autonomous TUG robots used in hospitals [13].

The goal of the autonomous system is to drive the robot better than a user would. Higher complexity locomotion algorithms will be used to give the combat robot a significant edge above all manual robots. This will be done by implementing two algorithms for movement. The first being an intercept or escape algorithm. The second being an algorithm that utilizes the robot's weapon design, which will attempt to keep the defensive wedge pointed at the opponent at all times.

The intercept or escape algorithm will determine if the robot should be avoiding or attacking the opponent. This will be dependent on if the chosen drum is at a sufficient velocity, or at least will be at sufficient velocity by the time it reaches the opponent. If the robot's drum weapon is at an acceptable speed, or the opponent is far enough away, the robot will use a simple tracking algorithm and traverse directly toward the opponent..

When the drum is not at full speed, and the opponent is nearby, the combat robot will use the escape algorithm. This algorithm is based on an optimal escape pattern in which the trajectory vector is set perpendicular to the current locomotion vector of the opposing robot. This will successfully escape the opponents so the drum weapon can obtain optimal speed. These algorithms will outperform manual operators as long as the speed of the robot is near or above the opponent. The full algorithm is shown below in figure 1.

The opponent facing tracking algorithm will implement a defensive strategy used by human wedge operators. The strategy is to keep the wedge always facing the opponent. This will cause the attacking opponent to drive on top of the wedge, and potentially flip over, when attempting to attack. While on top of the autonomous combat robot, the attacking robot's weapon system is less likely to hit the autonomous robot. Also giving it an opening to attack using the drum weapon.

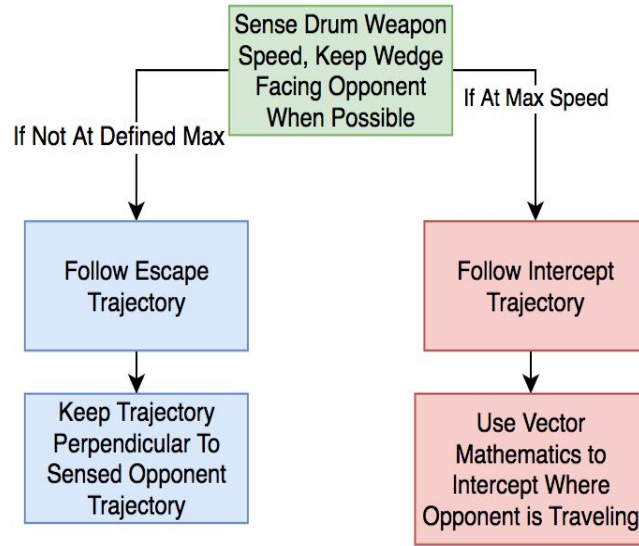


Figure 1: Autonomy Locomotion Algorithm.

[CH]

1.5 Marketing Requirements

The marketing requirements for the combat robot system are as follows:

1. The robot shall operate autonomously.
2. The sensor system shall read environment data to be sent to the autonomy system.
3. The autonomy system shall be able to differentiate between the arena walls and the enemy.
4. The autonomy system shall be able locate the enemy with respect to itself.
5. The autonomy system shall be able to make a fight or flight decision.
6. The autonomy system shall output a recommended location where the robot should travel.
7. The recommended location shall include the orientation to keep the enemy in front of the robot when possible.
8. The autonomy system shall easily interface with the system of The Motion and Actualization Team.
9. The autonomous system shall indicate if the system is still functional.
10. The robot shall be compliant with all other RoboGames rules [15].

[SV, FA, CH, TT, TW, SV, HL]

1.6 Objective Tree

The objective tree for the fully autonomous combat robot is shown in Figure 2 below. This was derived from the marketing requirements.

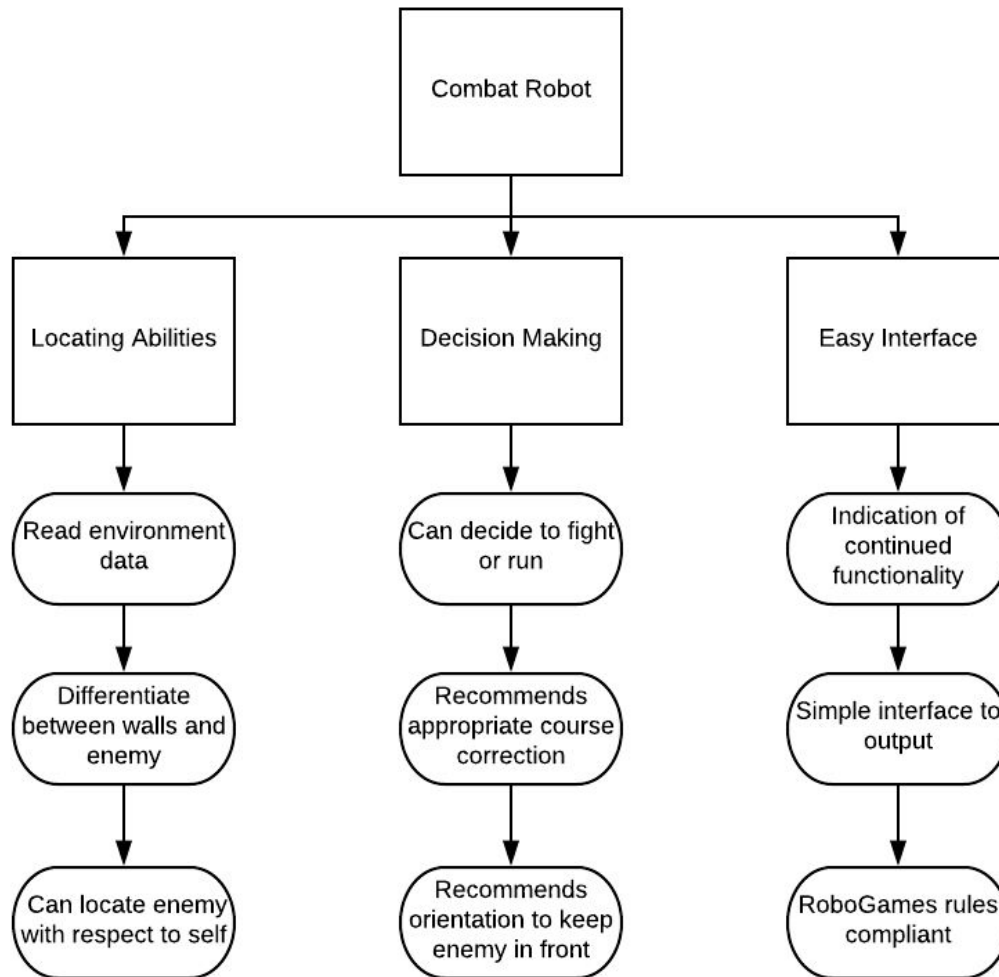


Figure 2: Combat Robot Autonomy Objective Tree.

[SV]

2.0 Design Requirements

Table 2: Design Requirements

Marketing Requirement	Engineering Requirements	Justification
9, 8	1. The autonomous system shall visually report the functionality of the autonomous system.	The operator of the robot must know when to switch to manual mode if the autonomous system fails.
3,4	2. The autonomous system shall visually report if an enemy is detected.	This should be done in order to verify that the movements/actions of the robot are due to the autonomous system.
5	3. The autonomous system shall visually report whether it is in fight, flight, or search mode.	This should be done in order to verify that the robot is following the correct course of action.
11	4. The robot and autonomy system shall operate for three minutes without interruption.	The length of a RoboGames match is three minutes. The robot must operate for the entirety of the match.
1,6,7,8	5. The autonomous system shall continuously output the recommended angle and speed (to the motion and actualization system).	The autonomous system should instruct the motor control system to charge or search for an opponent when possible (see 6).
1,10	6. The autonomous system shall sense and output a signal if the robot gets flipped over from its original orientation (to the motion and actualization system).	The autonomous system should stop operation if the robot has been flipped over.
1,2,4	7. The autonomy system shall be able to detect an object up to 12 meters away (within the field of view).	In order to effectively defend itself from an attacker and to effectively attack, the robot needs at least twelve meters to prepare.
1,2,4	8. The system must report the location of a detected enemy with no more than 5 degrees error from 20 feet away.	In order to accurately determine the recommended location, the robot needs to know where the enemy is.
1,7	9. The autonomous system shall	The robot needs to be ready to

	initialize within 60 seconds of the robot being powered up.	defend/attack at the start of the match.
11	10. The autonomous system shall weigh less than 6 pounds.	The robot shall be classified as a 60 lb combat bot and the rest of the components have been allotted the other 54 lbs.
11	11. The emergency stop shall power off and stop the autonomous system within 60 seconds.	RoboGames rules state that the robot should be powered off completely within sixty seconds.
8	12. The system shall have overcurrent protection on each board for 110% of the max current.	In order to protect the rest of the robot from malfunction, the robot shall not be allowed to draw more current than the power supply can provide.
8	13. The system shall operate on less than 144 watts.	The power supply is to be able to supply 144 to the autonomous system.
1,5	14. Robot shall not engage enemy unless the weapon ready signal has been received.	In order to effectively attack an enemy, the weapon needs to have enough momentum to do damage.

3.0 Accepted Technical Design

The sections below show system block diagrams broken down by sections.

[HL]

3.1.1 System Level 0 Block Diagram with Functional Requirement Table

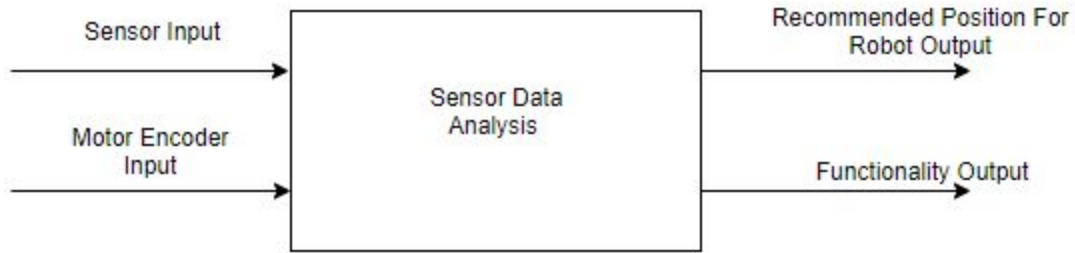


Figure 3: DT7A Autonomous System Block Diagram

[HL]

The level 0 functional requirement table, indicating the top-level inputs and outputs of the fully autonomous combat robot, as shown in Table 3 below.

Table 3: Level 0 Functional Requirement Table

Module	Combat Robot
Inputs	<ul style="list-style-type: none"> ● LiDAR Input Signal ● Ultrasonic Sensor Input Signal ● Motor Encoder Input Signal ● Gyroscope Input Signal
Outputs	<ul style="list-style-type: none"> ● Recommended Position for Robot Output: UART (output to the controller board from DT07B) ● Functionality Output: LED Light
Functionality	Determines the recommended position for the robot using an algorithm that uses the data from the sensors. This system also determines and indicates whether the autonomous system (as a whole) is functioning.

[FA, CH, HL]

3.1.2 Hardware Level 1 Block Diagram with Functional Requirement Table

The level 1 block diagram, which is an expansion of the level 0 diagram in Figure 2, is shown in Figure 4.

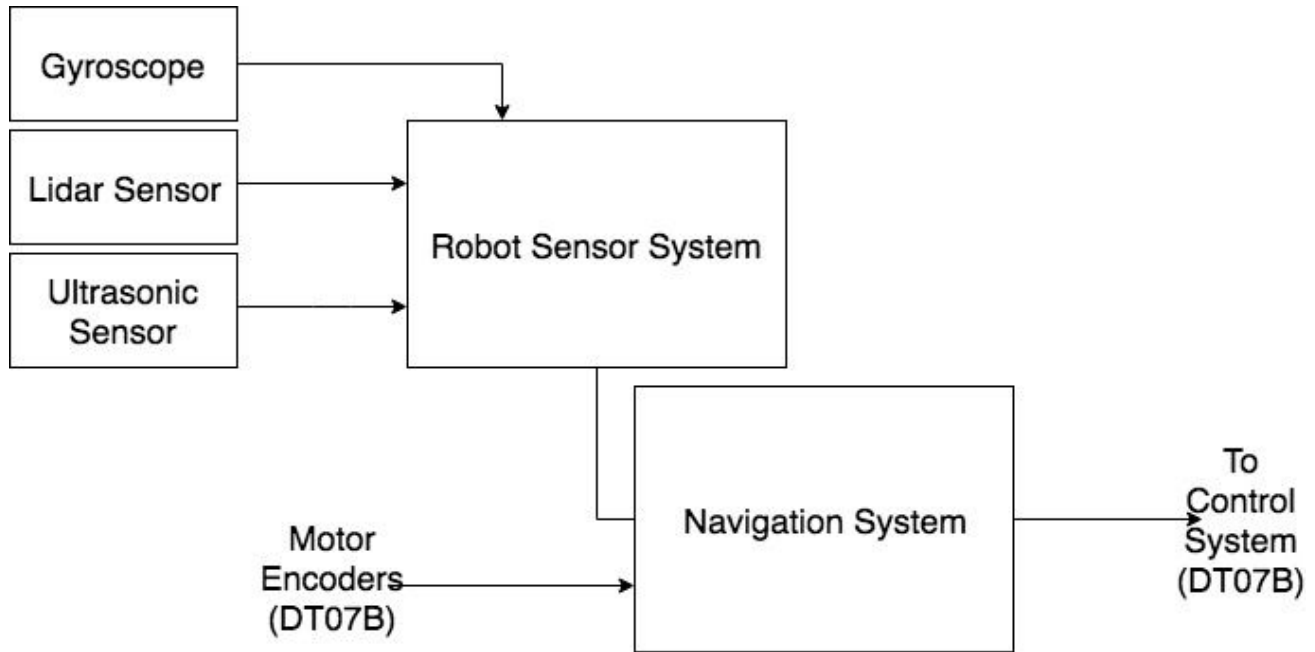


Figure 4: Level 1 Combat Robot Block Diagram.

[CH, HL]

The level 1 functional requirement table, indicating the second-level inputs and outputs of the fully autonomous combat robot, as shown in tables 4 and 5 below.

Table 4: Hardware Level 1 Sensor Array Functional Requirement Table.

Module	Robot Sensor System
Inputs	<ul style="list-style-type: none"> ● Sensor Input Signals
Outputs	<ul style="list-style-type: none"> ● Proximity and Orientation data
Functionality	Senses the location of the opposing robot and nearby walls. Outputs this information to the Navigation Center.

[HL]

Table 5: Hardware Level 1 Robot Control Center Functional Requirement Table.

Module	Robot Navigation Center
Inputs	<ul style="list-style-type: none"> ● Proximity and Orientation data ● Orientation ● Encoder Feedback
Outputs	<ul style="list-style-type: none"> ● Recommended Position ● Autonomous Enable/Emergency Stop
Functionality	Processes data from sensor system and makes autonomous fight or flight decisions. Produce a recommended location and orientation signal to send to controller.

[FA, CH]

The second level hardware diagram for the autonomous section of the combat robot is shown below. Items the left of the green dotted line are components of design team 7Bs system in which we will interact with.

3.1.3 Hardware Level 2 Block Diagram with Functional Requirement Table

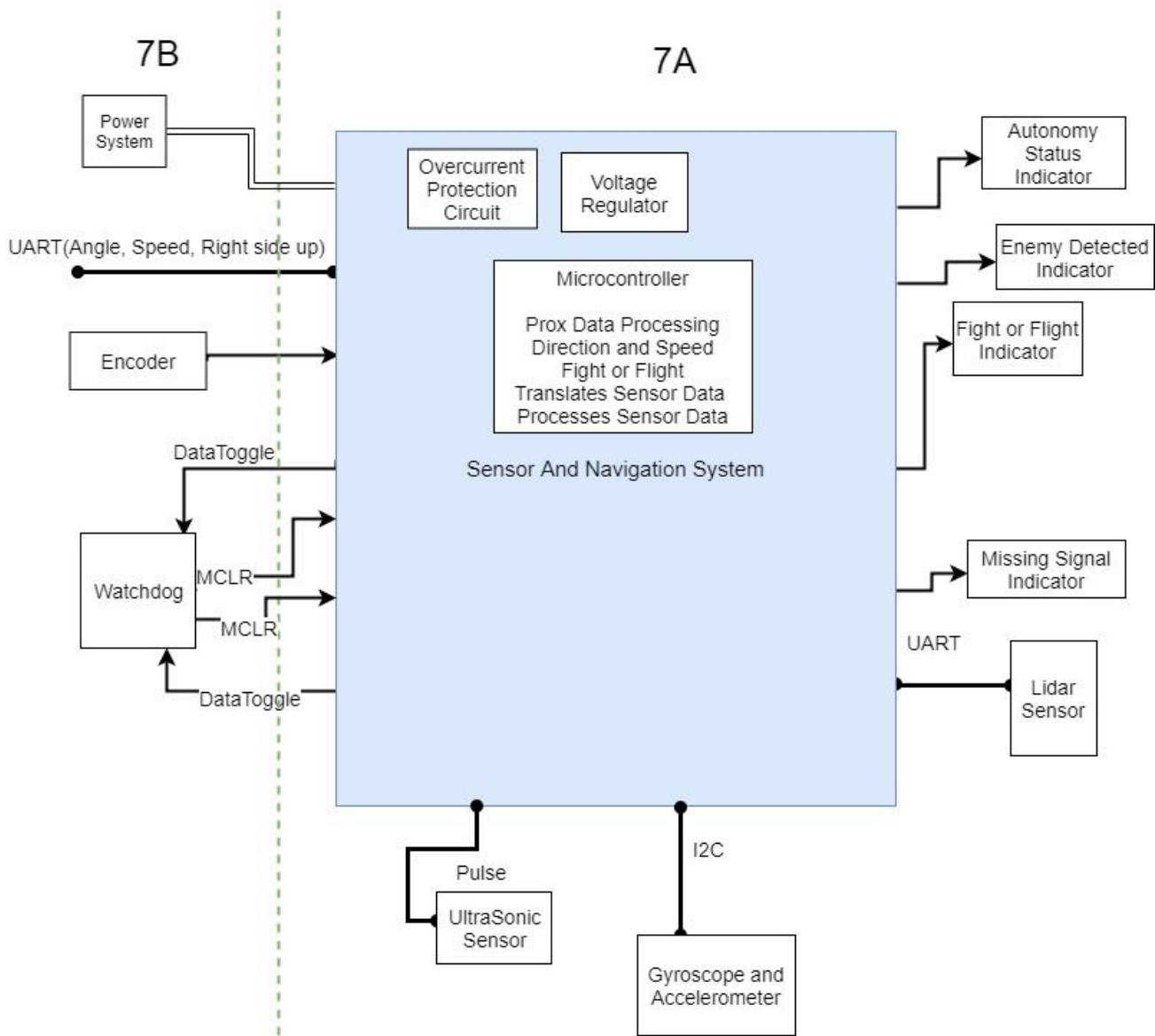


Figure 5 - Hardware Level 2 Block Diagram

Table 6: Ultrasonic Sensor Functional Requirement Table.

Module	Ultrasonic Sensor
Inputs	<ul style="list-style-type: none"> • Environmental data
Outputs	<ul style="list-style-type: none"> • Varying Pulse
Functionality	The ultrasonic sensor measures distances by sending an ultrasonic pulse and waiting for an echo. The sensor then returns a value that reflects the time between the emitted pulse and the echo.

[HL]

Table 7: Gyroscope Functional Requirement Table.

Module	Gyroscope and Accelerometer Sensor
Inputs	<ul style="list-style-type: none"> • Environmental data
Outputs	<ul style="list-style-type: none"> • I2C axial rotation and acceleration data
Functionality	The gyroscope and accelerometer will give data feedback on the robots motion containing G force information and angular rate information.

[CH]

Table 8: Lidar Sensor Functional Requirement Table.

Module	Lidar Sensor
Inputs	<ul style="list-style-type: none"> • Environmental data • PWM Motor Control Signal
Outputs	<ul style="list-style-type: none"> • Angle and distance over UART
Functionality	The Lidar sensor measures distances using a laser and measuring the reflection. It uses the reflection to detects object and output an angle and distance of their location.

[HL]

Table 9: Sensor System Functional Requirement Table.

Module	Sensor System Microcontroller
Inputs	<ul style="list-style-type: none"> •
Outputs	<ul style="list-style-type: none"> • Enemy Position and Wall Position: UART (To the Robot Navigation Center), • Gyroscope and Accelerometer data in UART • Ultrasonic sensor data
Functionality	

[CH]

Table 10: Overcurrent Protectors Functional Requirement Table.

Module	Overcurrent Protectors (One for each board)
Inputs	<ul style="list-style-type: none"> • Voltage from battery
Outputs	<ul style="list-style-type: none"> • Separate power supply for the navigation and sensor boards
Functionality	The overcurrent protectors break the circuit if the boards draw more than their allotted current.

[HL]

Table 11: Level 2 Voltage Regulators Functional Requirement Table.

Module	3.3 V And 5 V Voltage Regulators
Inputs	<ul style="list-style-type: none"> • 24 volt input
Outputs	<ul style="list-style-type: none"> • 3.3 volt and 5 volt outputs
Functionality	The voltage regulators take a 24 volt input and regulate it down to 3.3 v and 5 v to power the sensors and microcontrollers.

[HL]

Table 12: Level 2 LED Lights Functional Requirement Table.

Module	LED Lights
Inputs	<ul style="list-style-type: none"> ● Microprocessor outputs
Outputs	<ul style="list-style-type: none"> ● LED lights
Functionality	The LED lights are used to visually indicate the autonomy status, enemy detection, fight or flight and missing signal.

[HL]

Table 13: Sensor System Functional Requirement Table.

Module	Navigation/Sensor System Microcontroller
Inputs	<ul style="list-style-type: none"> ● Sensor Input Signal: UART from Lidar, Pulse for Ultrasonic, and I2C Gyroscope ● Motor Encoder Input Signal
Outputs	<ul style="list-style-type: none"> ● Recommended angle in UART ● Recommended speed setting in UART ● If the robot is right side up (Manual override is needed if the robot is flipped)
Functionality	The navigation/sensor system microcontroller will process the proximity data from the sensors. It will differentiate between the enemy and walls. The microcontroller will determine if high or low speed should be used and a angle to travel.

[CH]

3.1.4 Hardware Level 3 Diagram (Schematic Design)

The schematic of the sensor and navigation board is shown below. The microprocessor will be programmed in C.

Analysis on the clock cycle level lets us conclude that all of the Sensor and Navigation systems processing could be completed on one PIC32MZEF processor due to its speed and storage. With the speed of the processor being 252MHz, the time required for each navigation loop is estimated below:

- Read from LiDAR (50ms)
- Process LiDAR data (<1ms, 100 cycles)
- Check gyroscope (<1ms, 100 cycles)
- Check ultrasonics (if necessary) (1ms - 18.5ms, time hi and time low)
- Check encoders (if necessary) (<1ms, 100 cycles)
- Make path decision (<1ms, 100 cycles)
- Update current path (<1ms, 100 cycles)

All of these times add up to less than 50ms, with the fast speed of the processor this allows us to complete all of the other processes while reading from the LiDAR. Further analysis has been done in calculations section **3.5.4**. Based on these calculations, the baud rate required of the sensors is at least 20 MHz, the processor will accommodate this because its equivalent speed of reading is 63MHz.

Analysis was also done on the system to choose the processor. For ease of communications

and programming the same processor will be used for both design team 07A and 07B. Therefore, the processor chosen must be capable of performing the necessary tasks for each team independently.

The full combat robot system of both teams requires 3 UART modules (to communicate between the boards and to the LiDAR), a I2C module (to communicate to the accelerometer/gyroscope), 8 PWM capture compare modules (for the RC receiver and to drive the LiDAR motion), preferable USB capable (for troubleshooting), and at least 102,400 bytes of RAM flash memory (to store the sensor data). This is because the lidar needs to store 3200 bytes and we would like to store at least 4 samples in order to plan a path. This is calculated from 800 bytes of distance data plus 800 bytes of angle data per scan, multiplied by two additional data arrays to store processed data which will enable the robot to locate the enemy. This results in 3200 bytes, or 25,600 bits.

Multiple processors were found to meet these requirements. PIC32MX795F512H-80V appeared that it could meet our requirements, but its processors speed is only 80 MHz and its memory is 128 KB. The PIC32MZ2048EFH064 was chosen because for only \$1.85 more the processor is faster, 252Mhz, and the memory is larger, 512 KB.

LED's 0 through 8 will be used as indicators. LED 0-4 will report if the functionality (if autonomy can or cannot be used), if a enemy is detected, if none of the sensors are giving meaningful feedback, and if the system is in fight or flight mode. LED 9 and 10 will be used to indicate if our board has power.

The accelerometer outputs data in I2C to the processor. This will be interpreted and used to determine how the robot will need to move its motors to navigate. If the robot is upside down the motor directions will need flipped. The LiDAR, which communicates in UART, will be used to determine the enemies location. The ultrasonic sensor, which returns a hi to low time ratio, will be

used as a redundancy to determine the enemies location. The signal from the ultrasonic sensor will be interpreted using the input compare module and a timer to calculate the ratio between hi and low time. This will let us determine the distance measured to the detected object.

The sensor and navigation board will communicate to The Motion and Actualization Team in UART. The RX of our system will connect the the TX of there system and vice versa. An external oscillator will be used on the processor due to the sensitivity to UART to timing and the inaccuracy of internal oscillator in microcontrollers.

The Motion and Actualization Team will be providing power to the system. The power to our Sensor/Navigation board will be fully isolated from The Motion and Actualization Team's system. There is DC/DC isolation on the power source and isolation on the signals between the 2 boards. A backplane/card edge system will be used to interface with the hardware of The Motion and Actualization Team systems.

The sensors connections will be broken out directly to each sensor for testing purposes. In the final system they will be then routed through the backplane of 7B's system to the sensors to decrease noise and save space from less wiring needed.

All motion will cease within a minute after the emergency stop is pressed on The Motion and Actualization Team's system. This was tested by powering the lidar down while at full spin. It took 2.5 seconds for the lidar to stop rotating when powered down at a full speed spin. A USB interface has been added to the board for the purpose of ease of reading date from the sensor/navigation microcontroller. Unused pins are labeled with their communication protocol peripheral or other functionality for ease of addition of other hardware to the sensor and navigation schematic.

The minimum schematic for the microcontroller is shown below is shown below. The

decoupling capacitors are implemented in the circuit schematic that was designed. The suggested value of 0.1uF is used. This operates as a low pass filter. The trace has around 0.1 ohms, this results in a filter with a corner frequency of 16 MHz.

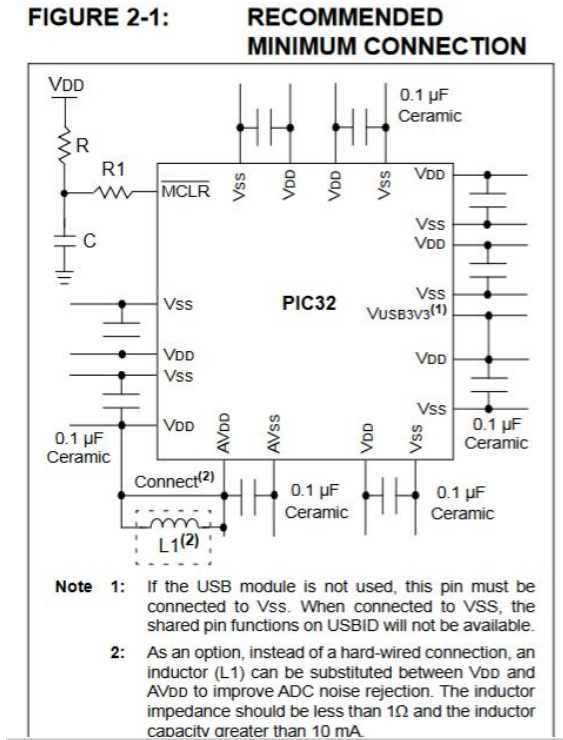


Figure 6: Minimum Hardware Schematic

[CH]

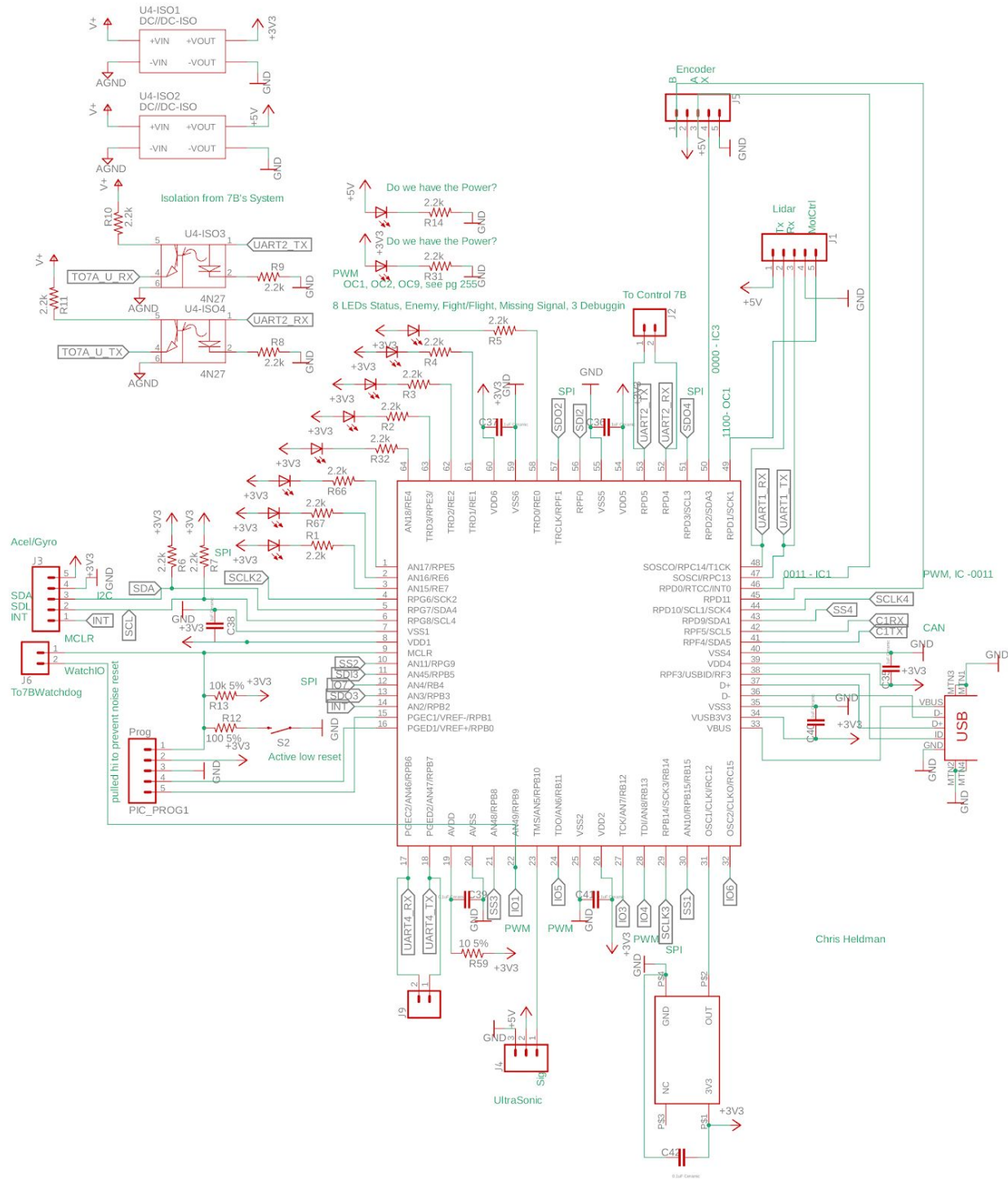
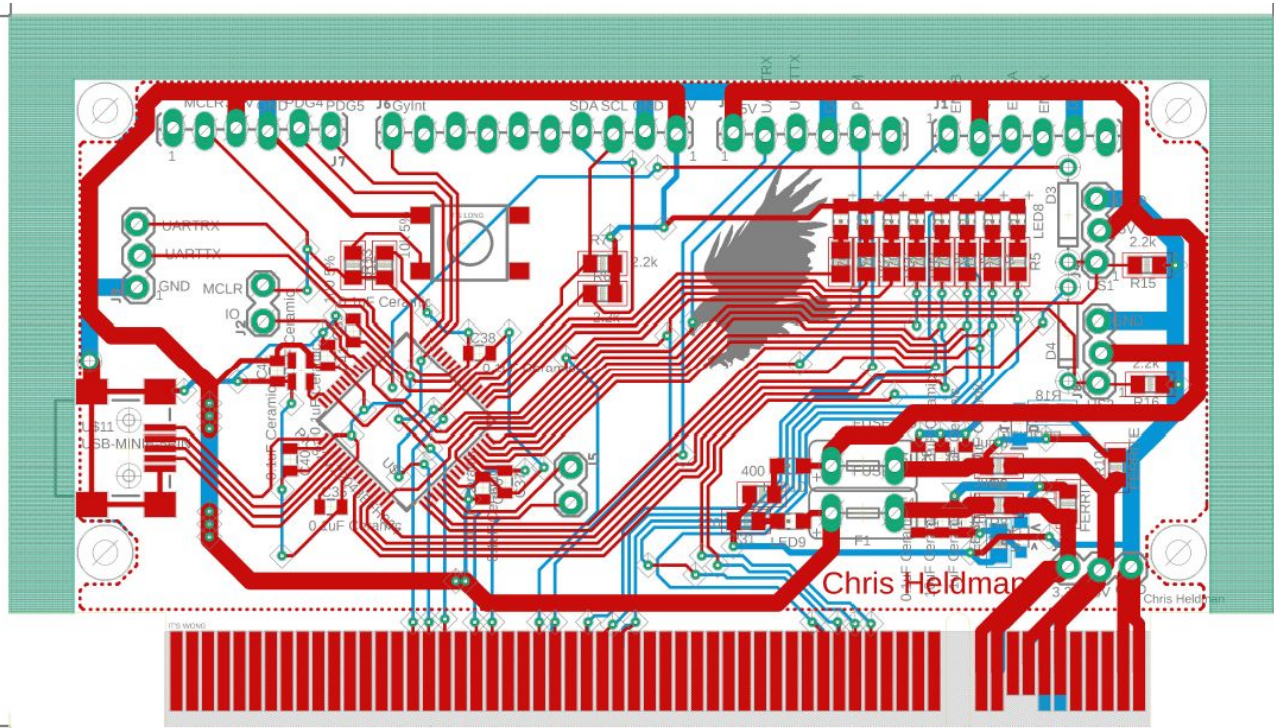


Figure 7: Schematic Diagram of the Sensor And Navigation System

[CH]

The board file is shown below for the sensor and navigation board.



Some specific notes about this design are, the current through the indicator LEDs are set to 15mA to light the leds bright enough to be seen across the room. This was calculated with the equation below.

$$R = \frac{V_S - V_{LED}}{I_{LED}} = \frac{3.3 - 2}{0.015} = 86.67 \approx 90 \text{ Ohms}$$

$$R = \frac{V_S - V_{LED}}{I_{LED}} = \frac{5 - 2}{0.005} = 395 \approx 400 \text{ Ohms}$$

The current for the pullup of the master clear was calculated to be 0.33mA, this is a high enough current for the CMOS technology of our microcontroller.

3.1.5 Hardware Simulation

There was a concern that the optical isolation would add too much delay to the uart communication. Because of this a simulation was performed on the optical isolation in order to optimize the circuit for low delay. This simulation is shown below. R1 sets the forward current of

the diode to 60mA as specified from the datasheet. A 2.2k standard pull up is used for the signal. R2 sets the sensitivity of the phototransistor. The capacitor is added to prevent unwanted turn ons due to noise from outside light or electromagnetic interference.

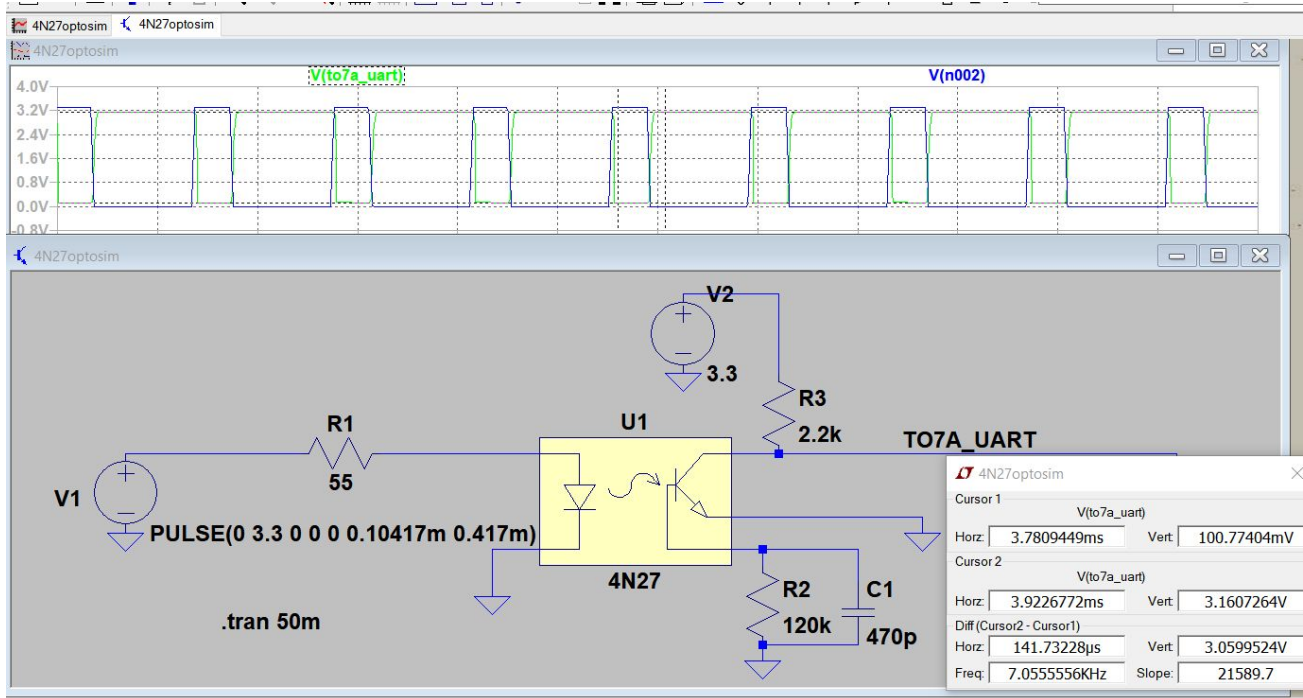


Figure 8: Optoisolator simulation

The design implemented on the eagle schematic is shown below. GND is our isolated systems ground and AGND is the ground from the robot power system.

[CH]

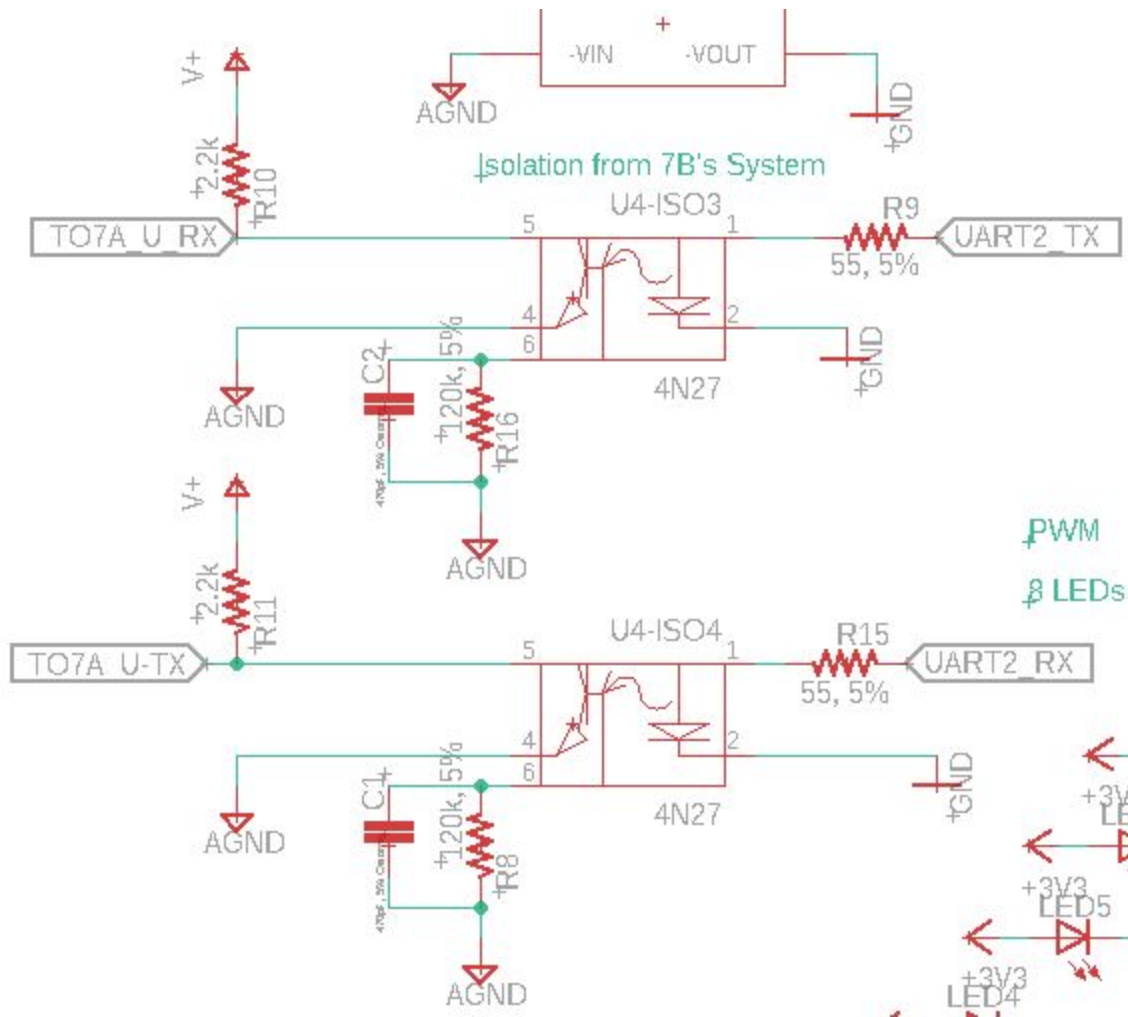


Figure 9: Optical Isolator Design

The Harmony Configurator system add on to MPLABX is a configuration system created by microchip in order to program the PIC32 series processors. Microchip has limited support for programming these series of processors using the classical coding method of configurations words and bitwise setting of configuration registers. To solve this problem Microchip created the Harmony Configuration system, which creates the configuration files for the PIC32 processors. The Harmony graphical user interface for the PIC32MZ2048EFH064 is show below in figure 10. This GUI allows the user to configure the pins with the desired internal peripherals.

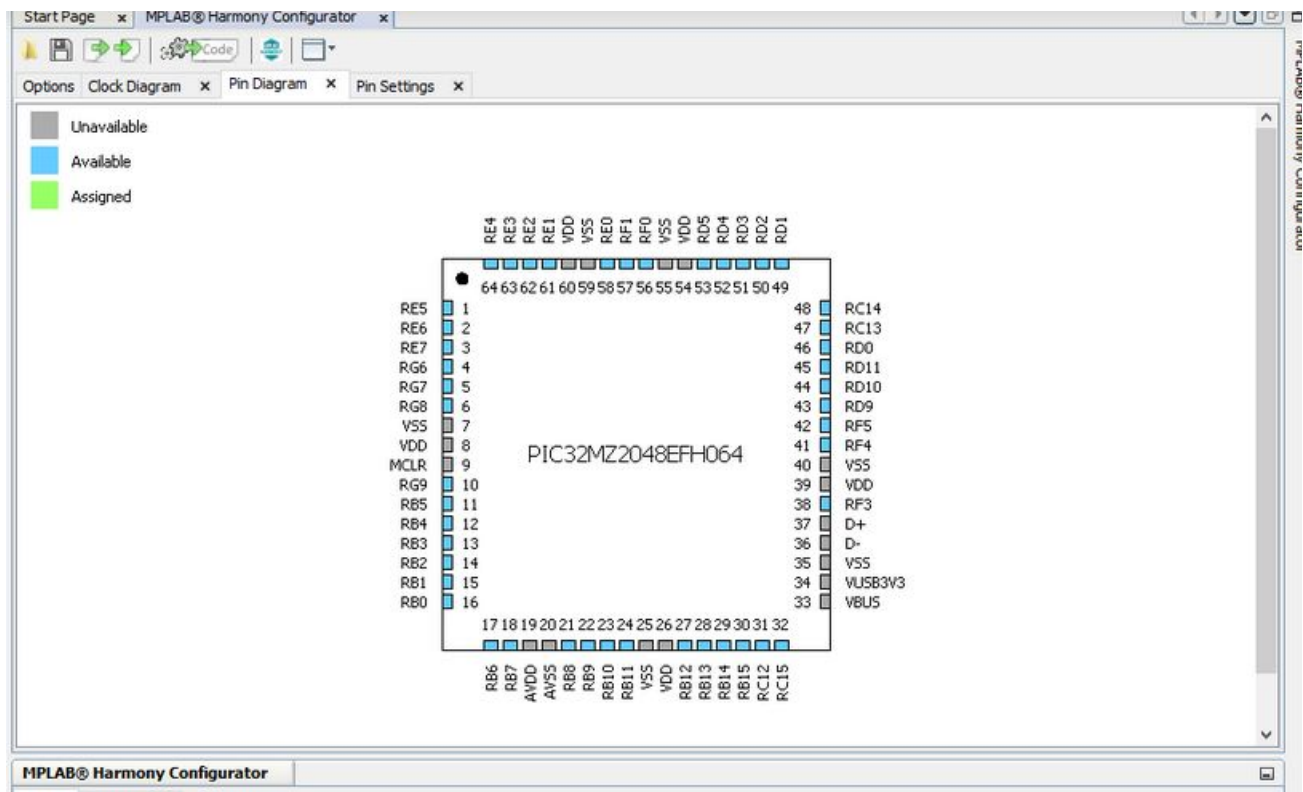


Figure 10: Harmony GUI for PIC32MZ2048EFH064

3.1.6 Hardware Testbench

In order to begin Hardware test benching a 100 pin PIM with the PIC32MZEF family microcontroller. This PIM was plugged into the Explorer 16/32 board. The processor then was configured using the Harmony Configurator shown below in Figure 11.

[CH]

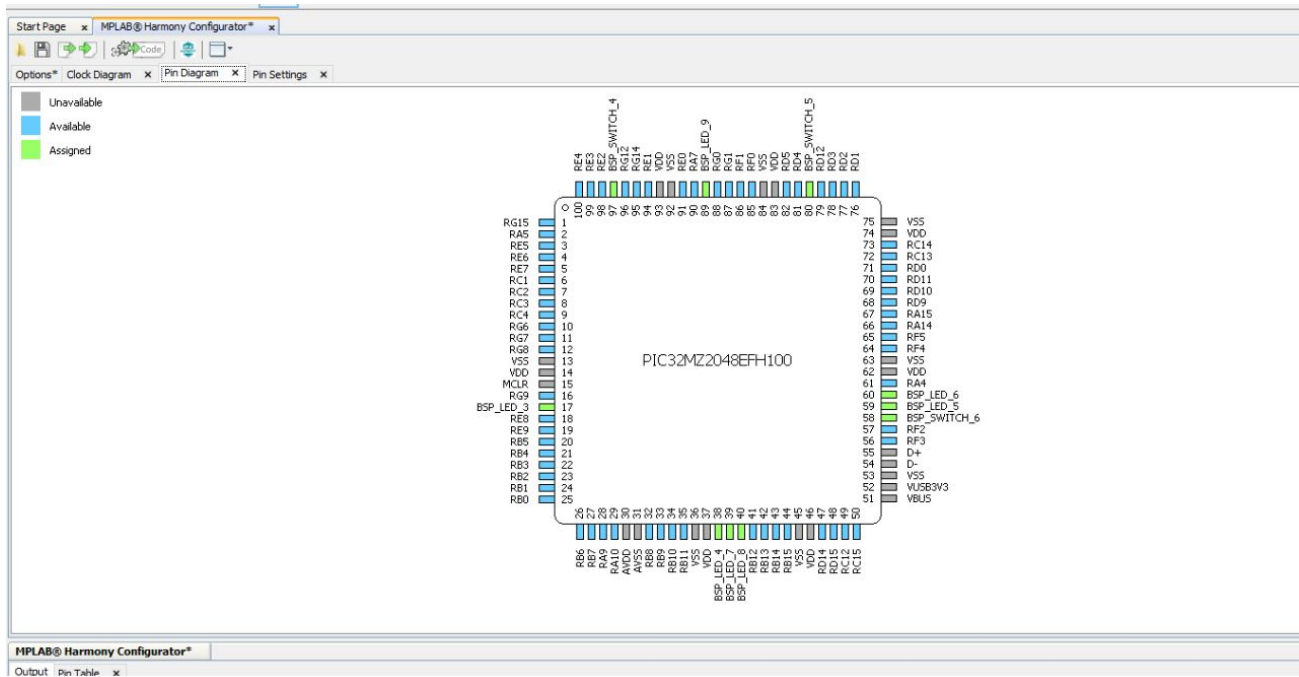


Figure 11: 100 pin PIM for PIC32MZEF

The microcontroller and its configuration using Harmony was tested by blinking a LED. This was successful. The next task was to begin testing with the sensors. Configuration with the LiDAR was determined.

In order to spin the lidar, it was necessary to generate a 25kHz PWM signal using the PIC24MZ PIM on an Explorer 16/32 board. To accomplish this, a timer was set up using a 16 prescaler on the 252MHz clock. This gave a timer frequency of $252\text{MHz}/16=15\text{MHz}$. This timer

was then used to set up a PWM with a period of 600 timer counts which is equivalent to a period of 25kHz. An output compare pulse width of 360 timer counts yields a 60% duty cycle which is nominal for 10Hz rotation of the RPLiDAR A3. Figure 12 shows the PWM output of the Explorer 16/32 board.

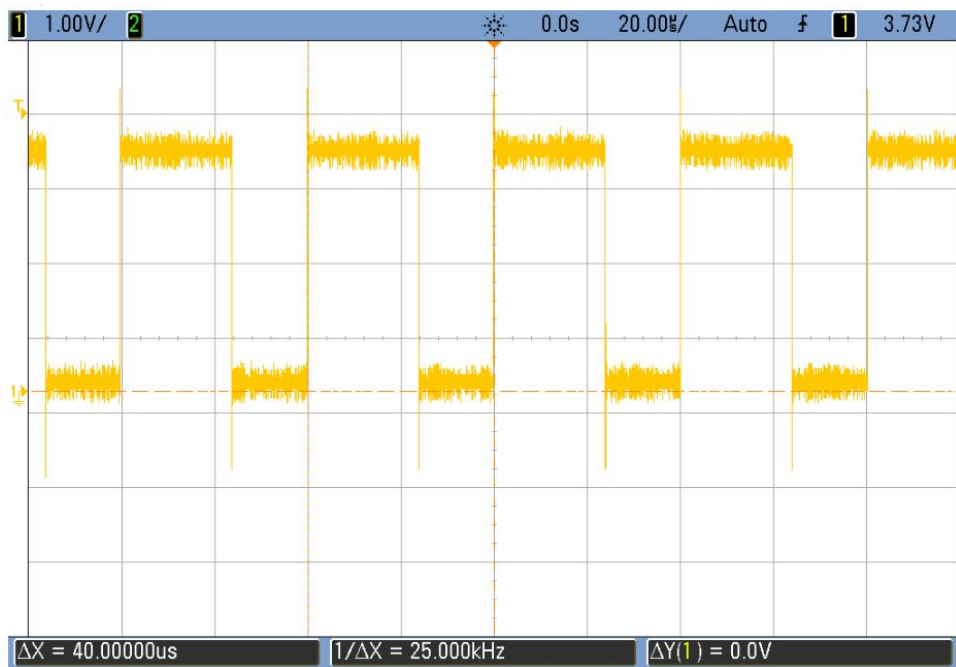


Figure 12: 60% duty cycle PWM output, 25kHz

[SV, HL, CH]

3.1.7 Software Level 0 Block Diagram and Functional Requirements Table

The software system is described in the preceding sections. The zero level block diagram is shown below.

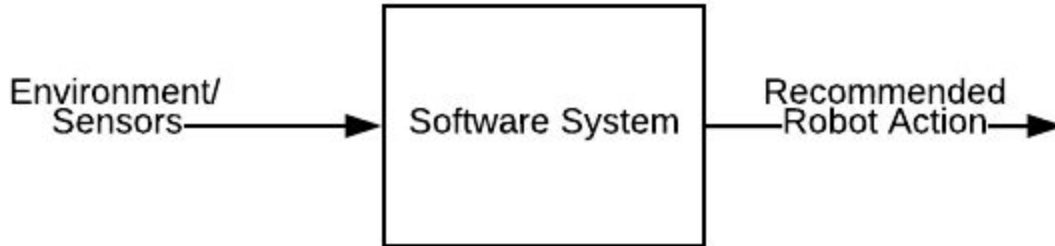


Figure 13: Software Level 0 Block Diagram

[FA]

Table 14: Software System Functional Requirement Table

Module	Software System
Inputs	<ul style="list-style-type: none"> • Environment
Outputs	<ul style="list-style-type: none"> • Recommended Robot Action
Functionality	The software system must collect data from the environment and use that data to determine a course of action for the robot. A course of action will be a direction and speed for the robot to travel and will constitute a fight or flight command.

[FA, SV]

3.1.8 Software Level 1 Block Diagram and Functional Requirement Tables

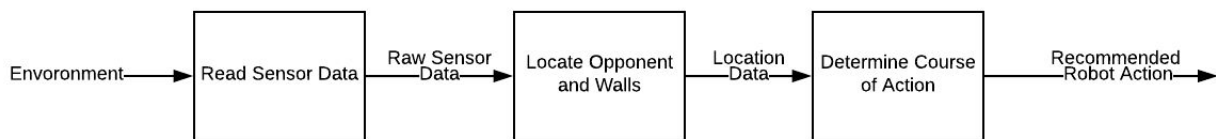


Figure 14: Software Level 1 Block Diagram

[SV]

Table 15: Read Sensor Functional Requirement Table

Module	Read Sensor Data
Inputs	<ul style="list-style-type: none"> • Environment
Outputs	<ul style="list-style-type: none"> • Raw Sensor Data
Functionality	The software must first collect raw data from all of the sensors monitoring the environment around the robot

[SV]

Table 16: Locate Opponent and Walls Functional Requirement Table

Module	Locate Opponent and Walls
Inputs	<ul style="list-style-type: none"> • Raw Sensor Data
Outputs	<ul style="list-style-type: none"> • Location Data
Functionality	The software must process the raw sensor data to differentiate the opponent from the arena walls and thereby locate the opponent. The location of the walls are also determined here.

[SV]

Table 17: Determine Course of Action Functional Requirement Table

Module	Determine Course of Action
Inputs	<ul style="list-style-type: none"> • Location Data
Outputs	<ul style="list-style-type: none"> • Recommended Robot Action
Functionality	The software must make a fight or flight decision based on the location of the enemy and the walls in the arena. The software will output where it thinks the robot should move.

[SV]

3.1.9 Level 2 Software Block Diagram

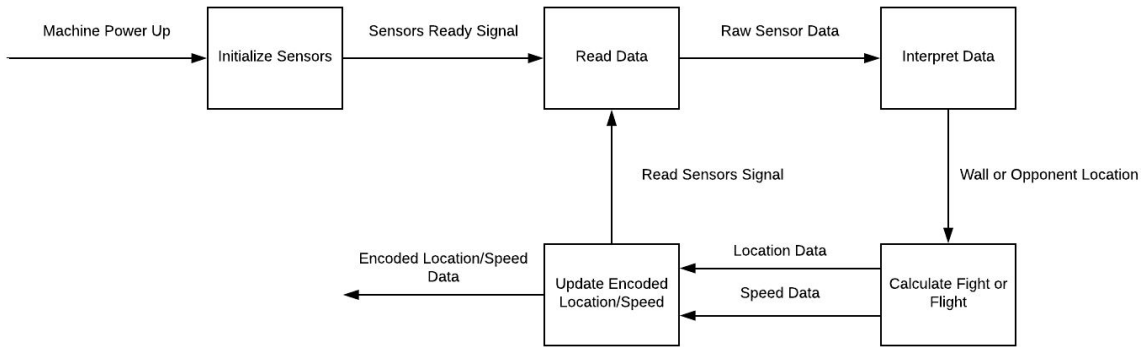


Figure 15: Software Level 2 Block Diagram

[FA, SV]

Table 18: Initialize Sensors Requirement Table

Module	Initialize Sensors
Inputs	<ul style="list-style-type: none"> Machine Power Up
Outputs	<ul style="list-style-type: none"> Sensors Ready Signal
Functionality	When the machine is powered up, start up the LiDAR and ultrasonic sensors. Proceed once data can be reliably read from both.

[FA, SV]

Table 19: Read Data Functional Requirement Table

Module	Read Data
Inputs	<ul style="list-style-type: none"> Sensors Ready Signal Read Sensors Signal
Outputs	<ul style="list-style-type: none"> Raw Sensor Data
Functionality	Reads data from the LiDAR and ultrasonic sensors, first when the sensors are started and again every time the current data has been processed. Returns raw data from the sensors.

[FA, SV]

Table 20: Interpret Data Functional Requirement Table

Module	Interpret Data
Inputs	<ul style="list-style-type: none"> ● Raw Sensor Data
Outputs	<ul style="list-style-type: none"> ● Wall or Opponent Location
Functionality	Checks the LiDAR data for anomalies and analyzes their shape to determine if it is looking at a wall or an opponent. For each anomaly, the data is sent to be processed for decision making.

[FA, SV]

Table 21: Calculate Fight or Flight Functional Requirement Table

Module	Calculate Fight or Flight
Inputs	<ul style="list-style-type: none"> ● Wall or Opponent Location
Outputs	<ul style="list-style-type: none"> ● Location Data ● Speed Data
Functionality	If an opponent is detected, makes decision to either run towards or away from it. If a wall is detected, makes decision to turn away from it. If nothing is detected, makes decision to turn until something is detected.

[FA]

Table 22: Update Encoded Location/Speed Functional Requirement Table

Module	Update Encoded Location/Speed
Inputs	<ul style="list-style-type: none"> ● Location Data ● Speed Data
Outputs	<ul style="list-style-type: none"> ● Encoded Location/Speed Data ● Read Sensors Signal
Functionality	Convert the location and speed decision to a UART data packet to be read by Team B. Read updated data from the sensors.

[FA]

3.1.10 Level 3 Software Block Diagram

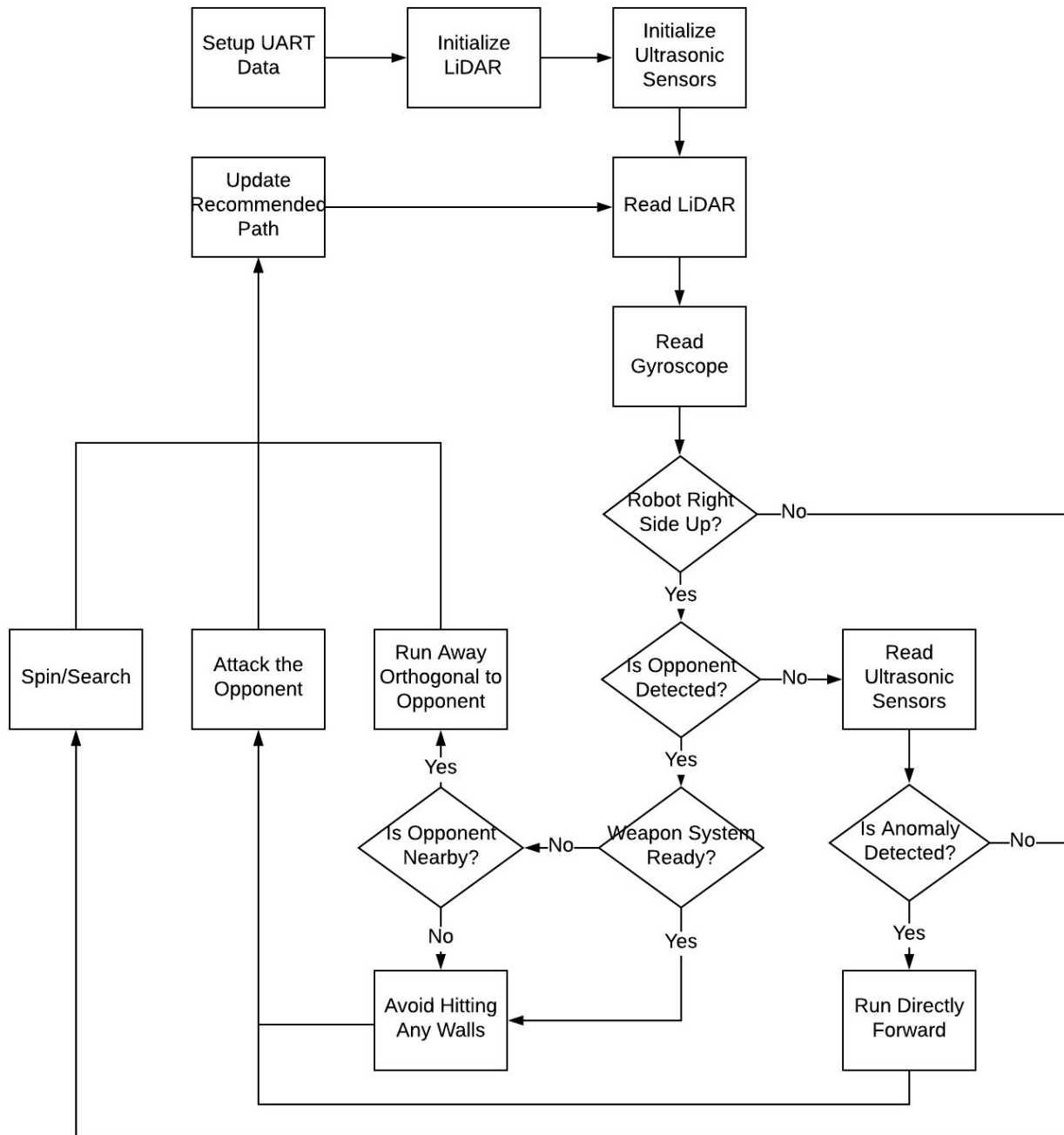


Figure 16: Level 3 Software Block Diagram

[FA]

3.2.0 Sensor Block Diagrams and Software

3.2.1 Main Program Software

main.c: controls the operation of the program

```
#include <stddef.h>                // Defines NULL
#include <stdbool.h>               // Defines true
#include <stdlib.h>                // Defines EXIT_FAILURE
#include "system/common/sys_module.h" // SYS function prototypes

int AssertFlag;

int main ( void )
{
    /* Initialize all MPLAB Harmony modules, including application(s). */
    // Start: //goto jump for timer done
    SYS_Initialize ( NULL );

    while ( true )
    {
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks ( );
    }

    /* Execution should not come here during normal operation */
    return ( EXIT_FAILURE );
}
```

app.h: header file for UART communication

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

```

#include <stdlib.h>
#include "system_config.h"
#include "system_definitions.h"
#include "system_config/default/system_config.h"

#include "autonomy.h"
#include "lidar.h"
#include "led.h"
#include "ultrasonic.h"
#include "gyroscope.h"
#include "timers.h"

typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_SERVICE_TASKS,

    /* TODO: Define states used by the application state machine. */

} APP_STATES;

typedef struct
{
    /* The application's current state */
    APP_STATES state;

    /* TODO: Define any additional data used by the application. */

} APP_DATA;

void APP_Initialize ( void );

void APP_Tasks( void );

```

app.c: source file for UART communication

```
#include "app.h"

APP_DATA appData;
lidarData lData;
short uartAngle;
short uartDrive;
DRV_HANDLE teraHandle;
DRV_HANDLE teraHandle1;
DRV_USART_TRANSFER_STATUS status;
DRV_USART_TRANSFER_STATUS status1;
size_t count;
size_t count1;

char send[100];
short sendB[2] = {1,0};
char receive[100];
timers_t ms100;
int sample = 0;
void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */

    SYS_INT_Enable();
    DRV_IC0_Start();
    DRV_IC1_Start();
    DRV_IC2_Start();
    DRV_TMR0_Start();
    DRV_TMR1_Start();
    DRV_TMR2_Start();
    DRV_OC0_Start();
}
```

```

DRV_OC1_Start();
DRV_OC2_Start();
setTimerInterval(&ms100, 256);
ledAllOn();
delay(1000);
DRV_USART_Initialize;
ledAllOff();
initLidar();
initUltras();
initGyro();

appData.state = APP_STATE_INIT;

}

void APP_Tasks ( void )
{

/* Check the application's current state. */
switch ( appData.state )
{
/* Application's initial state. */
case APP_STATE_INIT:
{
bool appInitialized = true;

if (appInitialized)
{
appData.state = APP_STATE_SERVICE_TASKS;
}
break;
}
}
}

```

```

case APP_STATE_SERVICE_TASKS:
{

    // while (1) {
        int counter = millis();

        //send[0] = readAngleX();

        //Gets information about opponent and walls via LiDAR

        statusOn();
        //LED_debug1On();
        lData = getLidarData();
        statusOff();
        //LED_debug1Off();

        //Autonomous path decision

        path dir = getPath(lData);

        if (dir.angle == 0) {
            uartAngle = 183;
        } else {
            uartAngle = dir.angle;
        }
        uartDrive = dir.drive;

        //BSP_Initialize();

        // LED_debug2On();
        int num1=getUltrasData(0);
        int num2=getUltrasData(1);

```



```

//LED_debug20ff();
//LED_debug30n();
int num3=976562.5*0.5/(Get_Encoder_Data ());
//LED_debug30ff();
//LED_debug40n();
int num4 = readAngleX();
//LED_debug40ff();
//int num4 = 0;
//lData = getLidarData();
int num5 = lData.opponentDistance;
int num6 = lData.opponentAngle;
sendB[0] = dir.angle;

//int num7 = (int)dir.angle;
//int num8 = (int)dir.drive;
// PORTE = ~lData.opponentDistance>>5;

/*
if(sendB[0]>=179)
{
    sendB[0] = 1;
}
if(sample >= 10)
{
    sendB[0]+=1;
    sample = 0;
}

sample++;
* */
//sendB[0]=183;//dir.angle; //num7;
sendB[1]=dir.drive; //num8;

```

```

        sprintf(send,"Ultrasonic Right = %i Ultrasonic Left = %i Encoder = %i RPM Gyroscope =
%i Lidar = %i Counter = %i UARTB = %i %i\n\r",num1,num2,num3,num4,num6,counter,sendB[0],sendB[1]);

        teraHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_WRITE); //open tera
usart channel

        count = DRV_USART_Write(teraHandle, send, strlen(send)); //write to tera
DRV_USART_Close(teraHandle); //discard tera handle

        /*
        teraHandle=DRV_USART_Open(DRV_USART_INDEX_0,DRV_IO_INTENT_READWRITE);
        count = DRV_USART_Read(teraHandle, receive, 1); //write to tera
        DRV_USART_Close(teraHandle); //discard tera handle

        // Encodes UART

        // Software patch for errors caused by sending 0 over UART
        */

        teraHandle1 = DRV_USART_Open(DRV_USART_INDEX_2, DRV_IO_INTENT_WRITE); //open tera
usart channel

        count1 = DRV_USART_Write(teraHandle1, sendB, 4); //write to tera
DRV_USART_Close(teraHandle1); //discard tera handle

        //ledAll0ff();

    // }

    break;

}

/* TODO: implement your application state machine.*/

/* The default state should never be executed. */
default:
{
    SYS_RESET_SoftwareReset();

    /* TODO: Handle error in application's state machine. */
    break;
}

```

```
    }  
  }  
}
```

autonomy.h: header file for fight or flight decision

```
#include "lidar.h"  
#include "system_config.h"  
#include <stdint.h>  
#include <stdbool.h>  
#include "system_config/default/system_config.h"  
  
typedef struct {  
    short angle;  
    short drive;  
} path;  
  
path getPath(lidarData lData);
```

autonomy.c: source file for fight or flight decision

```
#include "app.h"  
#include "autonomy.h"  
#include "gyroscope.h"  
#include "ultrasonic.h"  
#include "encoder.h"  
#include "led.h"  
#include "system_config/default/system_config.h"  
#include "system_config.h"
```

```

// Keeps track of where to search if LiDAR goes missing
unsigned short lastKnownAngle;
unsigned short lastKnownDistance;
unsigned short forwardCounter;

path getPath(lidarData lData){

    path dir;

    // Sends designated signal to indicate upside down
    if (isUpsideDown() == true) {
        dir.angle = 183; //Manual
        dir.drive = 183;
        return dir;
    }

    ///Preprocessing LiDAR data
    // Converts angles 0-90 to 90-180
    lData.opponentAngle += 90;
    // Converts angles 270-359 to 0-89
    if (lData.opponentAngle >= 360) lData.opponentAngle -= 360;
    // Constrains angles between 1-179 to avoid complications
    if (lData.opponentAngle == 0) lData.opponentAngle = 1;
    if (lData.opponentAngle >= 180) lData.opponentAngle = 179;

    // If the enemy could not be spotted with LiDAR
    if (lData.opponentDistance == 0) {
        forwardCounter += 1;
        //flightOff();
        // If the enemy is spotted by ultrasonics, go straight forward
        if (isUltraClose() ||
            ((lastKnownDistance < 1000) && (forwardCounter < 10) &&
            ((lastKnownAngle >= 80) || (lastKnownAngle <= 100)))) {

```

```

    enemyOn();
    missingOff();
    dir.angle = 90;
    dir.drive = 182; //Forward
    return dir;
} else { // If nothing is detected, go into search mode
    flightOn();
    enemyOff();
    missingOn();
    dir.drive = 181; //Search
    if (lastKnownAngle < 90) dir.angle = 1;
    else dir.angle = 179;
    flightOff();
}
} else {
    // If the enemy is found, go into fight or flight mode
    forwardCounter = 0;
    enemyOn();
    missingOff();
    // Fight logic
    if ((isWeaponReady()) || (lData.opponentDistance > 10000)) {
        flightOff();
        dir.angle = lData.opponentAngle;
        dir.drive = 182;
    // Flight logic
    } else {
        flightOn();
        dir.drive = 184;
        dir.angle = 90;
        // Make an 80 degree turn away from opponent
        //if (lData.opponentAngle < 90) {
        //    dir.angle = lData.opponentAngle + 80;
        //} else {

```

```

        //   dir.angle = lData.opponentAngle - 80;
        //}
    }
}
lastKnownAngle = dir.angle;
if (lData.opponentDistance != 0) {
    lastKnownDistance = lData.opponentDistance;
}
return dir;
}

```

timers.h: header file for internal clocks and timers

```

#include <stdbool.h>
#include <stdlib.h>

typedef struct {
    unsigned long timerInterval;
    unsigned long lastMillis;
} timers_t;

bool timerDone(timers_t * t);
void setTimerInterval(timers_t * t, unsigned long interval);
void resetTimer(timers_t * t);
void globalTimerTracker( );
unsigned long millis(void);
void delay(unsigned int val);
void isTimedOut(void);

```

timers.c: source file for internal clocks and timers

```

#include "timers.h"

unsigned long globalTime;
timers_t timeOut;

unsigned long millis(void)
{
    return globalTime;
}

bool timerDone(timers_t * t)
{
    if(abs(millis() - t->lastMillis) > t->timerInterval)
    {
        t->lastMillis=millis();
        return true;
    }
    else
    {
        return false;
    }
}

void setTimerInterval(timers_t * t, unsigned long interval)
{
    t->timerInterval= interval;
}

void resetTimer(timers_t * t)
{
    t->lastMillis=millis();
}

```

```

//Call this function in your timer interrupt that fires at 1ms
void globalTimerTracker( )
{
    globalTime++;
}

timers_t time;
void delay(unsigned int val)
{
    setTimerInterval(&time, val);
    int i;
    while(!timerDone(&time))
    {
        i++;
    }
}

// Triggers if the component fails to connect
void isTimedOut() {
    setTimerInterval(&timeOut, 100);
    while(!timerDone(&timeOut))
        missingOn();
    missingOff();
}

```

led.h: header file for enabling and disabling LEDs

```

void ledAllOff(void);
void ledAllOn(void);
void binaryOutput(unsigned short input);
void enemyOn(void);
void flightOn(void);

```



```
void missing0n(void);
void status0n(void);
void debug10n(void);
void debug20n(void);
void debug30n(void);
void debug40n(void);
```

led.c: source file for enabling and disabling LEDs

```
#include "led.h"
#include "app.h"
#include "system_config.h"
```

```
void ledAllOn(void)
{
    LED_EnemyOn();
    LED_FlightOn();
    LED_MissingOn();
    LED_StatusOn();
    LED_debug10n();
    LED_debug20n();
    LED_debug30n();
    LED_debug40n();
}
```

```
void ledAllOff(void)
{
    LED_EnemyOff();
    LED_FlightOff();
    LED_MissingOff();
    LED_StatusOff();
    LED_debug10ff();
}
```

```

    LED_debug20ff();
    LED_debug30ff();
    LED_debug40ff();
}
void binaryOutput(unsigned short input)
{
    if (input & 0b0001)
    {
        LED_debug10n();
    }
    if (input & 0b0010)
    {
        LED_debug20n();
    }
    if (input & 0b0100)
    {
        LED_debug30n();
    }
    if (input & 0b1000)
    {
        LED_debug40n();
    }
}
void enemyOn(void)
{
    LED_EnemyOn();
}
void enemyOff(void)
{
    LED_EnemyOff();
}
void flightOn(void)
{

```

```
    LED_FlightOn();
}
void flightOff(void)
{
    LED_FlightOff();
}
void missingOn(void)
{
    LED_MissingOn();
}
void missingOff(void)
{
    LED_MissingOff();
}
void statusOn()
{
    LED_StatusOn();
}
void statusOff()
{
    LED_StatusOff();
}
void debug10n(void)
{
    LED_debug10n();
}
void debug10ff(void)
{
    LED_debug10ff();
}
void debug20n(void)
{
    LED_debug20n();
```

```
}  
void debug20ff(void)  
{  
    LED_debug20ff();  
}  
void debug30n(void)  
{  
    LED_debug30n();  
}  
void debug30ff(void)  
{  
    LED_debug30ff();  
}  
void debug40n(void)  
{  
    LED_debug40n();  
}  
void debug40ff(void)  
{  
    LED_debug40ff();  
}
```

3.2.2 LiDAR Sensor

A LiDAR emits an infrared laser which bounces off of the nearest object to measure the distance to the nearest object. The robot will use a rotating LiDAR sensor to locate objects in the arena, specifically the enemy. The requirements for the LiDAR are listed in Table 23, and are based on the simulations detailed in 3.5.1 and 3.5.2 below

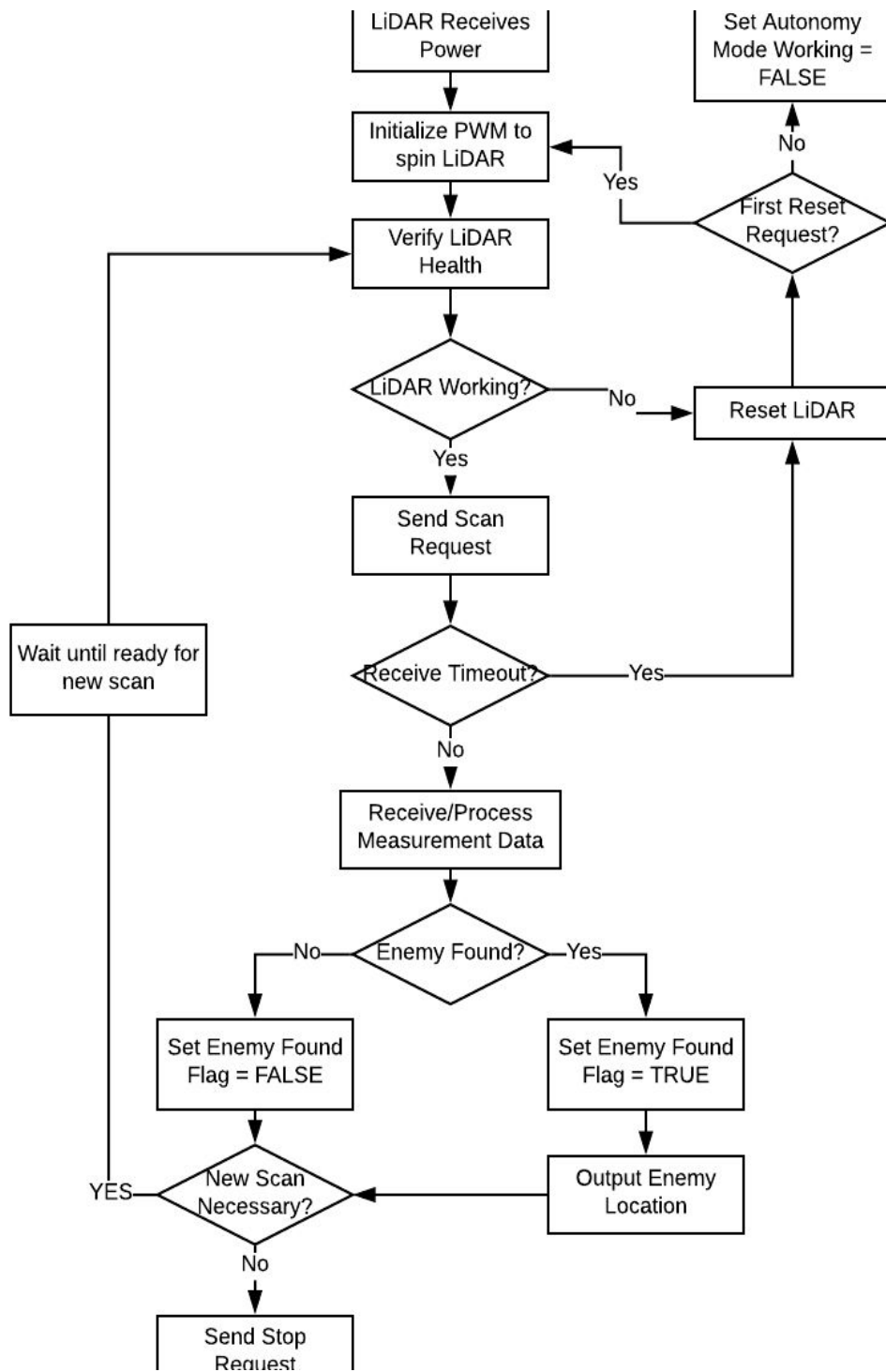
Table 23:LiDAR Sensor Requirements

	Desired	RPLiDAR A3
Maximum sensing distance	17m	20m
Minimum sensing distance	3m	1m
Sample Rate	8kHz	16kHz
Cost	As small as possible	\$600

Table 23 shows that the RPLiDAR A3 fulfills the basic requirements for resolution and sensing distance. It was selected because it was the cheapest available option that fulfilled the minimum requirements.

As the RPLiDAR A3 spins, it returns ordered pairs consisting of the distance to the nearest object, as well as the angle to that object. The robot will be programmed to process this data to search for anomalies consistent with an opponent in the arena. Figure 17 shows a flow chart representing the LiDAR interface process.

[SV]



[SV]

Figure 17: RPLiDAR Interface Flowchart

Below, the software is shown for the LiDAR sensor. The software shows the process for reading values and determining an opponent angle and distance.

lidar.h: header file for the LiDAR sensor

```
#include "system_config.h"
#include "system_config/default/system_config.h"

// Return the closest point of the opponent and the wall
// If none detected, return 0
typedef struct {
    unsigned short opponentAngle;
    unsigned short opponentDistance;
    unsigned short wallAngle;
    unsigned short wallDistance;
} lidarData;

void firstFilter(int sampleSpace);
void secondFilter();
void thirdFilter(int sampleSpace);
void isLidarTimedOut();
lidarData getLidarData(void);
void setLidarData(short oppAng, short oppDist, short wallAng, short wallDist);
void initLidar(void);
```

lidar.c: source file for the LiDAR sensor

```
#include "lidar.h"
#include "app.h"
#include "led.h"

// Example data
lidarData lidar = {0, 0, 0, 0};
```

```

//global variables
DRV_HANDLE lidarHandle;
DRV_HANDLE teraHandle2;

DRV_USART_TRANSFER_STATUS status;

timers_t timeOut;

unsigned char readBuffer[1852];
unsigned char writeBuffer[2];
size_t count_1, total;
int count1, corner_index, front_corner_index;
unsigned short extractAngle[369];
unsigned short extractDistance[369];
unsigned short firstDifference[369];
unsigned short secondDifference[5];
unsigned short outputAngleDistance[4] = {0,0,0,0};
unsigned short wallDistance = 0, wallAngle = 0;

unsigned short startArray = 0xa520;

int start = 0, found_A_Quality_Index = 0, n_samples_apart;

int sample_threshold[10] = {600, 600, 300, 400, 500, 600, 900, 1000, 1300, 600};
/*int sample_threshold_2 = 600;
int sample_threshold_3 = 300;
int sample_threshold_4 = 400;
int sample_threshold_5 = 500;
int sample_threshold_6 = 600;
int sample_threshold_7 = 900;
int sample_threshold_8 = 1000;
int sample_threshold_9 = 1300;

```



```

int sample_threshold_10 = 600;*/
int Found_Enemy = 0;

int j;
void firstFilter(int sampleSpace) {
    for(j = sampleSpace + 4; j < 360; j++)
    {
        //we suspect a zeroed value is why we returned zero
        if(extractDistance[j]>extractDistance[j-sampleSpace] && extractDistance[j-sampleSpace]!=0 )
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j-sampleSpace];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
            if(firstDifference[j]>sample_threshold[sampleSpace - 1] )
            {
                outputAngleDistance[0] = extractAngle[j];
                outputAngleDistance[1] = extractDistance[j-sampleSpace];
                corner_index = j;
                n_samples_apart = sampleSpace;
            }

        }

    }

}

void secondFilter(){
    secondDifference[0] = extractDistance[corner_index - n_samples_apart] -
extractDistance[corner_index - (n_samples_apart+1)];
    secondDifference[1] = extractDistance[corner_index - n_samples_apart] -
extractDistance[corner_index - (n_samples_apart+2)];
    secondDifference[2] = extractDistance[corner_index - n_samples_apart] -

```

```

extractDistance[corner_index - (n_samples_apart+3)];
    secondDifference[3] = extractDistance[corner_index - n_samples_apart] -
extractDistance[corner_index - (n_samples_apart+4)];
    secondDifference[4] = extractDistance[corner_index - n_samples_apart] -
extractDistance[corner_index - (n_samples_apart+5)];
    if(secondDifference[0]<50||secondDifference[0]>65435
        || secondDifference[1]<100||secondDifference[1]>65435
        || secondDifference[2]<100||secondDifference[2]>65435
        || secondDifference[3]<100||secondDifference[3]>65435
        || secondDifference[4]<100||secondDifference[4]>65435)
    {
        Found_Enemy = 1;
    }
}
/*
void thirdFilter(int sampleSpace){
    for(j = corner_index; j > sampleSpace + 1; j--)
    {
        if(extractDistance[j]>extractDistance[j + sampleSpace] &&
            extractDistance[j + sampleSpace] != 0)
        {

            firstDifference[j] = extractDistance[j] -
                extractDistance[j + sampleSpace];

            if(firstDifference[j]>sample_threshold[sampleSpace - 1])
            {
                outputAngleDistance[2] = extractAngle[j];
                outputAngleDistance[3] = extractDistance[j + sampleSpace];
                front_corner_index = j;
            }
        }
    }
}

```

```

    }
}
*/
// Triggers if the gyroscope fails to connect
void isLidarTimedOut() {
    size_t count_L;
    char send_L[100];
    DRV_HANDLE teraHandle_L;
    setTimerInterval(&timeOut,100);
    while(!timerDone(&timeOut)) {
        missingOn();
        sprintf(send_L,"Lidar Communication Timeout Inside While Loop");
        teraHandle_L = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_WRITE); //open tera usart
channel
        count_L = DRV_USART_Write(teraHandle_L, send_L, strlen(send_L)); //write to tera
        DRV_USART_Close(teraHandle_L); //discard tera handle
    }
    missingOff();
    sprintf(send_L,"Lidar Communication Timeout Outside While Loop");
    teraHandle_L = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_WRITE); //open tera usart channel
    count_L = DRV_USART_Write(teraHandle_L, send_L, strlen(send_L)); //write to tera
    DRV_USART_Close(teraHandle_L); //discard tera handle
}

// Temporary simulation until LiDAR is integrated
lidarData getLidarData(void) {

    int i,j,k,look,start_index;//,corner_index,front_corner_index;

    lidar.opponentAngle = 0;
    lidar.opponentDistance = 0;

    lidarHandle = DRV_USART_Open(DRV_USART_INDEX_1, DRV_IO_INTENT_READWRITE);

```

```

if (start == 0)
{
    start = 1;
    writeBuffer[0] = 0xA5;//scan request
    writeBuffer[1] = 0x20;

    count_1 = DRV_USART_Write(lidarHandle, writeBuffer, 2);

}

count_1 = DRV_USART_Read(lidarHandle, readBuffer, 1842);//store scan data
//this would be where we make sure count = 1842 to verify a good write

while(DRV_USART_TransferStatus(lidarHandle)==DRV_USART_TRANSFER_STATUS_RECEIVER_EMPTY);

DRV_USART_Close(lidarHandle);//discard lidar handle
//this for loop searches for a "quality type" byte by searching for even numbers
for(i=1;i<1840-95;i++)
{

    if(readBuffer[i]%2 == 0 && readBuffer[i+1]%2 == 1 &&
        readBuffer[i+5]%2 == 0 && readBuffer[i+6]%2 == 1 &&
        readBuffer[i+10]%2 == 0 && readBuffer[i+11]%2 == 1 &&
        readBuffer[i+15]%2 == 0 && readBuffer[i+16]%2 == 1 &&
        readBuffer[i+20]%2 == 0 && readBuffer[i+21]%2 == 1 &&
        readBuffer[i+25]%2 == 0 && readBuffer[i+26]%2 == 1 &&
        readBuffer[i+30]%2 == 0 && readBuffer[i+31]%2 == 1 &&
        readBuffer[i+35]%2 == 0 && readBuffer[i+36]%2 == 1 &&
        readBuffer[i+40]%2 == 0 && readBuffer[i+41]%2 == 1 &&
        readBuffer[i+45]%2 == 0 && readBuffer[i+46]%2 == 1 &&
        readBuffer[i+50]%2 == 0 && readBuffer[i+51]%2 == 1
    )

```

```

    {
        found_A_Quality_Index = i;
        break;
    }
}

start_index = (found_A_Quality_Index%5)+1;
j = 0;
k = 0;

//*****//
//*****//
//****          First Filter          ****//
//*****//
//*****//

//The first filter takes the difference between two neighboring samples. This
//difference is compared to a threshold value. If the difference is greater
//than the threshold value, an anomaly has been detected in the room which could
//be an enemy robot. The threshold value is adjusted based on the number of
//samples between the the two neighboring samples of interest.

//high certainty sweep. checks for enemies with strictest criteria
for(i=start_index;i<1840;i+=5)
{

    extractAngle[j] = ((readBuffer[i+1]<<8) | readBuffer[i])>>7;//combine 2 bytes of angle data
from lidar

    extractDistance[j] = ((readBuffer[i+3]<<8) | readBuffer[i+2])>>2;//combine 2 bytes of
distance data from lidar

    if(extractAngle[j]>90 && extractAngle[j] < 270)
    {

```

```

    extractDistance[j] = 0;
}

if(j>=8 && extractDistance[j] != 0)//ignore first four samples
{
    if(extractDistance[j]>extractDistance[j-4] && extractDistance[j-4]!=0 )
    {
        firstDifference[j] = extractDistance[j]-extractDistance[j-4];

        //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
        if(firstDifference[j]>sample_threshold[3])
        {
            outputAngleDistance[0] = extractAngle[j];
            outputAngleDistance[1] = extractDistance[j-4];
            corner_index = j;
            n_samples_apart = 4;
        }
    }
}

j++;
k+=2;

}

j = 0;
k = 0;

int sampleSpace = 3;
while(outputAngleDistance[1] == 0 && sampleSpace <= 9){
    firstFilter(sampleSpace);
    sampleSpace += 1;
}

```

```

/*
if(outputAngleDistance[1] == 0)
{
    for(j=9;j<360;j++)
    {
        if(extractDistance[j]>extractDistance[j-5] && extractDistance[j-5]!=0 )//we suspect a
zeroed value is why we returned zero
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j-5];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
            if(firstDifference[j]>sample_threshold_5 )
            {
                outputAngleDistance[0] = extractAngle[j];
                outputAngleDistance[1] = extractDistance[j-5];
                corner_index = j;
                n_samples_apart = 5;
            }

        }

    }

}

if(outputAngleDistance[1] == 0)
{
    for(j=9;j<360;j++)
    {
        if(extractDistance[j]>extractDistance[j-3] && extractDistance[j-3]!=0 )//we suspect a
zeroed value is why we returned zero
        {

```

```

firstDifference[j] = extractDistance[j]-extractDistance[j-3];

//secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
if(firstDifference[j]>sample_threshold_3 )
{
    outputAngleDistance[0] = extractAngle[j];
    outputAngleDistance[1] = extractDistance[j-3];
    corner_index = j;
    n_samples_apart = 3;
}

}

}

}

if(outputAngleDistance[1] == 0)
{
    for(j=10;j<360;j++)
    {
        if(extractDistance[j]>extractDistance[j-6] && extractDistance[j-6]!=0)//we suspect a
zeroed value is why we returned zero
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j-6];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
            if(firstDifference[j]>sample_threshold_6 )
            {
                outputAngleDistance[0] = extractAngle[j];
                outputAngleDistance[1] = extractDistance[j-6];
                corner_index = j;
                n_samples_apart = 6;
            }
        }
    }
}

```



```

        }

    }

}

if(outputAngleDistance[1] == 0)
{
    for(j=11;j<360;j++)
    {
        if(extractDistance[j]>extractDistance[j-7] && extractDistance[j-7]!=0 )//we suspect a
zeroed value is why we returned zero
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j-7];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
            if(firstDifference[j]>sample_threshold_7 )
            {
                outputAngleDistance[0] = extractAngle[j];
                outputAngleDistance[1] = extractDistance[j-7];
                corner_index = j;
                n_samples_apart = 7;
            }

        }

    }

}

```

```

if(outputAngleDistance[1] == 0)
{
    for(j=12;j<360;j++)
    {
        if(extractDistance[j]>extractDistance[j-8] && extractDistance[j-8]!=0 )//we suspect a
zeroed value is why we returned zero
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j-8];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
            if(firstDifference[j]>sample_threshold_8 )
            {
                outputAngleDistance[0] = extractAngle[j];
                outputAngleDistance[1] = extractDistance[j-8];
                corner_index = j;
                n_samples_apart = 8;
            }

        }

    }

}

//close enemy sweep. values for lidar tend to return more zero when very close. this sweep checks
for this scenario
if(outputAngleDistance[1] == 0)
{
    for(j=13;j<360;j++)
    {
        if(extractDistance[j]>extractDistance[j-9] && extractDistance[j-9]!=0 )//we suspect a
zeroed value is why we returned zero

```

```

    {

        firstDifference[j] = extractDistance[j]-extractDistance[j-9];

        //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
        if(firstDifference[j]>sample_threshold[8] )
        {
            outputAngleDistance[0] = extractAngle[j];
            outputAngleDistance[1] = extractDistance[j-9];
            corner_index = j;
            n_samples_apart = 9;
        }
    }

}

}

*/
//*****
//*****
//****          Second Filter          ****//
//*****
//*****

//The second filter is used to make sure a point of interest found by the first
//filter is, indeed, an enemy robot. This is useful when the lidar is close to
//a wall and two adjacent data points may have a large delta, giving a false
//positive for an enemy robot. The filter takes the sample of interest from the
//first filter and subtracts it from the preceding samples. If the preceding
//sample is on an opponent, we expect this delta to be smaller than if it were
//on a wall

```

```
if(outputAngleDistance[1] != 0)//one corner has been found, lets make sure it's the enemy, not
the wall
```

```
{
    secondFilter();
}
```

```
/***/
/***/
//****          Third Filter          ****//
/***/
/***/
```

```
/*Here's where we check for the other corner of the enemy.
*This is necessary to find out the full FOV that the enemy takes up.
*We can then use this information to locate our robot within the arena,
*specifically to find the nearest wall*/
/*
```

```
sampleSpace = 3;
while(outputAngleDistance[1] == 0 && sampleSpace <= 9){
    thirdFilter(sampleSpace);
    sampleSpace += 1;
}
*/
```

```
if(outputAngleDistance[1] != 0)//one corner has been found, lets find the other one
```

```
{
    for(j = corner_index;j>4;j--)
    {
        if(extractDistance[j]>extractDistance[j+3] && extractDistance[j+3]!=0 )
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j+3];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
```

```

        if(firstDifference[j]>sample_threshold[2] )
        {
            outputAngleDistance[2] = extractAngle[j];
            outputAngleDistance[3] = extractDistance[j+3];
            front_corner_index = j;
        }
    }

}

if(outputAngleDistance[1] != 0)//one corner has been found, lets find the other one
{
    for(j = corner_index;j>5;j--)
    {
        if(extractDistance[j]>extractDistance[j+4] && extractDistance[j+4]!=0 )
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j+4];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
            if(firstDifference[j]>sample_threshold[3] )
            {
                outputAngleDistance[2] = extractAngle[j];
                outputAngleDistance[3] = extractDistance[j+4];
                front_corner_index = j;
            }
        }
    }

}

if(outputAngleDistance[1] != 0)//one corner has been found, lets find the other one
{
    for(j = corner_index;j>6;j--)

```

```

{
    if(extractDistance[j]>extractDistance[j+5] && extractDistance[j+5]!=0 )
    {

        firstDifference[j] = extractDistance[j]-extractDistance[j+5];

        //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
        if(firstDifference[j]>sample_threshold[4] )
        {
            outputAngleDistance[2] = extractAngle[j];
            outputAngleDistance[3] = extractDistance[j+5];
            front_corner_index = j;
        }
    }

}

}

if(outputAngleDistance[1] != 0)//one corner has been found, lets find the other one
{
    for(j = corner_index;j>7;j--)
    {
        if(extractDistance[j]>extractDistance[j+6] && extractDistance[j+6]!=0 )
        {

            firstDifference[j] = extractDistance[j]-extractDistance[j+6];

            //secondDifference[j] = extractDistance[j-1]-extractDistance[j-2];
            if(firstDifference[j]>sample_threshold[5] )
            {
                outputAngleDistance[2] = extractAngle[j];
                outputAngleDistance[3] = extractDistance[j+6];
                front_corner_index = j;
            }
        }
    }
}

```

```

    }

}

}

//sort for min value if we found both ends of the robot
if(outputAngleDistance[1] != 0 && outputAngleDistance[3] != 0)
{
    for (j=8; j<360;j++)
    {

        if(j > front_corner_index && j < corner_index)
        {
            //extractDistance[j]=0;
            extractDistance[j]=extractDistance[j-1];
            extractAngle[j]=extractAngle[j-1];
        }
        else if(extractDistance[j] == 0)
        {
            extractDistance[j] = extractDistance[j-1];
            extractAngle[j]=extractAngle[j-1];
        }
        else if(extractDistance[j]>extractDistance[j-1])
        {
            extractDistance[j]=extractDistance[j-1];
            extractAngle[j]=extractAngle[j-1];
        }

        wallDistance = extractDistance[j];
        wallAngle = extractAngle[j];
    }
}

```

```

        //wallDistance = extractDistance[j];
        //wallAngle = extractAngle[j];

    }

    look = 1;
    //PORTE = ~wallAngle;//outputAngleDistance[2];//>>5;
    if(Found_Enemy == 1)
    {
        enemyOn();
        lidar.opponentAngle = outputAngleDistance[0];
        if(outputAngleDistance[3] != 0)
        {
            lidar.opponentAngle = (outputAngleDistance[2] + outputAngleDistance[0])>>1;
        }
        lidar.opponentDistance = outputAngleDistance[1];
        //PORTE = ~outputAngleDistance[1]>>5;
    }
    else
    {
        enemyOff();
        //PORTE = 0xff;
    }

    //outputAngleDistance[0] = angle of enemy's back corner
    //outputAngleDistance[1] = distance of enemy's back corner
    //outputAngleDistance[2] = angle of enemy's front corner
    //outputAngleDistance[3] = distance of enemy's front corner

    /*
    teraHandle2 = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_WRITE);//open tera usart channel
    count_1 = DRV_USART_Write(teraHandle2, extractAngle, 738);//write to tera

```



```

    DRV_USART_Close(teraHandle2);//discard tera handle
*/
/*
    teraHandle = DRV_USART_Open(DRV_USART_INDEX_1, DRV_IO_INTENT_WRITE);//open tera usart channel
    count = DRV_USART_Write(teraHandle, extractDistance, 738);//write to tera
    DRV_USART_Close(teraHandle);//discard tera handle

    teraHandle = DRV_USART_Open(DRV_USART_INDEX_1, DRV_IO_INTENT_WRITE);//open tera usart channel
    count = DRV_USART_Write(teraHandle, outputAngleDistance, 4);//write to tera
    DRV_USART_Close(teraHandle);//discard tera handle
*/
    outputAngleDistance[0] = 0; // reset output
    outputAngleDistance[1] = 0;
    outputAngleDistance[2] = 0;
    outputAngleDistance[3] = 0;
    wallDistance = 0;
    wallAngle = 0;
    corner_index = 0;
    front_corner_index = 0;
    n_samples_apart = 0;
    Found_Enemy = 0;

    return lidar;
}

// Function to write LiDAR data
void setLidarData(short oppAng, short oppDist, short wallAng, short wallDist) {
    lidar.opponentAngle = oppAng;
    lidar.opponentDistance = oppDist;
    lidar.wallAngle = wallAng;
    lidar.wallDistance = wallDist;
}

```

```
// Initial simulated LiDAR data
void initLidar(void) {
    setLidarData(lidar.opponentAngle, lidar.opponentDistance,
                lidar.wallAngle, lidar.wallDistance);
}
```

3.2.3 Ultrasonic Sensor

The ultrasonic sensors use time propagation of acoustic waves in atmosphere to determine distance. It does this by sending out a pulse and listening for its echo, then determining the time difference between the pulse and echo. The time is then converted to a distance. The requirements for the ultrasonic sensors in our application are as follows

Table 24: Ultrasonic Sensor Requirements

	Desired	Parallax Ping
Maximum distance	40 cm	3 m
Minimum distance	As small as possible	2 cm

The Parallax Ping Sensor was chosen over the other options due to price, simplicity of integration and meeting all of our requirements. The 40 cm maximum distance requirement was due to the fact that the chosen LiDAR sensor had limited reliability in this range. Due to this fact, it was also required that the ultrasonic sensor have as small a minimum range as possible. The other choices for ultrasonic sensors were not able to integrate with our microprocessor or were not accurate at very close range.

The ultrasonic sensors are necessary as a backup for the LiDAR sensor in close proximity. They will only be used when the LiDAR sensor data is inconclusive in determining whether there is an enemy in front of the robot. The measurement of distance is not as important as knowing that there is an enemy in view. When the robot makes decisions, it will try to use the LiDAR data before

it uses the ultrasonic data.

The chosen ultrasonic sensors are simple, robust and cost efficient sensors. They require one power connection and one signal connection. The trigger pulse or chirp and the echo pulse are both sent/received over the signal pin. The sensor communicates using only a varying width pulse. The width of the pulse varies with a minimum width of 2 microseconds and a typical width of 5 microseconds. Using the output compare module of the microcontroller, a trigger pulse is sent. Then using an input compare module, the width of the echo pulse is measured and that is converted to a distance using an equation that will be found experimentally. Figure 19 shows the ultrasonic sensor and the pulses that are sent and received. Figure 18 shows the pulses that are received and the trigger pulse that is sent, this comes from the datasheet for the sensor.

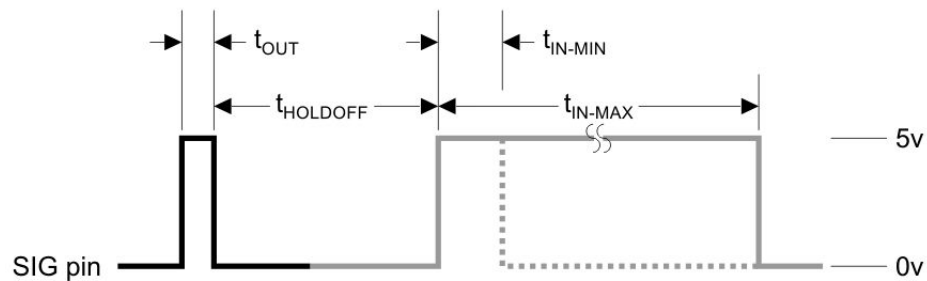


Figure 18: Ultrasonic Pulses (From Datasheet)

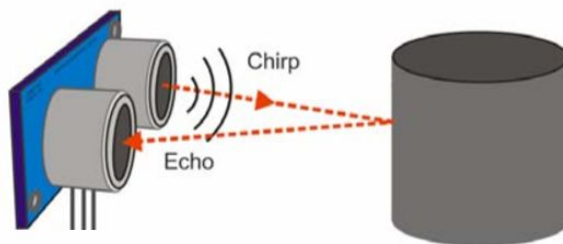


Figure 19: Ultrasonic Sensor

Below, the software is shown for the ultrasonic sensor. The software shows the process for converting the signal from the ultrasonic sensor, to a distance.

ultrasonic.h: header file for the ultrasonic sensors

```
#include <stdbool.h>

// Distance to opponent if detected
// Otherwise return 0
unsigned short getUltrasData(short select);
bool isUltraClose();
void setUltrasData(short data);
void initUltras(void);
```

ultrasonic.c: source file for the ultrasonic sensors

```
#include "ultrasonic.h"
unsigned short Thiu, Thiu2;
unsigned short Tru, Tru2;
unsigned short Tfu, Tfu2;
short ultras = 0;

unsigned short getUltrasData(short select) {
    if (select)
    {
        Tru2 = DRV_IC2_Capture16BitDataRead();
        Tfu2 = DRV_IC2_Capture16BitDataRead();
        if (Tfu2>Tru2)
        {
            Thiu2 = Tfu2 - Tru2;
        }
    }
}
```

```

        return Thiu2;
    }
    else
    {
        Tru = DRV_IC1_Capture16BitDataRead();
        Tfu = DRV_IC1_Capture16BitDataRead();
        if (Tfu>Tru)
        {
            Thiu = Tfu - Tru;
        }

        return Thiu;
    }
}

bool isUltraClose() {
    if (((getUltrasData(0) > 0) && (getUltrasData(0) < 1500)) ||
        ((getUltrasData(1) > 0) && (getUltrasData(1) < 1500)))
        return true;
    else
        return false;
}

void setUltrasData(short data) {
    ultras = data;
}

void initUltras(void) {
    ultras = 0;
}

```

3.2.4 Encoder

The encoder sensor is necessary to measure the rotational speed of the weapon which determines whether the robot is in fight or flight mode. The encoder must be attached to the shaft of the weapon motor so that it rotates at the same speed as the weapon. It will then generate pulses at a frequency proportional to the speed of the motor shaft. The CUI AMT103-V encoder was chosen because it operates at 5V and is lightweight. When it spins, it generates 2048 pulses per revolution (PPR). Per section 3.5.6, the threshold RPM to enter fight-mode is 825 RPM, thus the threshold pulse frequency for fight-mode is: $2048 \text{ PPR} * 825 \text{ RPM} / 60\text{s} = 28,160 \text{ Hz}$. This will be measured using an input compare module.

encoder.h: header file for the encoder that reads weapon speed

```
#include <stdbool.h>

bool isWeaponReady(void);
unsigned short Get_Encoder_Data (void);
```

encoder.c: source file for the encoder that reads weapon speed

```
#include "encoder.h"
#include "system/common/sys_common.h"
#include "app.h"
#include "system_definitions.h"

//int interruptNum = 1;
unsigned short Thie;
```

```

unsigned short Tre;
unsigned short Tfe;

unsigned short Get_Encoder_Data (void)
{
    Tre = DRV_IC0_Capture16BitDataRead();
    Tfe = DRV_IC0_Capture16BitDataRead();
    if (Tfe>Tre)
    {
        Thie = Tfe - Tre;
    }
    else
    {
        Thie=(65535-Tre)+Tfe;
    }

    return Thie;
}

bool isWeaponReady(void)
{
    if(Thie < 40000)
    {
        flightOff();
        return true;
    }
    else
    {
        //flightOn();
        return false; // CHANGE TO FALSE FOR DEMO
    }
}
}

```


3.2.5 Gyroscope/Accelerometer

The gyroscope is necessary in order to tell if the robot has been flipped over. This will be used by the Autonomous system and the control system (The Motion and Actualization Team). The sensor that was chosen communicates using the I2C protocol. It will let the microprocessor know whether gravity is up or down. If the robot has been flipped over, the motors will need to operate in reverse and the autonomous system will be deactivated because it would not operate with the LiDAR on the bottom. The accelerometer portion of the sensor is going to be used to assist the robot in knowing if it has hit an opponent. This information can be incorporated in the fight or flight algorithm as a double check in determining whether the robot should engage an enemy.

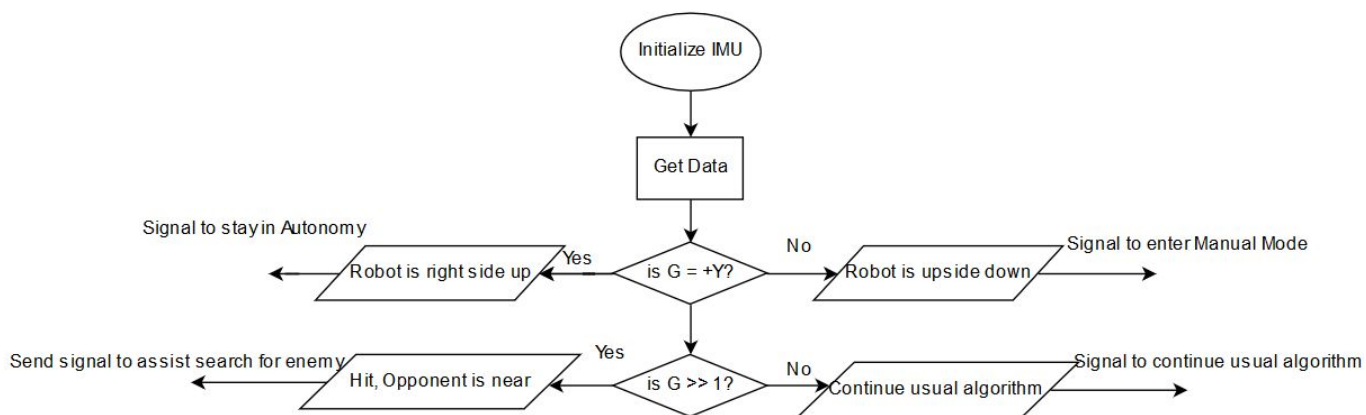


Figure 20: Gyroscope Diagram

gyroscope.h: header file for the gyroscope

```
#define MPU6050_REG_ACCEL_XOFFS_H    (0x06)
#define MPU6050_REG_ACCEL_XOFFS_L    (0x07)
#define MPU6050_REG_ACCEL_YOFFS_H    (0x08)
#define MPU6050_REG_ACCEL_YOFFS_L    (0x09)
#define MPU6050_REG_ACCEL_ZOFFS_H    (0x0A)
#define MPU6050_REG_ACCEL_ZOFFS_L    (0x0B)
#define MPU6050_REG_GYRO_XOFFS_H     (0x13)
#define MPU6050_REG_GYRO_XOFFS_L     (0x14)
#define MPU6050_REG_GYRO_YOFFS_H     (0x15)
#define MPU6050_REG_GYRO_YOFFS_L     (0x16)
#define MPU6050_REG_GYRO_ZOFFS_H     (0x17)
#define MPU6050_REG_GYRO_ZOFFS_L     (0x18)
#define MPU6050_SAMPLE_RATE_DIVIDER  (0x19)
#define MPU6050_REG_CONFIG           (0x1A)
#define MPU6050_REG_GYRO_CONFIG       (0x1B) // Gyroscope Configuration
#define MPU6050_REG_ACCEL_CONFIG      (0x1C) // Accelerometer Configuration
#define MPU6050_REG_FF_THRESHOLD      (0x1D)
#define MPU6050_REG_FF_DURATION       (0x1E)
#define MPU6050_REG_MOT_THRESHOLD     (0x1F)
#define MPU6050_REG_MOT_DURATION      (0x20)
#define MPU6050_REG_ZMOT_THRESHOLD    (0x21)
#define MPU6050_REG_ZMOT_DURATION     (0x22)
#define MPU6050_REG_INT_PIN_CFG       (0x37) // INT Pin. Bypass Enable Configuration
#define MPU6050_REG_INT_ENABLE        (0x38) // INT Enable
#define MPU6050_REG_INT_STATUS        (0x3A)
#define MPU6050_REG_ACCEL_XOUT_H      (0x3B)
#define MPU6050_REG_ACCEL_XOUT_L      (0x3C)
#define MPU6050_REG_ACCEL_YOUT_H      (0x3D)
#define MPU6050_REG_ACCEL_YOUT_L      (0x3E)
#define MPU6050_REG_ACCEL_ZOUT_H      (0x3F)
```

```

#define MPU6050_REG_ACCEL_ZOUT_L      (0x40)
#define MPU6050_REG_TEMP_OUT_H       (0x41)
#define MPU6050_REG_TEMP_OUT_L       (0x42)
#define MPU6050_REG_GYRO_XOUT_H      (0x43)
#define MPU6050_REG_GYRO_XOUT_L      (0x44)
#define MPU6050_REG_GYRO_YOUT_H      (0x45)
#define MPU6050_REG_GYRO_YOUT_L      (0x46)
#define MPU6050_REG_GYRO_ZOUT_H      (0x47)
#define MPU6050_REG_GYRO_ZOUT_L      (0x48)
#define MPU6050_REG_MOT_DETECT_STATUS (0x61)
#define MPU6050_REG_MOT_DETECT_CTRL  (0x69)
#define MPU6050_REG_USER_CTRL        (0x6A) // User Control
#define MPU6050_REG_PWR_MGMT_1       (0x6B) // Power Management 1
#define MPU6050_REG_WHO_AM_I         (0x75) // Who Am I

#include <math.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "system_config/default/framework/driver/i2c/drv_i2c_static_buffer_model.h"

typedef struct
{
    float XAxis;
    float YAxis;
    float ZAxis;
} Vector;

typedef struct
{
    bool isOverflow;
    bool isFreeFall;
    bool isInactivity;
}

```

```

bool isActivity;
bool isPosActivityOnX;
bool isPosActivityOnY;
bool isPosActivityOnZ;
bool isNegActivityOnX;
bool isNegActivityOnY;
bool isNegActivityOnZ;
bool isDataReady;

```

```

}Activites;

```

```

typedef enum

```

```

{
    MPU6050_CLOCK_KEEP_RESET      = 0b111,
    MPU6050_CLOCK_EXTERNAL_19MHZ  = 0b101,
    MPU6050_CLOCK_EXTERNAL_32KHZ  = 0b100,
    MPU6050_CLOCK_PLL_ZGYRO       = 0b011,
    MPU6050_CLOCK_PLL_YGYRO       = 0b010,
    MPU6050_CLOCK_PLL_XGYRO       = 0b001,
    MPU6050_CLOCK_INTERNAL_8MHZ   = 0b000

```

```

} mpu6050_clockSource_t;

```

```

typedef enum

```

```

{
    MPU6050_SCALE_2000DPS         = 0b11,
    MPU6050_SCALE_1000DPS         = 0b10,
    MPU6050_SCALE_500DPS          = 0b01,
    MPU6050_SCALE_250DPS          = 0b00

```

```

} mpu6050_dps_t;

```

```

typedef enum

```

```

{
    MPU6050_RANGE_16G              = 0b11,
    MPU6050_RANGE_8G               = 0b10,

```

```
    MPU6050_RANGE_4G           = 0b01,
    MPU6050_RANGE_2G           = 0b00,
} mpu6050_range_t;
```

```
typedef enum
```

```
{
    MPU6050_DELAY_3MS           = 0b11,
    MPU6050_DELAY_2MS           = 0b10,
    MPU6050_DELAY_1MS           = 0b01,
    MPU6050_NO_DELAY            = 0b00,
} mpu6050_onDelay_t;
```

```
typedef enum
```

```
{
    MPU6050_DHPF_HOLD           = 0b111,
    MPU6050_DHPF_0_63HZ         = 0b100,
    MPU6050_DHPF_1_25HZ         = 0b011,
    MPU6050_DHPF_2_5HZ          = 0b010,
    MPU6050_DHPF_5HZ            = 0b001,
    MPU6050_DHPF_RESET          = 0b000,
} mpu6050_dhpf_t;
```

```
typedef enum
```

```
{
    MPU6050_DLPF_6               = 0b110,
    MPU6050_DLPF_5               = 0b101,
    MPU6050_DLPF_4               = 0b100,
    MPU6050_DLPF_3               = 0b011,
    MPU6050_DLPF_2               = 0b010,
    MPU6050_DLPF_1               = 0b001,
    MPU6050_DLPF_0               = 0b000,
} mpu6050_dlpf_t;
```

```

typedef enum{
    MPU6050_Address_1 = (0xD0),
    MPU6050_Address_2 = (0xD2)
}MPU6050_ADDRESS;

typedef struct{
    //private:
    Vector ra, rg; // Raw vectors
    Vector na, ng; // Normalized vectors
    Vector tg, dg; // Threshold and Delta for Gyro
    Vector th;     // Threshold
    Activites a;  // Activities

    float dpsPerDigit, rangePerDigit;
    float actualThreshold;
    bool useCalibrate;
    MPU6050_ADDRESS Address;
}MPU_6050_t;
MPU_6050_t MPU_1;

void isGyroTimedOut();
bool initGyro();
void setClockSource(MPU6050_ADDRESS address, mpu6050_clockSource_t source);
mpu6050_clockSource_t getClockSource(MPU6050_ADDRESS address);
void setScale(MPU_6050_t *mpu,mpu6050_dps_t scale);
mpu6050_dps_t getScale(MPU6050_ADDRESS address);
void setRange(MPU_6050_t *mpu,mpu6050_range_t range);
mpu6050_range_t getRange(MPU6050_ADDRESS address);
void setSleepEnabled(MPU6050_ADDRESS address, bool state);
bool getSleepEnabled(MPU6050_ADDRESS address);
int8_t readRegister8(MPU6050_ADDRESS address,uint8_t reg);
void writeRegister8(MPU6050_ADDRESS address,uint8_t reg, uint8_t value);
bool readRegisterBit(MPU6050_ADDRESS address,uint8_t reg, uint8_t pos);

```

```

void writeRegisterBit(MPU6050_ADDRESS address,uint8_t reg,
    uint8_t pos, bool state);
Vector readRawAccel(void);
Vector readScaledAccel(void);
double readAngleX(void);
bool isUpsideDown(void);

```

gyroscope.c: source file for the gyroscope

```

#include "gyroscope.h"
#include "timers.h"

timers_t sec, ms10, ledTimeout;
timers_t printTimer;
timers_t IMU_UpdateTimer;
timers_t timeOut;

// Starts up the gyroscope and tests an initial reading
bool initGyro() {
    // Use chosen scale enum, chosen range enum,
    // and MPU6050_ADDRESS for parameters
    MPU_6050_t *mpu = &MPU_1;
    mpu6050_dps_t scale = MPU6050_SCALE_250DPS;
    mpu6050_range_t range = MPU6050_RANGE_2G;
    MPU6050_ADDRESS mpua = MPU6050_Address_1;

    setTimerInterval(&ledTimeout,100);
    setTimerInterval(&printTimer,100);
    setTimerInterval(&IMU_UpdateTimer,10);
    setTimerInterval(&timeOut,500);

    // Set Address
    mpu->Address = mpua;

```

```

// Reset calibrate values
mpu->dg.XAxis = 0;
mpu->dg.YAxis = 0;
mpu->dg.ZAxis = 0;
mpu->useCalibrate = false;

// Reset threshold values
mpu->tg.XAxis = 0;
mpu->tg.YAxis = 0;
mpu->tg.ZAxis = 0;
mpu->actualThreshold = 0;

// Set Clock Source
setClockSource(mpu->Address, MPU6050_CLOCK_PLL_XGYRO);

// Set Scale & Range
setScale(mpu, scale);

setRange(mpu, range);

//SET THE SAMPLE RATE
writeRegister8(mpu->Address, MPU6050_SAMPLE_RATE_DIVIDER, 9);

//Config DLPF
writeRegister8(mpu->Address, MPU6050_REG_CONFIG, 1);

// Disable Sleep Mode
setSleepEnabled(mpu->Address, false);

// Tests the first reading of the accelerometer
readRawAccel();
return true;
}

```



```

// Sets the clock source of the MPU at initialization
void setClockSource(MPU6050_ADDRESS address, mpu6050_clockSource_t source) {
    uint8_t value;
    value = readRegister8(address, MPU6050_REG_PWR_MGMT_1);
    value &= 0b11111000;
    value |= source;
    writeRegister8(address, MPU6050_REG_PWR_MGMT_1, value);
}

```

```

// Gets the clock source of the MPU at initialization
mpu6050_clockSource_t getClockSource(MPU6050_ADDRESS address) {
    uint8_t value;
    value = readRegister8(address, MPU6050_REG_PWR_MGMT_1);
    value &= 0b00001111;
    return (mpu6050_clockSource_t)value;
}

```

```

// Sets the scale of the MPU at initialization
void setScale(MPU_6050_t *mpu, mpu6050_dps_t scale) {
    uint8_t value;

    switch (scale)
    {
        case MPU6050_SCALE_250DPS:
            mpu->dpsPerDigit = .007633f;
            break;
        case MPU6050_SCALE_500DPS:
            mpu->dpsPerDigit = .015267f;
            break;
        case MPU6050_SCALE_1000DPS:
            mpu->dpsPerDigit = .030487f;
            break;
    }
}

```

```

    case MPU6050_SCALE_2000DPS:
        mpu->dpsPerDigit = .060975f;
        break;
    default:
        break;
}

value = readRegister8(mpu->Address, MPU6050_REG_GYRO_CONFIG);
value &= 0b11100111;
value |= (scale << 3);
writeRegister8(mpu->Address, MPU6050_REG_GYRO_CONFIG, value);
}

// Gets the scale of the MPU at initialization
mpu6050_dps_t getScale(MPU6050_ADDRESS address) {
    uint8_t value;
    value = readRegister8(address, MPU6050_REG_GYRO_CONFIG);
    value &= 0b00011000;
    value >>= 3;
    return (mpu6050_dps_t)value;
}

// Sets the range of the MPU at initialization
void setRange(MPU_6050_t *mpu, mpu6050_range_t range) {
    uint8_t value;

    switch (range) {
        case MPU6050_RANGE_2G:
            mpu->rangePerDigit = .000061f;
            break;
        case MPU6050_RANGE_4G:
            mpu->rangePerDigit = .000122f;
            break;
    }
}

```

```

    case MPU6050_RANGE_8G:
        mpu->rangePerDigit = .000244f;
        break;

    case MPU6050_RANGE_16G:
        mpu->rangePerDigit = .0004882f;
        break;

    default:
        break;
}

value = readRegister8(mpu->Address, MPU6050_REG_ACCEL_CONFIG);
value &= 0b11100111;
value |= (range << 3);
writeRegister8(mpu->Address, MPU6050_REG_ACCEL_CONFIG, value);
}

// Gets the range of the MPU at initialization
mpu6050_range_t getRange(MPU6050_ADDRESS address) {
    uint8_t value;

    value = readRegister8(address, MPU6050_REG_ACCEL_CONFIG);
    value &= 0b00011000;
    value >>= 3;

    return (mpu6050_range_t)value;
}

// Disables sleep mode at initialization
void setSleepEnabled(MPU6050_ADDRESS address, bool state) {
    writeRegisterBit(address, MPU6050_REG_PWR_MGMT_1, 6, state);
}

// Gets sleep mode at initialization
bool getSleepEnabled(MPU6050_ADDRESS address) {
    return readRegisterBit(address, MPU6050_REG_PWR_MGMT_1, 6);
}

```

```

// Read 8-bit from register
int8_t readRegister8(MPU6050_ADDRESS address,uint8_t reg) {
    resetTimer(&timeOut);
    char value;
    int8_t addr = reg;
    DRV_I2C_BUFFER_HANDLE handle = DRV_I2C0_Transmit(address,&addr,1,NULL);
    while(!(DRV_I2C0_TransferStatusGet(handle)==DRV_I2C_BUFFER_EVENT_COMPLETE ||
        DRV_I2C0_TransferStatusGet(handle)==DRV_I2C_BUFFER_EVENT_ERROR ||
        timerDone(&timeOut)));

    handle = DRV_I2C0_Receive(address,&value,1,NULL);
    while(!(DRV_I2C0_TransferStatusGet(handle)==DRV_I2C_BUFFER_EVENT_COMPLETE ||
        DRV_I2C0_TransferStatusGet(handle)==DRV_I2C_BUFFER_EVENT_ERROR ||
        timerDone(&timeOut)));

    return value;
}

// Write 8-bit to register
void writeRegister8(MPU6050_ADDRESS address,uint8_t reg, uint8_t value) {
    resetTimer(&timeOut);
    char byte[] = {reg, value};
    DRV_I2C_BUFFER_HANDLE handle = DRV_I2C0_Transmit (address,byte,2,NULL);
    while(!(DRV_I2C0_TransferStatusGet(handle)==DRV_I2C_BUFFER_EVENT_COMPLETE ||
        DRV_I2C0_TransferStatusGet(handle)==DRV_I2C_BUFFER_EVENT_ERROR ||
        timerDone(&timeOut)));
}

// Read register bit
bool readRegisterBit(MPU6050_ADDRESS address,uint8_t reg, uint8_t pos) {
    int8_t value;
    value = readRegister8(address,reg);
}

```

```

    return ((value >> pos) & 1);
}

// Write register bit
void writeRegisterBit(MPU6050_ADDRESS address,uint8_t reg, uint8_t pos,
    bool state) {
    int8_t value;
    value = readRegister8(address,reg);

    if (state)
        value |= (1 << pos);
    else
        value &= ~(1 << pos);

    writeRegister8(address,reg, value);
}

// Reads the raw accelerometer values from the gyro
Vector readRawAccel(void) {

    resetTimer(&timeOut);

    // Should only need to transmit the address of the first registers
    unsigned char bytesTX = {MPU6050_REG_ACCEL_XOUT_H};
    unsigned char bytesRX[6];

    DRV_I2C_BUFFER_HANDLE handle =
        DRV_I2C0_Transmit((&MPU_1)->Address,&bytesTX,1,NULL);
    while(!( DRV_I2C0_TransferStatusGet(handle) == DRV_I2C_BUFFER_EVENT_COMPLETE
        || DRV_I2C0_TransferStatusGet(handle) == DRV_I2C_BUFFER_EVENT_ERROR
        || timerDone(&timeOut)));

    handle = DRV_I2C0_Receive((&MPU_1)->Address,&bytesRX,6,NULL);
}

```

```

while(!( DRV_I2C0_TransferStatusGet(handle) == DRV_I2C_BUFFER_EVENT_COMPLETE
        || DRV_I2C0_TransferStatusGet(handle) == DRV_I2C_BUFFER_EVENT_ERROR
        || timerDone(&timeOut)));

(&MPU_1)->ra.XAxis = bytesRX[0] << 8 | bytesRX[1];
(&MPU_1)->ra.YAxis = bytesRX[2] << 8 | bytesRX[3];
(&MPU_1)->ra.ZAxis = bytesRX[4] << 8 | bytesRX[5];

return (&MPU_1)->ra;
}

// Applies the scale to the raw accelerometer value
Vector readScaledAccel(void) {
    readRawAccel();

    (&MPU_1)->na.XAxis = (&MPU_1)->ra.XAxis * (&MPU_1)->rangePerDigit;
    (&MPU_1)->na.YAxis = (&MPU_1)->ra.YAxis * (&MPU_1)->rangePerDigit;
    (&MPU_1)->na.ZAxis = (&MPU_1)->ra.ZAxis * (&MPU_1)->rangePerDigit;

    return (&MPU_1)->na;
}

// Read the X angle from the accelerometer
double readAngleX(void) {
    Vector radians = readScaledAccel();
    return (180/3.141592) * radians.XAxis;
}

int gyroBuffer[10];    // Stores the last 10 accelerometer readings
int bufferTracker = 0; // Looping iterator for gyroBuffer
double xValue = 0;    // Stores the X angle
double threshold = 90.0;// Maximum angle to consider robot upside down
// Read whether the gyro is upside down or not

```

```

bool isUpsideDown(void) {
    // Reads the accelerometer Z angle
    xValue = readAngleX();

    // Tests if the robot is upside down based on the threshold angle
    if (xValue > threshold) {
        gyroBuffer[bufferTracker] = 1;
    } else {
        gyroBuffer[bufferTracker] = 0;
    }

    // Counts how many of the last 10 readings were upside down
    int i, upsideDownCount = 0;
    for (i = 0; i < 10; i++){
        if (gyroBuffer[i] == 1) {
            upsideDownCount++;
        }
    }

    // Updates the iterator
    if (bufferTracker < 9) {
        bufferTracker++;
    } else {
        bufferTracker = 0;
    }

    // If at least half the readings show upside down, return true
    if (upsideDownCount == 10){
        return true;
    }

    // Otherwise return false
    return false;
}

```

[HL, CH]

3.2.6 Communication Protocols Software

For the four sensors, four different communication protocols are necessary. For the LiDAR, UART and PWM are required. For the Gyroscope, I2C is required. Lastly, a standard digital pulse signal is required for the Ultrasonic sensor and the encoder. MPLab Harmony simplifies the process of assigning pins and using the communications protocols. Drivers for all the necessary protocols are included with the configuration files.

For the communication protocol between the boards of the Autonomy System and the Controls Actualization system, UART is used. UART is used because of its simplicity, speed and because it is asynchronous. It is important that it is asynchronous because that simplifies the connection between the two systems, and only requires two wires. In addition, it is also the protocol that the LiDAR uses to send its data to the microprocessor. That helps with the design process because the same skeleton UART program can be used for both parts of the system. The microprocessor needs to have one UART input and one UART output.

Using an Explorer 16/32 board, preliminary bench tests have already been done. Using the MPLab software with MPLab Harmony Configurator, key signals have already been simulated. The first thing that was done was to use the UART functions to send and receive a message with the processor. A simple program was found on the Microchip Developers Website that would accept a character over UART and send back the next character. For our testing, we used the build in UART to USB controller on the Explorer 16/32 board. This communicated with a laptop with Putty installed on it. This program was not configured to use the PIC32MZEF100 so a custom configuration file was made using the MPLab Harmony Configurator, allowing the program to function correctly. This program was modified to do some other things as well, including flashing

LEDs and require buttons to do certain things.

[HL]

3.3.0 Bill of Material as Standing

3.3.1 Parts List

The parts list included shows all the parts required for the Autonomous System. This includes the sensors and all the necessary components for the microcontroller board.

Table 25: Parts List/Materials Budget List

Qty	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Website	System	Unit Cost	Total Cost
1	RPLIDARA3	Lidar	DFRobot	DFR0583	Link	Software	\$ 599.00	\$ 599.00
2	28015	Ultrasonic	Parallax	28015-ND	Link	Software	\$ 29.99	\$ 59.98
1	SEN0142	6 DOF SENSOR - MPU6050	Digi-Key	1738-1070-ND	Link	Electrical	\$ 10.20	\$ 10.20
1	AMT103-V	Encoders Axial Encoder Kit 9 sleeve, base, cover	Mouser	490-AMT103-V	Link	Electrical	\$ 23.62	\$ 23.62
1	MA320019	PIC32MZ2048EFH100 - Plug-In Module (PIM) (For explorer 16/32 board)	Digikey	MA320019-ND	Link	Software	\$ 25.75	\$ 25.75
1	N/A	Boards 5X5	OSH Park		Link	Electrical	\$ 120.00	\$ 120.00
1	PIC32MZ2048EFH064	Microcontroller	Digikey	PIC32MZ2048EFH064-I/P T-ND	Link	Electrical	\$ 12.23	\$ 12.23
1	N/A	Passive Components- Resistors, Capacitors, etc. (See Eagle Parts List)	ECE Stock/Misc			Electrical	\$ 65.00	\$ 65.00
							Total	\$ 915.78

3.3.2 Eagle BOM

Qty	Value	Device	Package	Parts
3		CONN_02	1X02	J2, J6, J9
1		CONN_03LOCK	1X03_LOCK	J4
3		CONN_05LOCK	1X05_LOCK	J1, J3, J5
10		LEDCHIP-LED0805	CHIP-LED0805	LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8, LED9, LED10
1		PCIFEMALE	PCI-CONNECTOR	LED
10	0.1uF Ceramic	CAP0805	805	C5, C8, C35, C36, C37, C38, C39, C40, C41, C42
1	100 5%	R-US_R0805	R0805	R12
1	10k 5%	R-US_R0805	R0805	R13
2	10nF Ceramic	CAP0805	805	C4, C9
2	120k, 5%	R-US_R0805	R0805	R8, R16
2	1uF Ceramic	CAP0805	805	C3, C7
4	2.2k	R-US_R0805	R0805	R6, R7, R10, R11
1	270	R-US_R0805	R0805	R31
1	400	R-US_R0805	R0805	R14
2	470pF, 5% Ceramic	CAP0805	805	C1, C2

2	4N27	4N27	DIP880W50P254L780H400 Q6B	U4-ISO3, U4-ISO4
2	55, 5%	R-US_R0805	R0805	R9, R15
8	90	R-US_R0805	R0805	R1, R2, R3, R4, R5, R32, R66, R67
2	DC//DC-ISO	DC//DC-ISO	8-SMD_5-LEADS	U4-ISO1, U4-ISO2
1	KC2520B	KC2520B	KC2520B	U\$9
1	PIC32MZ2048EF H064	PIC32MZ2048EFH06 4	TQFP64	U\$5
1	Prog	M05LOCK_LONGPADS	1X05_LOCK_LONGPADS	PIC_PROG1
1	SWITCH-MOMENTARY-2SMD-4	SWITCH-MOMENTARY-2SMD-4	TACTILE_SWITCH_TALL	S2
1	USB-MINIB-5PIN	USB-MINIB-5PIN	USB-MINIB	U\$11

3.4 Mechanical Sketch of System

Through research trade studies (see Appendix) and video research of various combat robotics competitions, the team has decided to incorporate a hybrid wedge drum weapons system. This is different than the original idea of a spinner. The team is currently still developing this design, and researching other weapon design ideas. Wedge bots have a advantage over many because they can get under the opponent robot and avoid the opponent's weapon or disabling them by flipping them. The drawback of a wedge alone is if the enemy robot is designed to drive on both top and bottom the wedge is not likely to disable the robot. Wedges alone generally do not inflict critical damage.

The proposed design will add a drum weapon mechanism to the wedge design. This will allow the robot to inflict critical damage to the underside of the opponent's robot. The team's hybrid design will have the robustness and defensive capabilities of a wedge, but have the damaging capabilities of a drum weapon system. This is because, as mentioned previously, the wedge is good primarily for avoiding the opponent's weapons and flipping the opponent but not inflicting damage to destroy the opponent.

Using a wedge-drum hybrid will allow for all of the benefits of the wedge, but will also provide a mechanism which will be able to actually disable or destroy the opposing robot. This hybrid design will support the implementation of the intercept or escape algorithms. Figure 21 shows is the original rough mechanical sketch of the system for the autonomous robot. In essence, this automatic autonomous combat robot which will be fully autonomous will be a combination of a drum weapon and a wedge weapon. Further explanation of the diagram can be found in table 3.

[CH]

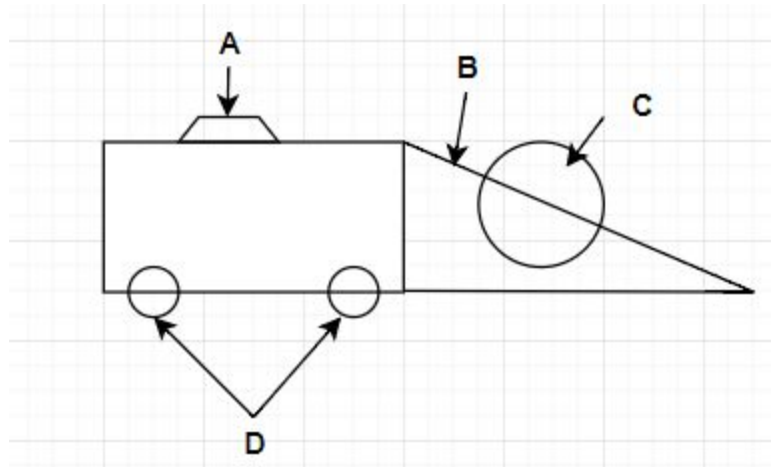


Figure 21: Mechanical Sketch of System.

Table 26: Mechanical Components of System.

Label	Component and Purpose
A	LiDAR Sensor to enable the autonomous combat robot to sense opposing robot and walls of arena.
B	Wedge. Serves as armor, defensive system, and secondary weapon during combat.
C	Drum. Serves as primary weapon for the combat robot.
D	Wheels. Currently a subject being further investigated. The team may implement 360 degree wheels to keep the robot facing the opponent at all times.

Figure 22 and 23 below shows the second rendition of the mechanical design. The team plans on moving the LiDAR sensors into the chassis to protect them. The drum spinner has been made a small width to enable fast speed up times and increase impact delivery force by minimizing the distribution of the impact.

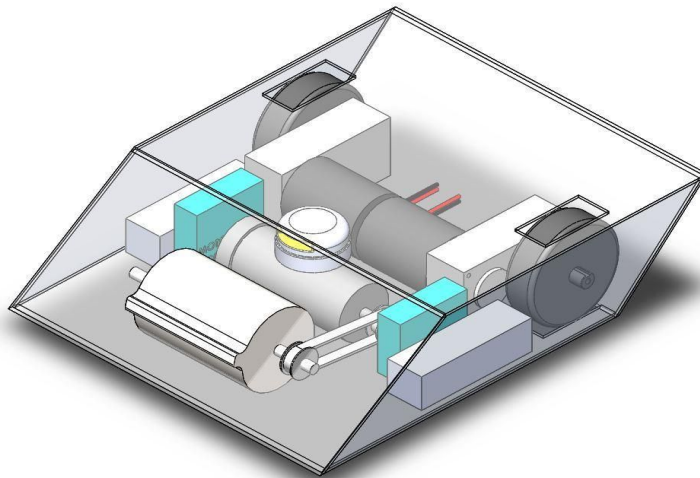


Figure 22: Updated Mechanical Design - Isometric View.

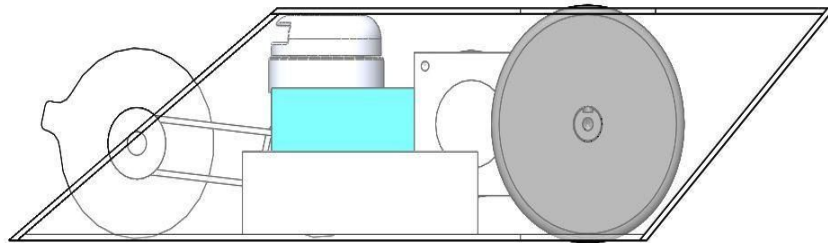


Figure 23: Updated Mechanical Design - Planar View

[FA, CH, AS, TT, TW]

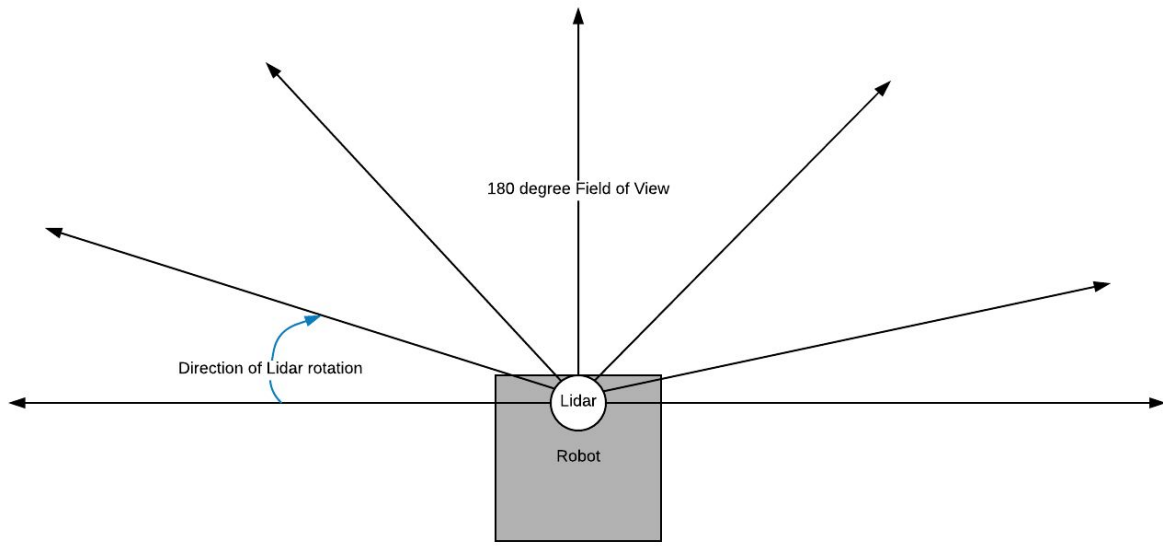


Figure 24: Representation of Lidar Field of View

[SV]

3.5 Calculations

3.5.1 LiDAR Calculations

The size of the arena and the Lidar range. The RoboGames arena is 40 feet squared for a maximum distance from opponent 56.56 feet.

$$\textit{longestdistance} = 40^2 + 40^2 = 56.56\textit{ft}$$

[HL]

The range of the Lidar sensor that we have chosen has a range of 25 meters or 82 feet for white objects. The range of the sensor is lower (around 10 meters or 32 feet) for black objects. With an average range of 20 meters, the enemy should always be in range.

$$\textit{Range} = 20\textit{meters} = 65\textit{feet} > 56.56\textit{ft}$$

[HL]

Lidar's ability to detect the opponent

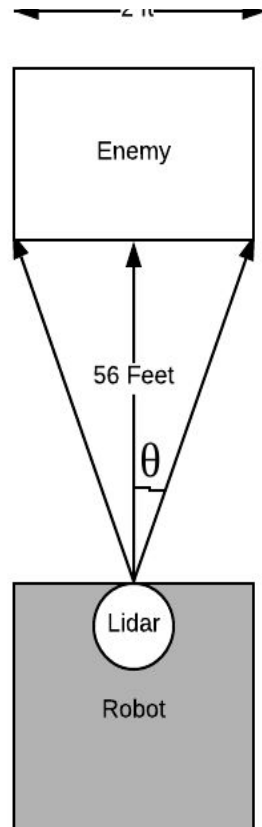


Figure 25: Lidar FOV With Enemy at Max Distance

[SV]

Based on Figure 25, in a situation where the opponent is as far away in the arena as possible and is 2 feet wide:

$$\Theta = \tan^{-1}\left(\frac{1}{56}\right) = 1.023^\circ$$

$$2 * \Theta = 2.046^\circ = \text{minimum field of view occupied by enemy}$$

Lidar sample rate = 16kHz

Lidar spin rate = 10Hz = 3600 Degrees/second

$$16000 \frac{\text{samples}}{\text{second}} * \frac{1 \text{ second}}{3600 \text{ degrees}} = 4.44 \frac{\text{samples}}{\text{degree}}$$

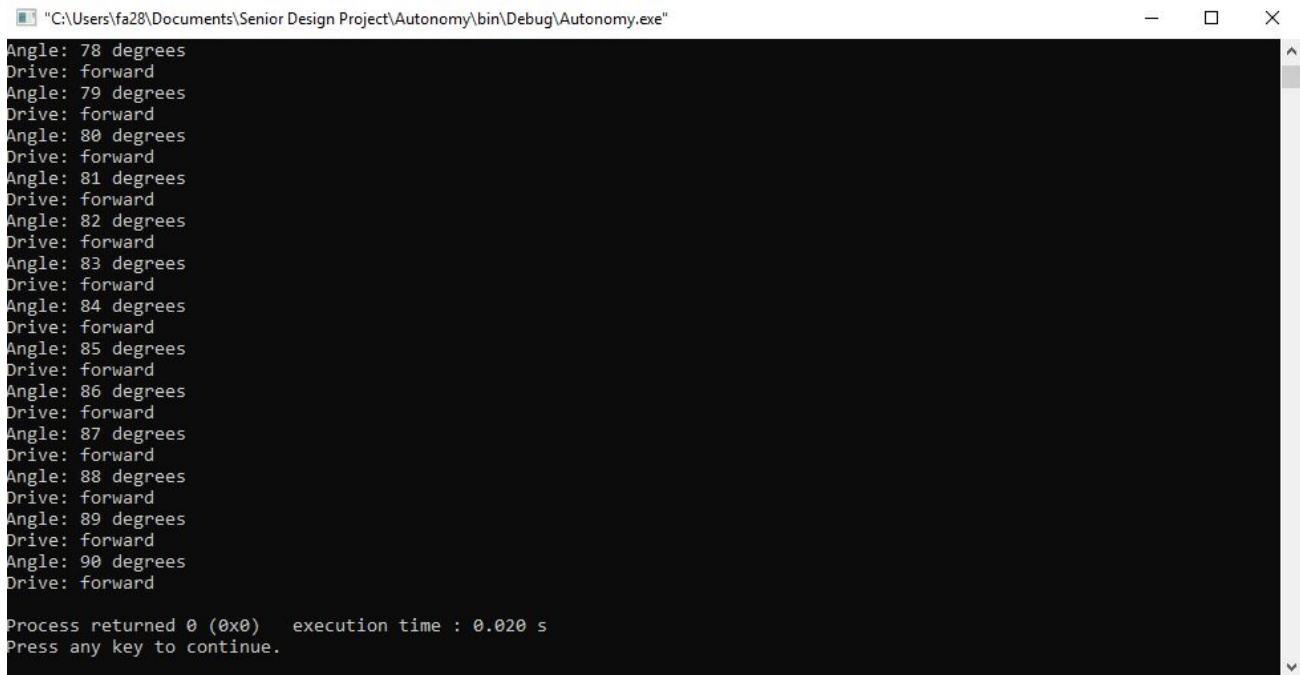
$$4.44 \frac{\text{samples}}{\text{degree}} * 2.046 \text{ degrees} = 9 \text{ samples}$$

Therefore, 9 samples is the fewest number of Lidar samples per revolution that will see the enemy, assuming the opponent is in field of view. Sample rate and spin rate of the Lidar are based on the RPLIDAR A3 which was selected based on sample rate, working distance, and low cost. [16]

[SV]

3.5.2 Main Algorithm Simulation

In order to test the functionality of the main algorithm and the autonomous decision making of the robot, all the sensor inputs have been replaced with simulated data. In Figure 26, the opponent is “moving” in front of the robot until it is at a 90 degree angle. The robot’s decision making responds accordingly:



```
"C:\Users\fa28\Documents\Senior Design Project\Autonomy\bin\Debug\Autonomy.exe"
Angle: 78 degrees
Drive: forward
Angle: 79 degrees
Drive: forward
Angle: 80 degrees
Drive: forward
Angle: 81 degrees
Drive: forward
Angle: 82 degrees
Drive: forward
Angle: 83 degrees
Drive: forward
Angle: 84 degrees
Drive: forward
Angle: 85 degrees
Drive: forward
Angle: 86 degrees
Drive: forward
Angle: 87 degrees
Drive: forward
Angle: 88 degrees
Drive: forward
Angle: 89 degrees
Drive: forward
Angle: 90 degrees
Drive: forward
Process returned 0 (0x0) execution time : 0.020 s
Press any key to continue.
```

Figure 26: Simulated Output of the Main Algorithm

This simulation demonstrates that the robot can appropriately follow the opponent at every iteration of the loop. It also demonstrates a quick processing time, which is crucial in real combat as most of the processing time will be consumed by reading sensors.

3.5.3 LiDAR Simulation & Data Processing

Figure 27 shows the robot in the arena in an ideal scenario:

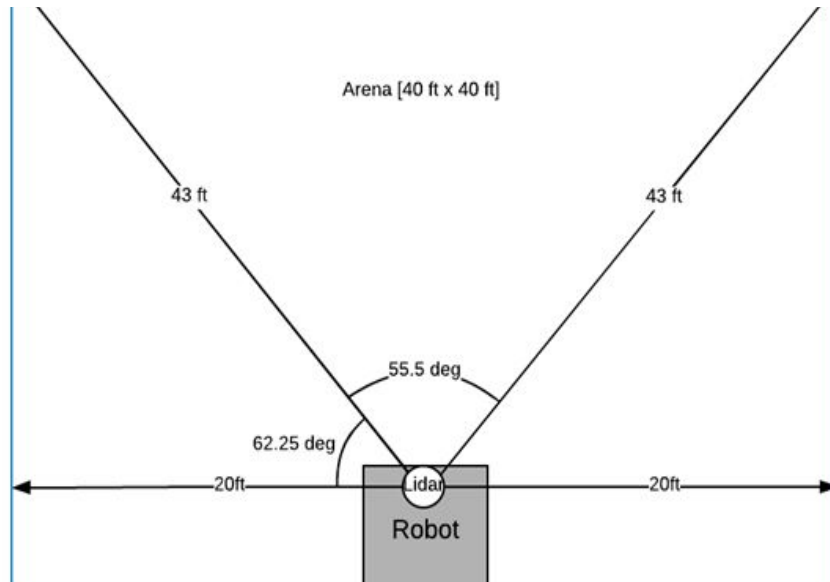
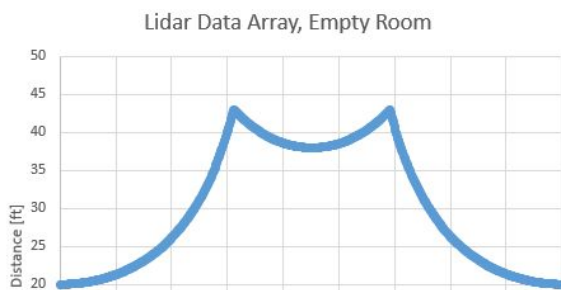


Figure 27: Ideal Orientation, Full FOV

In Figure 27 the robot has its back to the wall, is in the middle of the wall, and can use its full field of view to scan the empty arena. In this instance, the LiDAR will return the symmetrical data array shown in Figure 28. Figure 29 shows this same scenario except with a 2'x2' opponent located in the middle of the opposite wall. Together, these two graphs show how an opponent will be discernible from the arena walls. This can be done simply by subtracting one sample from its neighboring sample. The largest delta in neighboring samples will be due to the enemy robot in the ideal orientation.



124

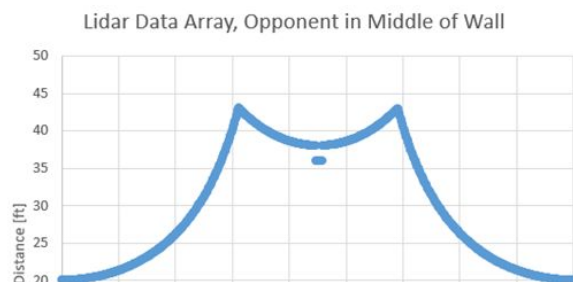


Figure 28 (left) Empty Room, Figure 29 (right) Opponent On Opposite Wall

[SV]

Figure 29 shows the robot in the arena in a non-ideal orientation. Here, only half of the total field of view can be utilized.

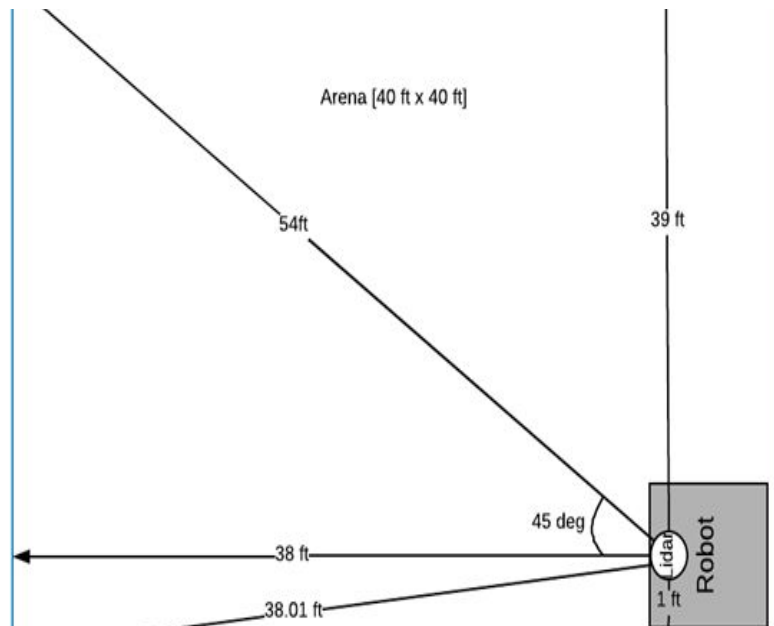


Figure 30: Non-ideal Orientation, Half FOV

[SV]

The data array resulting from the Figure 30 orientation is shown in Figure 31. Figure 32 shows the array that results from subtracting each sample in Figure 31 from its neighbor. Since Figure 29 shows that there are delta values larger than two feet, it would not be possible to locate a 2'x2' enemy when the robot is in this non-ideal orientation. More data processing must be performed to locate the enemy.

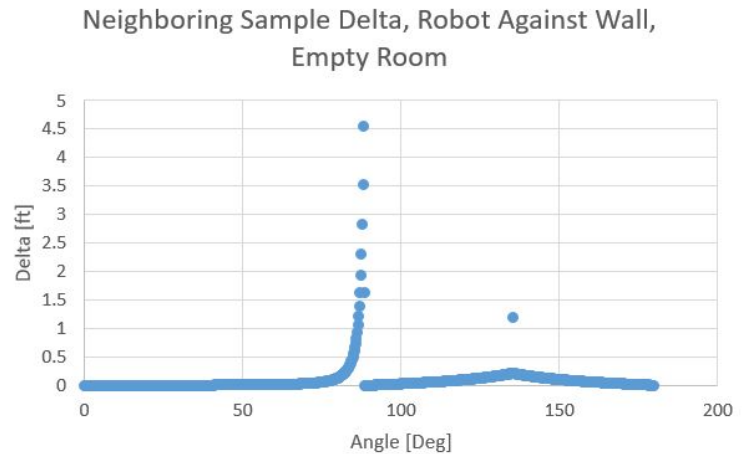
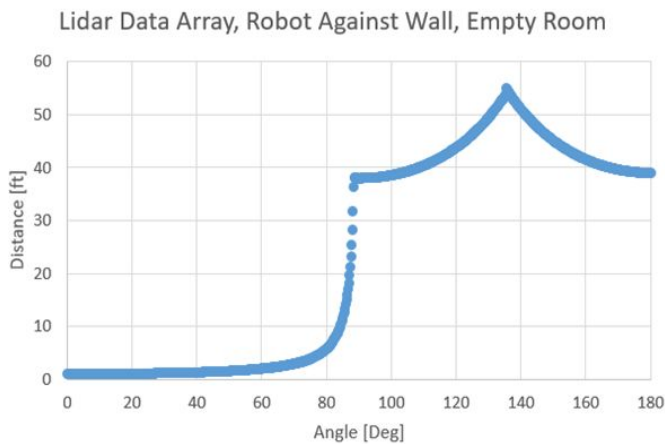


Figure 31 (left) Non-ideal Orientation, Empty Room. Figure 32 (right) Neighboring Sample Delta Array

[SV]

In order to isolate the enemy, a logical test can be performed on each data point whose delta was greater than 2 feet from its neighbor. Since an enemy is on average 2 feet wide, when a sample has a delta greater than two feet, the next two samples should have deltas less than two feet. If this is not the case, then the LiDAR is not measuring the enemy. Figure 31 shows the resulting data array when this algorithm is applied to the array from Figure 32, with three enemies added to the arena.

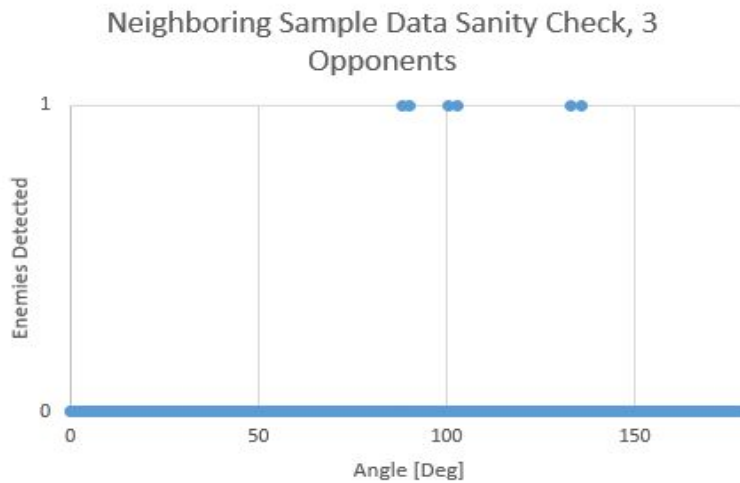


Figure 33: Enemies Isolated From Walls, Non-ideal Orientation

[SV]

Figure 33 shows three positive matches found, which correspond to the three simulated enemies placed in the arena. This indicates that an enemy can be located anywhere in the arena even when the LiDAR can only use half of its field of view.

Preliminary testing has begun with the RPLiDAR A3. Using the manufacturer's demo application, raw data was collected, processed and displayed. The resulting chart, shown in Figure 34, is the LiDAR's 360 degree output when it is spun on the top shelf in ASEC North 525. The outlier samples on the 120 degree heading represent the samples taken through the open door.

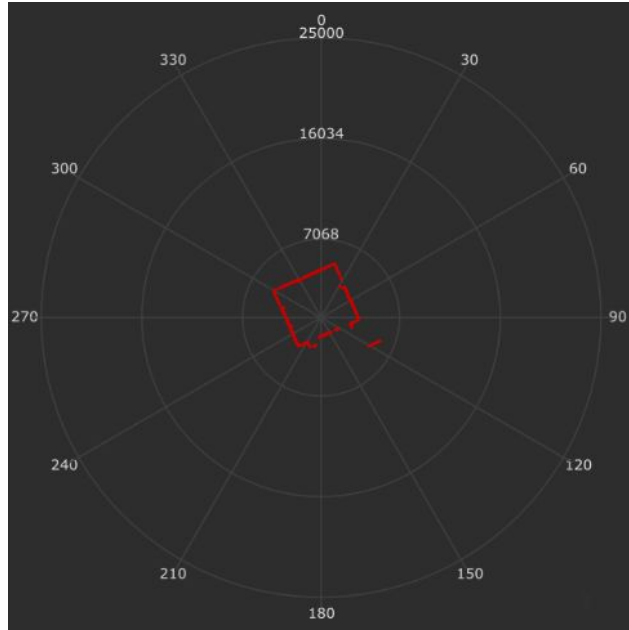


Figure 34: RPLiDAR A3 Measurement Data

[SV]

3.5.3 Data Resolution Calculations

The output of the navigation board will be UART. The maximum amount of data that can be transmitted in a single UART packet is 9 bits. In order for packets to be self-contained, both the speed and direction angle should be incorporated in those 9 bits. Allocating one bit for speed (run or turn in place), the remaining 8 bits will provide an angular resolution of:

$$2^8 = 256 \text{ possible options.}$$

Given that the LiDAR will have a 90° range of vision in each direction, 181 of the 256 possible options will be dedicated to provide an angle, from -90° to 90°, rounded to the nearest degree. The angle data will be encoded in two's complement binary notation.

[FA]

3.5.4 Computing Calculations

The processor needs to accommodate for the sampling frequency of each of the sensors. The baud rates for the sensors are as follows.

Table 27: Baud Rate for Processor Calculations

Sensor	Baud Rates
Lidar	256 kHz
Ultrasonic	500 kHz
Gyroscopic	1 MHz

The minimum processor speed that can be used is

$$\text{minimumprocessorspeed} = 20 \times \max(\text{baudrates}) = 20 \text{ MHz}$$

The PIC32MZEF has a processor speed of 252MHz. The equivalent rate of reading data from a sensor is one fourth of the processor speed. This is due to the architecture of the processor and the cycles required to read data in machine code. This results in our processor able to read a baud/sample rate up to 63 MHz.

[CH, HL]

The Lidar sensor rotates at a frequency of 10 Hz. This means that it passes every degree once every 100 milliseconds. The average enemy robot might be moving at a speed of 10 miles per hour or 14 feet per second. That means that the enemy could have moved 1.4 feet in between samples from the lidar. This means that the autonomous system needs to be able to account for this time. The change in angle can be calculated using Pythagorean theorem.

Figure 35: Distance Traveled Between Samples

If the enemy were to move 1.4 feet in between samples, at a distance of 5 feet the combat robot would see a change in angle of 15.6 degrees.

[CH]

3.5.5 Power, Voltage and Current Calculations

The table below shows the current, voltage, and power for the components of the autonomy system. The maximum power allowed for this system is 144W. As can be seen, the power of the system is within acceptable range.

Table 28: Power Calculation Table

Device	Voltage	Current	Power
Lidar	5	1.5	7.5
Ultrasonic Sensor	3.3	0.035	0.1155
Gyroscope and Accelerometer	3.3	0.004	0.0132
Microcontroller	3.3	0.3	0.99
LEDs	3.3	0.18	0.594
Supporting IC estimate	5	0.16	0.8
		Total	10.0127

From the equation $P = VI$. The total power is 10.01 watts.

The efficiency of most regulators are around 85%, this gives us 11.78 watts.

[CH]

3.5.6 Mechanical Calculations

In order to determine whether the weapon motor is running at half of its maximum RPM, an encoder will be used. The weapon motor is going to be run at 70% the stall torque. The stall torque for the motor is going to be around 3.521 newton meters. This came from the performance chart of the motor. The maximum speed of the weapon will be 1650 rpm. For our algorithm we enter fight mode when the rpm of the motor has reached 825 rpm.

The moment of inertia of the weapon can be estimated as a disk. The radius of the disk is 2.5 inches or 0.064 meters. The weight is 18 pounds or 8.165 kilograms. Therefore

$$\text{momentofinertiaoftheweapon} \approx \frac{1}{2} * \text{mass} * r^2 = \frac{1}{2} * 8.165 * 0.064^2 = 0.0167$$

The torque supplied by the motor then equals the moment of inertia times the angular acceleration.

$$\text{angularacceleration} = \alpha = \frac{\tau}{I} = \frac{3.521}{0.0167} = 210.84 \frac{\text{rad}}{\text{sec}^2}$$

The threshold weapon speed to enter fight mode is 824 rpm. That is equivalent to 86.39 rad/sec.

Therefore it will take 409.7 milliseconds to get to full speed.

$$\text{timehalfofmaximumspeed} = \frac{\frac{(\text{maxspeed})}{2}}{\alpha} = \frac{\frac{86.39}{2}}{210.84} = 409.7 \text{ milliseconds}$$

[HL, CH]

4.0 Schedule

4.1 Calendar View

September

- Weekly meetings will be established at a date and time that is convenient for the majority of members. Items that will be discussed include administration, project progress, barriers to progress and action items to be completed by the next weekly meeting.
- Alternate “work days” will be established where members will work on the design or construction of the robot as a group.
- Reserve room space to store the robot and materials (most likely the design center).
- Have timeline planned out for the project.
- System architecture determined (hardware, mechanical, software)

October

- Preliminary design review.
- Design Calculations
- Electrical
- Mechanical

November

- Critical design review.
- Order all critical components for the robot.
- Hardware design and mechanical design complete
- Software Pseudocode design
- Present Design to department

December

- Fabrication begins
- Software design complete

January

- Preliminary testing
- Control integration

February

- Robot fabrication ends
- Finalize competition travel plans

March

- Final testing
- Code revisions

April

- Compete in the RoboGames

May

- Review and revise results from the competition
- Begin brainstorming ideas for the next year

[CH,AS]

4.2 Gantt Chart Fall Semester

Task Name	Duration	Start	Finish	Resource Names
SDP1 Fall 2018				
Project Design				
Preliminary report	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Cover page	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
T of C, L of T, L of F	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Need	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Objective	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Background	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Marketing Requirements	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Objective Tree	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Block Diagrams Level 0, 1, ... w/ FR tables	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Hardware modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Software modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Mechanical Sketch	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Team information	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
References	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Preliminary Parts Request Form	11 days	Thu 9/6/18	Sun 9/16/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Midterm Report	35 days	Thu 9/6/18	Wed 10/10/18	
Design Requirements Specification	14 days	Mon 9/17/18	Sun 9/30/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Midterm Design Gantt Chart	14 days	Mon 9/17/18	Sun 9/30/18	Holden LeBlanc
Design Calculations	24 days	Mon 9/17/18	Wed 10/10/18	
Electrical Calculations	24 days	Mon 9/17/18	Wed 10/10/18	
Communication	24 days	Mon 9/17/18	Wed 10/10/18	Holden LeBlanc
Computing	24 days	Mon 9/17/18	Wed 10/10/18	Christopher Heldman,Holden LeBlanc
Control Systems	24 days	Mon 9/17/18	Wed 10/10/18	Christopher Heldman,Fabian Ardeljan,Holden LeBlanc,Stephen Veillette
Power, Voltage, Current	24 days	Mon 9/17/18	Wed 10/10/18	Christopher Heldman
Mechanical Calculations	24 days	Mon 9/17/18	Wed 10/10/18	
Weight Requirements	24 days	Mon 9/17/18	Wed 10/10/18	Christopher Heldman
Block Diagrams Level 2 w/ FR tables & ToO	7 days	Mon 9/17/18	Sun 9/23/18	
Hardware modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18	Christopher Heldman,Holden LeBlanc
Software modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18	Fabian Ardeljan,Stephen Veillette

Block Diagrams Level 3 w/ FR tables & ToO	7 days	Mon 9/24/18	Sun 9/30/18	
Hardware modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18	Christopher Heldman, Holden LeBlanc
Software modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18	Fabian Ardeljan, Stephen Veillette
Midterm Design Presentations Part 1	1 day	Thu 10/11/18	Thu 10/11/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Midterm Design Presentations Part 2	1 day	Thu 10/18/18	Thu 10/18/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Project Poster	14 days	Mon 10/8/18	Sun 10/21/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Secondary Parts Request Form	21 days	Mon 9/17/18	Sun 10/7/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Final Design Report	52 days	Mon 10/8/18	Wed 11/28/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Abstract	52 days	Mon 10/8/18	Wed 11/28/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Software Design	31 days	Mon 10/8/18	Wed 11/7/18	
General Communication Psuedocode	31 days	Mon 10/8/18	Wed 11/7/18	Holden LeBlanc
Algorithm Psuedocode	31 days	Mon 10/8/18	Wed 11/7/18	Fabian Ardeljan
Sensor Psuedocode	31 days	Mon 10/8/18	Wed 11/7/18	
Lidar Psuedocode	31 days	Mon 10/8/18	Wed 11/7/18	Stephen Veillette
Gyroscope/Acellerometer Psuedocode	31 days	Mon 10/8/18	Wed 11/7/18	Christopher Heldman
Ultrasonic Psuedocode	31 days	Mon 10/8/18	Wed 11/7/18	Holden LeBlanc
Hardware Design	31 days	Mon 10/8/18	Wed 11/7/18	
Sensor Board	31 days	Mon 10/8/18	Wed 11/7/18	
Schematic	31 days	Mon 10/8/18	Wed 11/7/18	Stephen Veillette
Simulation	31 days	Mon 10/8/18	Wed 11/7/18	Stephen Veillette
Navigation Board	31 days	Mon 10/8/18	Wed 11/7/18	
Schematic	31 days	Mon 10/8/18	Wed 11/7/18	Christopher Heldman
Simulation	31 days	Mon 10/8/18	Wed 11/7/18	Christopher Heldman
Parts Lists	52 days	Mon 10/8/18	Wed 11/28/18	
Parts list(s) for Schematics	52 days	Mon 10/8/18	Wed 11/28/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Materials Budget list	52 days	Mon 10/8/18	Wed 11/28/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Proposed Implementation Gantt Chart	52 days	Mon 10/8/18	Wed 11/28/18	Holden LeBlanc
Conclusions and Recommendations	52 days	Mon 10/8/18	Wed 11/28/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Final Design Presentations Part 1	1 day	Thu 11/8/18	Thu 11/8/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Final Design Presentations Part 2	1 day	Thu 11/15/18	Thu 11/15/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Secondary Parts Request Form	14 days	Thu 10/4/18	Wed 10/17/18	Christopher Heldman, Fabian Ardeljan, Holden LeBlanc, Stephen Veillette
Final Parts Request Form	56 days	Mon	Sun	Christopher Heldman, Fabian Ardeljan, Holden

		10/8/18	12/2/18	LeBlanc,Stephen Veillette
--	--	---------	---------	---------------------------

[HL]

4.3 Gantt Chart Spring Semester

Task Name	Duration	Start	Finish	Predecessors	Resource Names
SDPII Implementation 2018	103 days	Mon 1/14/19	Fri 4/26/19		
Revise Gantt Chart	14 days	Mon 1/14/19	Sun 1/27/19		
Implement Project Design	96 days	Mon 1/14/19	Fri 4/19/19		
Hardware Implementation	56 days	Mon 1/14/19	Sun 3/10/19		
Breadboarding	13 days	Mon 1/14/19	Sat 1/26/19		
LiDAR Breadboard	13 days	Mon 1/14/19	Sat 1/26/19		Steve Veillette
Ultrasonic Breadboard	13 days	Mon 1/14/19	Sat 1/26/19		Holden LeBlanc
Gyroscope/Accelerometer Breadboard	13 days	Mon 1/14/19	Sat 1/26/19		Holden LeBlanc
Layout and Generate PCB(s)	14 days	Sun 1/27/19	Sat 2/9/19	6	Chris Heldman
Assemble Hardware	7 days	Sun 2/10/19	Sat 2/16/19	9	Chris Heldman,Fabian Ardeljan,Holden LeBlanc
Test Hardware	14 days	Sun 2/17/19	Sat 3/2/19	10	Chris Heldman,Holden LeBlanc,Steve Veillette
Revise Hardware	14 days	Sun 2/17/19	Sat 3/2/19	10	Chris Heldman,Holden LeBlanc,Steve Veillette
<i>MIDTERM: Demonstrate Hardware</i>	5 days	Sun 3/3/19	Thu 3/7/19	11	Chris Heldman,Fabian Ardeljan,Holden LeBlanc,Steve Veillette
SDC & FA Hardware Approval	0 days	Fri 3/8/19	Fri 3/8/19	13	
Software Implementation	56 days	Mon 1/14/19	Sun 3/10/19	14	
Develop Software	27 days	Mon 1/14/19	Sat 2/9/19		
Autonomy Software	27 days	Mon 1/14/19	Sat 2/9/19		Fabian Ardeljan
LiDAR Software	27 days	Mon 1/14/19	Sat 2/9/19	17	Steve Veillette
Ultrasonic Software	27 days	Mon 1/14/19	Sat 2/9/19	18	Holden LeBlanc
Gyroscope/Accelerometer Software	27 days	Mon 1/14/19	Sat 2/9/19	19	Holden LeBlanc
Test Software	21 days	Sun 2/10/19	Sat 3/2/19	20	Fabian Ardeljan,Holden LeBlanc,Steve Veillette
Revise Software	21 days	Sun 2/10/19	Sat 3/2/19	17	Fabian Ardeljan,Holden LeBlanc,Steve Veillette
<i>MIDTERM: Demonstrate Software</i>	5 days	Sun 3/3/19	Thu 3/7/19	22	Chris Heldman,Fabian Ardeljan,Holden LeBlanc,Steve Veillette
SDC & FA Software Approval	0 days	Fri 3/8/19	Fri 3/8/19	23	
System Integration	42 days	Sat 3/9/19	Fri 4/19/19		Chris Heldman,Fabian Ardeljan,Holden LeBlanc,Steve Veillette
Assemble Complete System	14	Sat	Fri		Chris Heldman,Fabian

	days	3/9/19	3/22/19		Ardeljan, Holden LeBlanc, Steve Veillette
Test Complete System	21 days	Sat 3/23/19	Fri 4/12/19	26	Chris Heldman, Fabian Ardeljan, Holden LeBlanc, Steve Veillette
Revise Complete System	21 days	Sat 3/23/19	Fri 4/12/19	26	Chris Heldman, Fabian Ardeljan, Holden LeBlanc, Steve Veillette
<i>Demonstration of Complete System</i>	7 days	Sat 4/13/19	Fri 4/19/19	28	Chris Heldman, Fabian Ardeljan, Holden LeBlanc, Steve Veillette
Develop Final Report	99 days	Mon 1/14/19	Mon 4/22/19		
Write Final Report	99 days	Mon 1/14/19	Mon 4/22/19		Chris Heldman, Fabian Ardeljan, Holden LeBlanc, Steve Veillette
Submit Final Report	0 days	Mon 4/22/19	Mon 4/22/19	31	Chris Heldman, Fabian Ardeljan, Holden LeBlanc, Steve Veillette
Spring Recess	7 days	Mon 3/25/19	Sun 3/31/19		
Combat Robotics Competition					
<i>Project Demonstration and Presentation</i>	0 days	Fri 4/26/19	Fri 4/26/19		

[HL]

5. Design Team Information

Fabian Ardeljan, Computer Engineer, Software Lead

Chris Heldman, Electrical Engineer, Team Lead

Holden LeBlanc, Electrical Engineer, Archivist

Stephen Veillette, Electrical Engineer, Hardware Lead

6. Conclusion and Recommendations

The robot design process is well on its way. Most of the components have been chosen and the basic outlines of software have been written. A PIC32 board, a sample ultrasonic sensor, and a LiDAR sensor have been provided for testing. At this point, most of the design has involved rough estimations of values using datasheets, hand calculations, and simulation. The schematic for the sensor and navigation system has been completed. With the pace of the project, the robot is on track to be fully assembled by the end of April.

From this point on, the culmination work of the software and hardware can commence. The hardware system has been designed by creating a Eagle schematic with a single PIC32MZEF microcontroller. It was determined that one microcontroller could handle the operations of our software due to its advanced speed and storage. The schematic includes the connections and support circuits for each sensor, as well as isolation from DT07B's system. LED indicators, a USB interface, and a cardedge system have also been designed.

The base software for the autonomous algorithm has been completed. Some software has already been tested on the Explorer Board, including software that creates a PWM signal and software that communicates over UART. The software for the sensors has yet to be written. However, the LiDAR and current ultrasonic sensor can now be used, and the PIC32 board can be programmed to read and process data from them. Once the gyroscope and encoders arrive, they can be programmed and merged into the project. After all the sensors have software written for them on the Explorer 16/32 board, the custom board with the microprocessor can be programmed using that software. The final task is then to establish UART communications with The Motion and Actualization Team.

7. References

- [1] S. Bachand-Amirault, *Types of Battlebots*. [Online]. Available: <https://sbainvent.com/battlebot-design/battlebot-types.php>. [Accessed: 20-Apr-2018].
- [2] *lidar-uk.com*. [Online]. Available: <http://www.lidar-uk.com/how-lidar-works/>. [Accessed: 20-Apr-2018].
- [3] “What is an Ultrasonic Sensor?,” *Ultrasonic Sensor | What is an Ultrasonic Sensor?*[Online]. Available: http://education.rec.ri.cmu.edu/content/electronics/boe/ultrasonic_sensor/1.html. [Accessed: 20-Apr-2018].
- [4] T. A. Kinney and Baumer Electric, “Proximity Sensors Compared: Inductive, Capacitive, Photoelectric, and Ultrasonic,” *Machine Design*, 13-Mar-2017. [Online]. Available: <http://www.machinedesign.com/sensors/proximity-sensors-compared-inductive-capacitive-photoelectric-and-ultrasonic>. [Accessed: 20-Apr-2018].
- [5] W. by AZoSensors, “What is a Photoelectric Sensor?,” *AZoSensors.com*, 15-Jun-2015. [Online]. Available: <https://www.azosensors.com/article.aspx?ArticleID=311>. [Accessed: 20-Apr-2018].
- [6] “Photoelectric sensors,” *Balluff*. [Online]. Available: <https://www.balluff.com/local/us/products/sensors/photoelectric-sensors/>. [Accessed: 20-Apr-2018].
- [7] “CN107140047A - Competitive foot combat robot,” *Google Patents*. [Online]. Available: <https://patents.google.com/patent/CN107140047A/en?q=competition&oq=combat+robot+competition>. [Accessed: 20-Apr-2018].
- [8] “WO2017143567A1 - Fighting robot,” *Google Patents*. [Online]. Available: <https://patents.google.com/patent/WO2017143567A1/en?q=competition&oq=combat+robot+competition>. [Accessed: 20-Apr-2018].
- [9] “KR20050055822A - The weapon system of a battlebot using pneumatic circuit,” *Google Patents*. [Online]. Available: <https://patents.google.com/patent/KR20050055822A/en?q=robot&oq=competition+combat+robot&page=2>. [Accessed: 20-Apr-2018].

- [10] V. Magnier, D. Gruyer and J. Godelle, "Automotive LIDAR objects detection and classification algorithm using the belief theory," *2017 IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, CA, 2017, pp. 746-751. <https://ieeexplore.ieee.org/document/7995806/>
- [11] A. Börcs, B. Nagy and C. Benedek, "Instant Object Detection in Lidar Point Clouds," in *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 7, pp. 992-996, July 2017.. Available: <https://ieeexplore.ieee.org/document/7927715/>
- [12] Renishaw. (2018). *Renishaw: Application note: Optical encoders and LiDAR scanning*. [Online] Available: <http://www.renishaw.com/en/optical-encoders-and-lidar-scanning--39244> [Accessed 25 May 2018].
- [13] Wong, W. (2018). *Safe Robots Rely On Sensors*. [Online] Electronic Design. Available: <http://www.electronicdesign.com/embedded/safe-robots-rely-sensors> [Accessed 25 May 2018].
- [14] P. Agharkar and F. Bullo, "Vehicle routing algorithms to intercept escaping targets," *2014 American Control Conference*, Portland, OR, 2014, pp. 952-957. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6858759>
- [15] *Combat Robot Rules*. [Online]. Available: <http://robogames.net/rules/combat.php>. [Accessed: 20-Apr-2018].
- [16] SLAMTEC RPLIDAR A3 data sheet. [Online]. Available: http://bucket.download.slamtec.com/aaf96dddba2f6a9baa03261628c01af9fc2f866c/LD310_SLAMTEC_rplidar_datasheet_A3M1_v1.0_en.pdf [Accessed: 10-October-2018].

Appendix

LiDAR Datasheet

http://bucket.download.slamtec.com/aaf96dddba2f6a9baa03261628c01af9fc2f866c/LD310_SLAMT_EC_rplidar_datasheet_A3M1_v1.0_en.pdf

Gyroscope Datasheets

<https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>

Ultrasonic Datasheet

<https://www.parallax.com/sites/default/files/downloads/28015-PING-Sensor-Product-Guide-v2.0.pdf>

PIC32MZEFH100

<http://ww1.microchip.com/downloads/en/DeviceDoc/60001320E.pdf>

Honors Project Final Design Report

Design Project: Autonomous Combat Robot

DT07B

Andrew Szabo

Tanya Tebcherani

Tristin Weber

Dr. French

April 25, 2019

Table of Contents

Abstract	7
1. Problem Statement	8
1.1 Need	8
1.2 Objective	8
1.3 Background	11
1.3.1 Research Survey	11
1.3.2 Proposed Design Overview	17
1.4 Marketing Requirements	20
1.5 Objective Tree	21
2. Design Requirements Specification	22
3. Accepted Technical Design	23
3.1 Design Calculations	23
3.2 System Overview	25
3.2.1 Hardware Overview	25
3.2.1.1 Hardware Block Diagrams	26
3.2.1.1.1 Level 0 Hardware Block Diagram with Functional Requirement Table	26
3.2.1.1.2 Level 1 Hardware Block Diagram with Functional Requirement Table	27
3.2.1.1.3 Level 2 Hardware Block Diagram with Functional Requirement Table	29
3.2.1.2 Hardware Overview and Schematics	35
3.2.2 Software Overview	45
3.2.2.1 Software Flowcharts	46
3.2.2.1.1 Level 0 Software Flowchart	46
3.2.2.1.2 Level 1 Software Flowchart	47

3.2.2.2 Pseudocode	50
3.2.3 Mechanical Overview	60
4. Parts List	62
5. Project Schedules	65
6. Design Team Information	69
7. Conclusion and Recommendations	70
8. References	71
9. Appendices	74
9.1 Software Code	74
9.1.1 Header Files	74
9.1.1.1 config.h	74
9.1.1.2 defines.h	75
9.1.1.3 functions.h	76
9.1.2 Source Files	78
9.1.2.1 Auto_Mode.c	78
9.1.2.2 IC1_y.c	80
9.1.2.3 IC2_x.c	82
9.1.2.4 IC3_w.c	85
9.1.2.5 IC4_estop.c	87
9.1.2.6 IC5_am.c	89
9.1.2.7 IC_Misc.c	91
9.1.2.8 LEDs.c	92
9.1.2.9 Man_Mode.c	93
9.1.2.10 Misc.c	94

9.1.2.11 newmain1.c	96
9.1.2.12 OC1_RDM_PWM.c	102
9.1.2.13 OC2_LDM_PWM.c	104
9.1.2.14 OC3_WM_PWM.c	106
9.1.2.15 UART4.c	108
9.2 Board schematics	109
9.2.1 Control Board schematics	109
9.2.2 Power Board schematics	111
9.3 Useful links	112
9.4 Pictures	113

List of Figures

Figure 1: Combat Robot Objective Tree	20
Figure 2: Level 0 Hardware Block Diagram	25
Figure 3: Level 1 Hardware Block Diagram	26
Figure 4: Level 2 Hardware Block Diagram for Control System	28
Figure 5: Level 2 Hardware Block Diagram for Power System	30
Figure 6: Level 3 Hardware Block Diagram for Control System	33
Figure 7: Control System Hardware Schematic - Backplane Connector	35
Figure 8: Control System Hardware Schematic - Microprocessor	36
Figure 9: Power System and Backplane Schematic - Voltage Regulators	38
Figure 10: Power System and Backplane Schematic - Connector	39
Figure 11: Backplane Connector Layout	42
Figure 12: Level 0 Software Flowchart	45
Figure 13: Level 1 Software Flowchart	46
Figure 14: Level 1 Weapon Control Flowchart	47
Figure 15: Level 1 Motion Control Flowchart	48
Figure 16: Mechanical Design - Isometric View	60
Figure 17: Mechanical Design - Planar View	60
Figure 18: Gantt Chart Fall 2018	64
Figure 19: Gantt Chart Spring 2019	65

List of Tables

Table 1: Comparison of Combat Robot Types	11
Table 2: Design Requirements Specification Table	21
Table 3: Level 0 Functional Requirement Table	26
Table 4: Level 1 Robot Control Center Functional Requirement Table	27
Table 5: Level 1 RGB LED Status Indicator Functional Requirement Table	27
Table 6: Level 1 Combat Robot Chassis and Weapon System Functional Requirement Table	27
Table 7: Level 2 RC Receiver Board Functional Requirement Table	29
Table 8: Level 2 Microcontroller Hardware Functional Requirement Table	29
Table 9: Level 2 Motor Controller Hardware Functional Requirement Table	29
Table 10: Level 2 LED Status Indicators Functional Requirement Table	30
Table 11: Level 2 Battery and Charger System Functional Requirement Table	31
Table 12: Level 2 Central Power Board Functional Requirement Table	31
Table 13: Level 2 Weapon and Drive Motor Controllers Functional Requirement Table	31
Table 14: Level 2 Motor Functional Requirement Table	32
Table 15: LED Indicators	34
Table 16: Control System Signals	37
Table 17: Control Board BOM	61
Table 18: Power Board/Backplane BOM	62
Table 19: General Electrical System BOM	63
Table 20: Week by Week Updates (9/23 - 11/20)	67

Abstract

The purpose of this project is to design and build a 60 lb. weight class autonomous combat robot that will participate in the international RoboGames competition. Being autonomous will allow the combat robot to outperform manually driven robots during competition. The chosen design is a dual wedge robot which has a parallelogram shape. There is a wedge on the front of the robot as well as another wedge on the back. In the case the robot is flipped upside down, it can still fight due to the wedge on the back of the robot. In order to be in compliance with RoboGames rules, the robot must be able to be controlled manually.

[AS]

1. Problem Statement

1.1 Need

Combat robotics is a discipline that requires much skill and a quick response time. It is often the case that the winner is not the best robot, but rather the best operator. Human operators inherently lack a consistent, fast reaction time when using a remote controlled system for combat robots. Human operators also have difficulty keeping up with the fast decision making necessary to maneuver their combat robots. It would be much faster and more effective for a combat robot to operate independently of human controls. Autonomous control of the operation and locomotion of a combat robot would outperform a manual operator.

A fully autonomous system has the ability to make algorithmic decisions, follow a locomotion algorithm, and attack with more precision than a manual operator. Therefore, an autonomous combat robot is needed to outperform opponents in movement and weapon reaction time.

[AS, FA, CH]

1.2 Objective

The objective of this project is to design and build a 60 lb weight class combat robot that will function autonomously and outperform the manually driven robots during competition.

While running autonomously, the robot will use LiDAR sensors to detect its environment and the opponent, and will attack the opponent when possible. This robot will also be able to be remote controlled in manual mode. This will mitigate the risk in case the autonomy or sensors fail. LED lights on the robot will indicate whether it is in full autonomous or manual override

mode. The system will also be able to be armed and disarmed remotely, even while in autonomous mode. Lastly, the robot will incorporate an emergency shut off and braking system.

The autonomous combat robot will outperform its human driven opponents by following a variety of combat algorithms. The autonomous system will follow an intercept or escape locomotion pattern to outperform the human operators. While the weapon systems are reaching full speed, the combat robot will follow a avoidance and escape algorithm. When the weapon system is ready the robot will follow an intercept algorithm to attack the opponent. The autonomous system will also attempt to keep the robot pointed in the correct orientation, facing the opponent at all times.

This project is a robotic system. It will have heavy reliance on electrical, software, and mechanical subsystems. The design team for this project consists of five electrical engineering students, two computer engineering students, and three mechanical engineering students. These students will be split into three teams - sensing and autonomy (ECE Team 7A), motion and actualization (ECE Team 7B), and mechanical and structural (ME Team). While these teams will have separate deliverables and responsibilities, they will coordinate closely with each other to maintain a coherent end project. A brief breakdown of the teams and their roles is given below.

[TW, CH]

Role of Electrical and Computer Team 7A

- Create the control, feedback, and sensing system to control the combat robot
- Implement opponent facing tracking algorithm
- Implement control intercept or escape algorithm
- Create LiDAR sensing data interpretation and detection programming and algorithm
- Autonomous sensing and control

Role of Electrical and Computer Team 7B

- Convert signal from Team 7A's control algorithms into movement instructions
- Control of electric motor system
- Control of weapon system
- Create the power system to run the motors and robotic system
- Meet Robogames electrical safety standards (emergency stop, manual control, etc.)

Role of Mechanical Team¹

- Create the robot's chassis
- Create the mechanical weapon system
- Create a mechanical drive system

[CH, TW]

¹ The mechanical engineering team has obtained preliminary approval to work with the electrical and computer engineering team from the Mechanical Engineering Department, and is in the process of receiving full approval upon the approval of the ECE's final proposal. The mechanical engineering team will synchronize its deadlines with the ECE team's deadlines in completion of their requirements with the project.

This document will focus on ECE Team 7B, which is the motion and actualization team. The role of this team will be to design an embedded system, embedded system, control system, and general electrical system that can turn the processed sensing data from ECE Team 7A and translate it into motion via a decision algorithm, power supply, and motor controller interfaces. Team 7B will also be responsible for implementing manual control via interfacing with an RC controller and meeting various other Robogames regulations such as emergency stops, visual status indicators, and electrical safety.

[TW]

1.3 Background

The following sections provide a background and general overview of the combat robot design approach.

[TT]

1.3.1 Research Survey

Currently, the vast majority of combat robots operate by being remote controlled by a operator. Autonomy requires additional financial investments and far more work. However, the benefits of these investments are well worth the cost for first time contenders facing operators with many years of experience.

The explicit goal of combat robotics is to immobilize the opponent robot before it can do the same to one's own robot. There are many ways to accomplish this, from attacking with blunt force, to trying to impale key mechanisms of the robot, to lifting and getting the opponent robot stuck in an immobile position. The key to a successful combat robot is having both powerful offensive and defensive strategies.

Table 1 below shows a trade off analysis of the typical combat robotics weapon systems.

This was used as a research tool to determine the best combat weapon system for the autonomous robot.

Style	Pros	Cons
Wedge	<ul style="list-style-type: none"> - Structural integrity provides an excellent defense - Simple design - Able to get under an opponent to drive them into the wall or other hazards - Can incorporate other design features 	<ul style="list-style-type: none"> - Must have a skilled operator - Weak offense - Weak matchup against other wedges - Some competitions have banned combat bots that only use wedges
Spinner	<ul style="list-style-type: none"> - Weapon serves as both offense and defense - Low skill floor for operator 	<ul style="list-style-type: none"> - Potential to damage itself - Difficult to design - Difficult to upgrade with additional features
Drum	<ul style="list-style-type: none"> - High destructive potential - Allows for a sturdy frame - Room for additional features 	<ul style="list-style-type: none"> - Difficult to control
Crusher	<ul style="list-style-type: none"> - Potential to cause structural damage via blunt force - Allows for a sturdy frame 	<ul style="list-style-type: none"> - Requires a skilled operator to operate manually - Hard to stay within weight limits
Flipper	<ul style="list-style-type: none"> - Potential to flip other robots over for damage or immobilization - Room for additional features 	<ul style="list-style-type: none"> -Requires a skilled operator to operate manually -Weapon presents a vulnerable spot -Pneumatics limited by air tank size
Hybrid	<ul style="list-style-type: none"> - Flexibility and ability to have multiple weapon systems 	<ul style="list-style-type: none"> -Complex design -Hard to keep within weight restrictions

Table 1: Comparison of Combat Robot Types [1]

[TT]

The proposed combat robot will be operated autonomously, with an option to be controlled manually if desired. A major component in creating autonomous robots is sensing their surrounding environment. Although the sensor(s) that will be used for this project has yet to be selected, the combat bot will most likely use existing sensors for automation purposes. Three sensors will be discussed for this project: LiDAR sensors, ultrasound sensors, and photoelectric sensors.

LiDAR (light detection and ranging) sensors use light in the form of pulsed lasers to detect an object's distance from the sensor. When it shoots a laser out, the laser will hit an object and that object will reflect the laser back to the sensor. The sensor measures the time it takes to receive the reflection and by using the speed of light, it then calculates the distance of the object is from the sensor. LiDAR sensors shoot out approximately 150,000 pulses per second, so they can quickly build a "map" of their surroundings [2]. This seems like the ideal sensor for the combat robot because it can provide a map of the area and it will easily identify the opponent from the point cloud data.

An ultrasound sensor is similar to a LiDAR sensor in that it measures its distance from an object, but uses sound instead of light. It sends out a sound wave at a specific frequency and waits for the wave to hit an object and bounce back. Based on the time the sound wave takes to bounce back and the speed of sound, the ultrasound sensor can determine its distance from the object the sound wave hit. This sensor would not work if it were to hit an object that deflects the sound wave instead of returning it to the sensor, or if it hit something that was made of a material that absorbs sound instead of reflecting it [3]. This is why it would not be a good choice.

Photoelectric sensors are used extensively in the field of automation. They can detect objects as small as 1mm in diameter, and objects as far as 60m away [4]. There are many different types of photoelectric sensors, but all of them use a light transmitter, often infrared, and receiver. Depending on the type of sensor used, one of two things will happen. 1) When the infrared signal hits an object, it is reflected back to the signal and is received by a photoelectric receiver. 2) When the infrared signal hits an object, the object breaks the light beam, and an object can be sensed in that way instead of by its reflection. Photoelectric sensors can be used to not only detect the object's distance from the sensor, but also the object's color, size, shape, presence, among other features [5, 6]. The drawback of these sensors is they are used in a yes or no configuration. They are generally used to tell if a object is directly in front of the sensor, not to 3d map or sense the environment.

[TT]

Another potential sensor would be the Microsoft's Kinect sensor, which used infrared sensors. Infrared sensors are good for detection between 1-5 meters or 3-15 feet. The arena is 40 feet by 40 feet. Therefore, infrared sensors will not be sufficient for the competition. Microsoft's Kinect system uses infrared so the system being used will need a more novel sensor design. After further research a LiDAR system seems to be the best choice due to its high range and accurate position sensing capabilities, while still being cost effective.

[AS]

Almost all combat robots at robogames are manually-controlled, which means their effectiveness in competition is severely limited by the skill level and reaction time of their operator. This flaw in current designs exists because the algorithms and sensors needed to have a

competitive combat robot are quite complex. If one were to write such an algorithm that could effectively perform combat maneuvers and pair it with an adequate sensor array and a robust, reliable combat robot, the resulting combination could yield a very competitive end product.

While there was a senior design team during the 2017-2018 school year that constructed an autonomous combat robot, it was a very complex and heavy (220 lb) design, and thus encountered issues with reliability. In addition, this contributes to limited autonomous capabilities, which are restricted to simple fight or flight. While their design was excellent, there is room for further improvement. With a lighter, simpler design, the proposed combat robot could autonomously perform more advanced maneuvers such as, intercept and escape maneuvers, attacking with the wedge and drum, and keeping the robots wedge always facing the opponent to defend against their weapon.

This bot is similar to existing designs because it still has all of the components of a traditional combat bot. It will have a weapon as well as all of the required electrical engineering components. For example, it will have motors, actuators, controls, power supplies, programming, wireless communications, etc. It is also similar to other self-driving (autonomous) vehicles because it will use LiDAR sensors to accomplish autonomy. Note that, although LiDAR is constantly used for self-driven vehicles, autonomous combat bots are not the standard. Most combat bots have an operator who controls the bot during a fight. Thus, making the bot autonomous is different from existing technologies.

[TT]

There are current patents on robotic systems that are similar to the one in which the team will create. Some interesting ideas can be drawn from these innovative patents.

One of these was a patent on a combat robot which used a walking system instead of the conventional wheels. This was proposed because many combat robots fail because they lose mobility due to their fragile rubber wheels being destroyed. The weapon system on this robot was of the flipping kind [7].

Another patent of interest is a combat robot which uses infrared emitters to sense its surroundings and autonomously detect and attack the opponent robot. This robot was a wedge combat design. The design mitigates the weakness of external vulnerable wheels by having them enclosed within its chassis [8].

A further patent found was on a flipping- wedge hybrid robotic combat system. This patent only discussed the weapon system design. The robot was of a wedge shape, and could function as a wedge combat robot to attack other bots by flipping them over and preventing their mobility. (If the wheel system was on the bottom of their robot). The more interesting weapon of this system was a flipping arm. The combat robot would have an arm which could reach under other robots and then lift at high velocity to send the opponent in the air. This system was driven by pneumatic circuits. It would use a compressed gas tank to drive the arm with a very large torque to flip the other robot [9].

[CH]

1.3.2 Proposed Design Overview

For offense, the combat robot will use drum method, which involves an upward spinning horizontal drum with extensions used to hit and possibly throw the opponent robot. This drum will be in combination with a wedge. Hybrid combat robots are often not used because of complexity of design but having a mechanical subteam will allow the team to make a more complex and effective weapon system. Most notably, the drum method is also used by Touro Maximus, a long time contender and multiple time finalist in combat robotics tournaments. The wedge method was used by the winning robogames combat robot Original Sin. By combining these two weapon systems and having a fully autonomous robot, it can outperform its opponents. By using autonomy, this method can be further enhanced to make sure the weapon system turns to face the opponent as fast as possible, and can follow ideal intercept and escape paths.

On the defensive side, it is a clear advantage to have a robot that can withstand being flipped. Robots that can operate on their back usually recover more effectively from being tossed as well. However, the design should also focus on a hard outer shell that resists damage to both blunt and sharp attacks. Autonomy can also help defensively, by allowing the robot to sense in all directions where the opponent is and turn to face the opponent with the weapon system before the opponent can strike from behind or from the side.

[FA]

The team has found that LiDAR is the best sensing technology for automated driving of combat robots. In 3D applications, it uses a laser beam to scan the environment with very high accuracy, and as a result is highly suited for estimating shapes of objects [10]. The data returned by LiDAR can be interpreted as a 3D point cloud, where clusters of detected points form objects.

The robot's algorithm will accept this 3D point cloud and divide it into foreground and background layers [11]. To determine if the opponent robot is in range, it will analyze the surface of the foreground layer and determine if it is a flat wall or an opponent based on the curvature of the point cloud. This method has already been tested using Microsoft XBox Kinect sensors by the former University of Akron Combat Robotics team, and has shown promising results. Internally, LiDAR works by using a rotating mirror system to take panoramic picture data. It is controlled by motors and rotary encoders that determine the tilt of the mirrors used to take the pictures [12]. This mechanical complexity along with the large amount of data processing make this very expensive. Smaller, cheaper versions are used in mobile robots such as Aethon's autonomous TUG robots used in hospitals [13].

The goal of the autonomous system is to drive the robot better than a user would. Higher complexity locomotion algorithms will be used to give the combat robot a significant edge above all manual robots. This will be done by implementing two algorithms for movement. The first being an intercept or escape algorithm. The second being an algorithm that utilizes the robot's weapon design, which will attempt to keep the defensive wedge pointed at the opponent at all times.

The intercept or escape algorithm will determine if the robot should be avoiding or attacking the opponent. This will be dependent on if the chosen drum is at an optimal velocity and the robot is in optimal health. If the robot's drum weapon is at an acceptable speed the team will enter the intercept dynamics of the robot, and if the drum is still speeding up the robot will enter escape. During intercept the robot will use the current and previous position of the enemy to determine its direction and position. It can then use the intercept algorithm to attack the

opponent. Instead of using a simple tracking algorithm and traversing directly toward the opponent, the robot will use a algorithm based on vector mathematics. The algorithm will command the robot to travel to where the opponent will be, not where the opponent is. This similar to a predator catching prey to quickly intercept the opponent. This algorithm will succeed against a manual operator of an opposing robot attempting to flee [14].

The opponent facing tracking algorithm will implement a defensive strategy used by human wedge operators. The strategy is to keep the wedge always facing the opponent. This will cause the attacking opponent to drive on top of the wedge, and potentially flip over, when attempting to attack. When driving on top of the autonomous combat robot, the attacking robot's weapon system will miss the autonomous robot and give it a opening to attack using the drum weapon.

[CH]

Team 7B will receive an angle from Team 7A as well as an intercept/escape signal. This information and the information from the RC controllers will be used to determine if the combat bot should operate in manual or autonomy mode. Once the mode of operation is determined, the data from Team 7A or the data from the RC controller will be used to determine what to do to the motors to achieve those instructions. This will involve a control system design to achieve motion of the motors specific to angles, directions, and intercept/escape.

[TT]

1.4 Marketing Requirements

The marketing requirements for team 7B's part of the the combat robot system are as follows:

1. The robot shall have an internal, rechargeable power system for motor control and sensor operation.
2. The robot shall indicate when it is in autonomous mode.
3. The robot shall operate for the duration of the match.
4. The robot shall have a manual control mode as per robogames rules.
5. The robot shall have an emergency stop for all weapon and movement systems
6. The robot shall accept autonomous algorithm output signals from team 7A's system and turn them into motion.

[AS, TT, TW]

1.5 Objective Tree

The objective tree for the fully autonomous combat robot is shown in Figure 1 below.

This was derived from the marketing requirements.

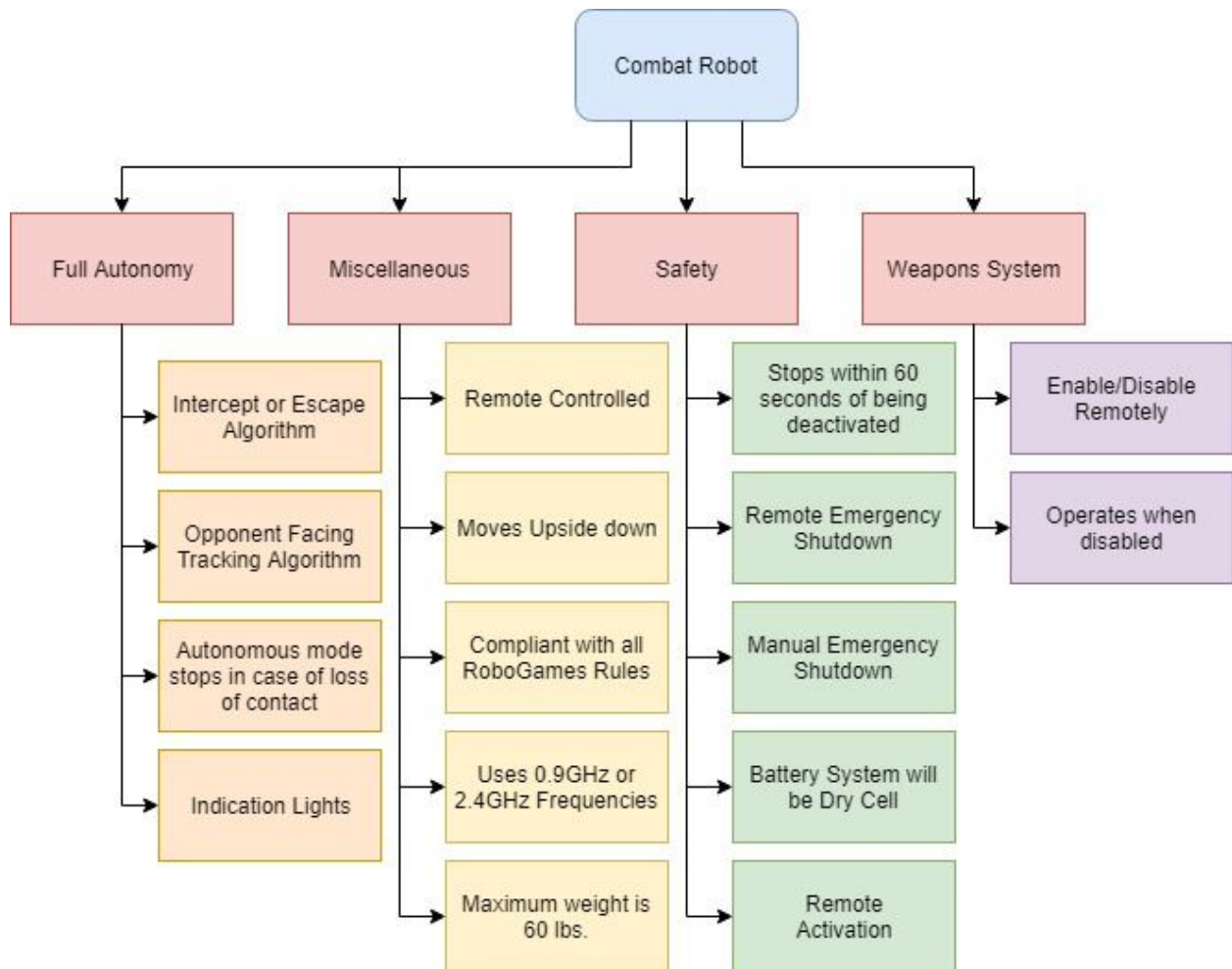


Figure 1: Combat Robot Objective Tree

[AS, TT]

2. Design Requirements Specification

Table 2 contains the engineering requirements along with the marketing requirement(s) they correspond to and the justification for why each engineering requirement is necessary.

Marketing Requirements	Engineering Requirements	Justification
1,3	1. Internal power system should be able to run the robot for at least 4 minutes under full load.	Duration of the match is 3 minutes.
1, 5	2. The power system shall have overcurrent protection that trips at 110% of the battery's maximum impulse amp rate.	Exceeding current limits of the battery could be an electrical hazard and cause catastrophic failure.
1	3. The weight of the battery and electronic components designed and used shall not exceed 10 lbs.	Contributes to weight requirement of 60lbs.
1, 3	4. The power system shall be able to provide the motors and microcontrollers with enough power to all run simultaneously at top speed.	Needed to run the robot.
2	5. The robot shall indicate when it is in autonomous mode with a visual indicator.	This is a requirement of RoboGames.
4, 5	6. An emergency stop signal from the RC controller shall be used to stop all motion of the robot within 60 seconds.	This is a requirement of RoboGames. It will also ensure safety when testing.
4	7. The robot shall have a manual control mode, operated with an RC controller.	This is a requirement of RoboGames.
4	8. The robot shall not start in autonomous mode when first powered on.	Safety
6	9. Team 7B's embedded system shall be able to communicate with team 7A's microcontroller(s).	This is necessary for the robot to be able to move autonomously.
6	10. The weapon motor shall be able to reach a rotation speed of 2500 rpm.	This is needed in order for the robot to do damage to an opposing robot.

Table 2: Design Requirements Specification Table

[AS, TT, TW]

3. Accepted Technical Design

The sections below describe the system design.

[TT]

3.1 Design Calculations

The following equations govern the basic design of the power system:

$$V = I * R$$

$$P = V * I = (I^2) * R = (V^2) / R$$

$$AH = A * Hours$$

$$WH = W * Hours$$

Selecting an appropriate capacity battery package is appropriate for the system. A set of cells should have enough AH to run the system for at least 4 minutes to meet the established design requirements. The capacity needed can be calculated from the following equations:

$$\textit{Total Current Draw} = \textit{Sum of All Currents}$$

$$AH = (\textit{Total Current Draw}) * (4 \textit{ minutes}) * \frac{(1 \textit{ hour})}{(60 \textit{ minutes})}$$

where AH is the desired battery capacity in Amp-Hours.

Making some approximations, it is noted that roughly ~39A will be needed to run the system at full-speed. This means that a capacity of just over 2.5 AH is needed to run for four minutes.

The following calculations were used to determine the selection of the motors for the robot:

To begin with, required speed for the motors were found using this formula:

$$\text{Robot Speed [Mph]} = \frac{(\text{Distance Per Wheel Rotation}) * (\text{Motor Top Speed})}{(\text{Gearbox Gear Ratio})}$$

After conversion factors were accounted for, it was determined that in order to travel at a speed of 10mph, the drive motors had to turn at about 5000 rpm.

The next step was to consider acceleration of the robot. The following equations were used:

$$\sum \text{Force on Wheels} = \frac{2 * (\text{Motor Torque}) * (\text{Motor Speed}) * (\text{Conversion of Units})}{(\text{Gearbox Gear Ratio})}$$

$$\text{Robot Acceleration [ft/s}^2\text{]} = (\sum \text{Force on Wheels}) * (\text{Mass of Robot}) * (\text{Conversion Factor})$$

Assuming a potential motor torque of 85 oz-in (motor decided later), the robot could reach top speed in about 4 seconds from a standstill. This was deemed acceptable.

Using these calculations, it was deemed necessary to find drive motors with a nominal speed of at least 5000 rpm and torque of 85 oz-in.

Lastly, in order to control the speed of these motors, a DC voltage of variable amplitude had to be applied to their terminals. This is achieved by generating a PWM with a motor controller. For all intents and purposes, the DC voltage across a motors terminals is equal to the average voltage across the terminals. The average voltage across a motor is determined by the following equation:

$$V_{Avg} = (\text{PWM Duty Cycle}) * V_{max}$$

In this case, the amplitude of the PWM from the motor controller, $V_{max} = 24V$ (the rail). A motor controller had to be selected that could achieve a duty cycle from 0% - 100% to allow for variable speed control. The motor controller would also have to be able to output a maximum amplitude of 24V to allow the motors to run at top speed. Note that this PWM being output by the motor controller will have the same frequency and duty cycle (D) as the PWM being sent into the motor controller, but it will have a much higher amplitude. Using this principle, the speed of a motor can be varied by varying the duty cycle of the PWM signal being sent to the motor controller.

[TW]

3.2 System Overview

The sections below present an overview to the hardware, software, and mechanical systems of the combat bot.

[TT]

3.2.1 Hardware Overview

RoboGames competitions put a heavy strain on the electrical systems of any combat robot, especially the motors. Due to the sensitivity of the electronics needed to support autonomy, a robust, reliable, and interference-free electrical system is required to ensure peak performance and continued operation even when damage is sustained throughout the competition.

The hardware system shall include a set of batteries with enough capacity to run the robot under a heavy load for the entirety of an intense 3 minute round of combat. The system will also contain 24V, 5V, and 3.3V rails to supply power to both Team 7B and Team 7A's

electronics. There will also be proper overcurrent protection in place to prevent critical failure.

The below block diagrams represent an overview of the hardware design of the combat bot.

[TW]

3.2.1.1 Hardware Block Diagrams

The below sections show the block diagrams of the hardware system.

[TT]

3.2.1.1.1 Level 0 Hardware Block Diagram with Functional Requirement Table

The level 0 block diagram indicates the top-level inputs and outputs of the fully autonomous combat robot. Figure 2 shows the level 0 diagram for DT07B's part of the autonomous combat robot.

[CH]

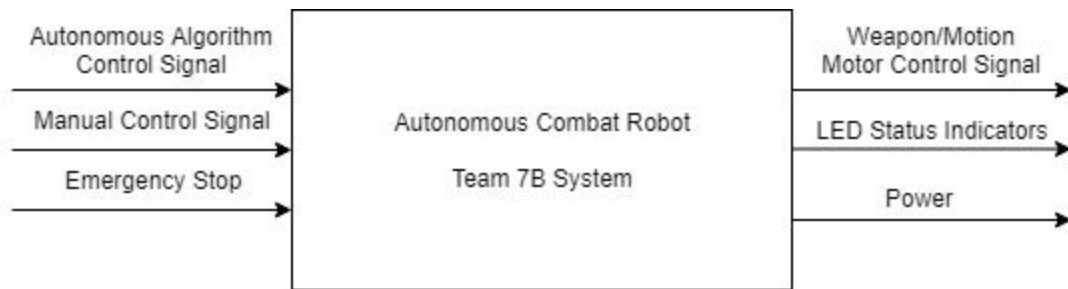


Figure 2: Level 0 Hardware Block Diagram

[TT, TW]

The level 0 functional requirement table, indicating the top-level inputs and outputs of the fully autonomous combat robot, as shown in Table 3 below.

Module	Combat Robot
Inputs	<ul style="list-style-type: none"> • Autonomous control algorithm signals from DT07A’s system • Manual Control Signal • Emergency Stop
Outputs	<ul style="list-style-type: none"> • Motion Motor Control Signals • Weapon Motor Control Signals • LED Status indicators • Power
Functionality	Accepts a signal from DT07A’s autonomous control algorithm and translates it into motor control for motion and weapon motors. Also accepts manual control signal from handheld controller, should autonomous mode fail. Will use LEDs to indicate manual or autonomous mode. Supplies power to DT07A.

Table 3: Level 0 Functional Requirement Table

[FA, CH, AS, TT, TW]

3.2.1.1.2 Level 1 Hardware Block Diagram with Functional Requirement Table

The level 1 block diagram is an expansion of the level 0 diagram. Figure 3 shows the level 1 diagram for DT07B’s part of the autonomous combat robot.

[TW]

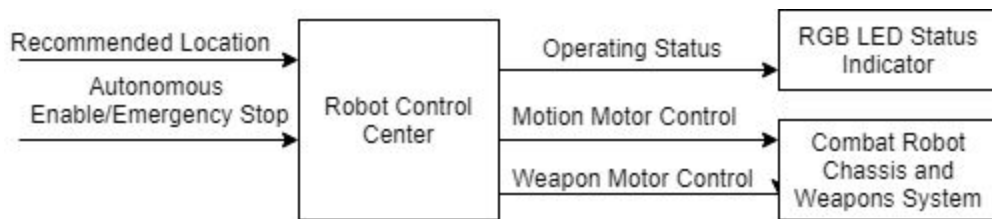


Figure 3: Level 1 Hardware Block Diagram

[AS]

The level 1 functional requirement table, indicating the second-level inputs and outputs of the fully autonomous combat robot, as shown in Tables 4, 5, and 6 below.

Module	Robot Control Center
Inputs	<ul style="list-style-type: none"> ● Fight or Flight Signal ● Autonomous Enable/Emergency Stop
Outputs	<ul style="list-style-type: none"> ● Operating Status ● Motion Motor Control ● Weapon Motor Control
Functionality	Controls motor movement.

Table 4: Level 1 Robot Control Center Functional Requirement Table

Module	RGB LED Status Indicator
Inputs	<ul style="list-style-type: none"> ● Operating Status
Outputs	<ul style="list-style-type: none"> ● LED light
Functionality	LED indicates robot operation status by color.

Table 5: Level 1 RGB LED Status Indicator Functional Requirement Table

Module	Combat Robot Chassis and Weapon System
Inputs	<ul style="list-style-type: none"> ● Motion Motor Control ● Weapon Motor Control
Outputs	<ul style="list-style-type: none"> ● Chassis/Weapon Feedback
Functionality	Controls start up, shut down, and movement of weapon.

Table 6: Level 1 Combat Robot Chassis and Weapon System Functional Requirement Table

[AS, TT]

3.2.1.1.3 Level 2 Hardware Block Diagram with Functional Requirement Table

The level 2 block diagrams are an expansion of the level 1 diagram. Figure 4 shows the level 2 hardware block diagram for the embedded system of the autonomous combat robot.

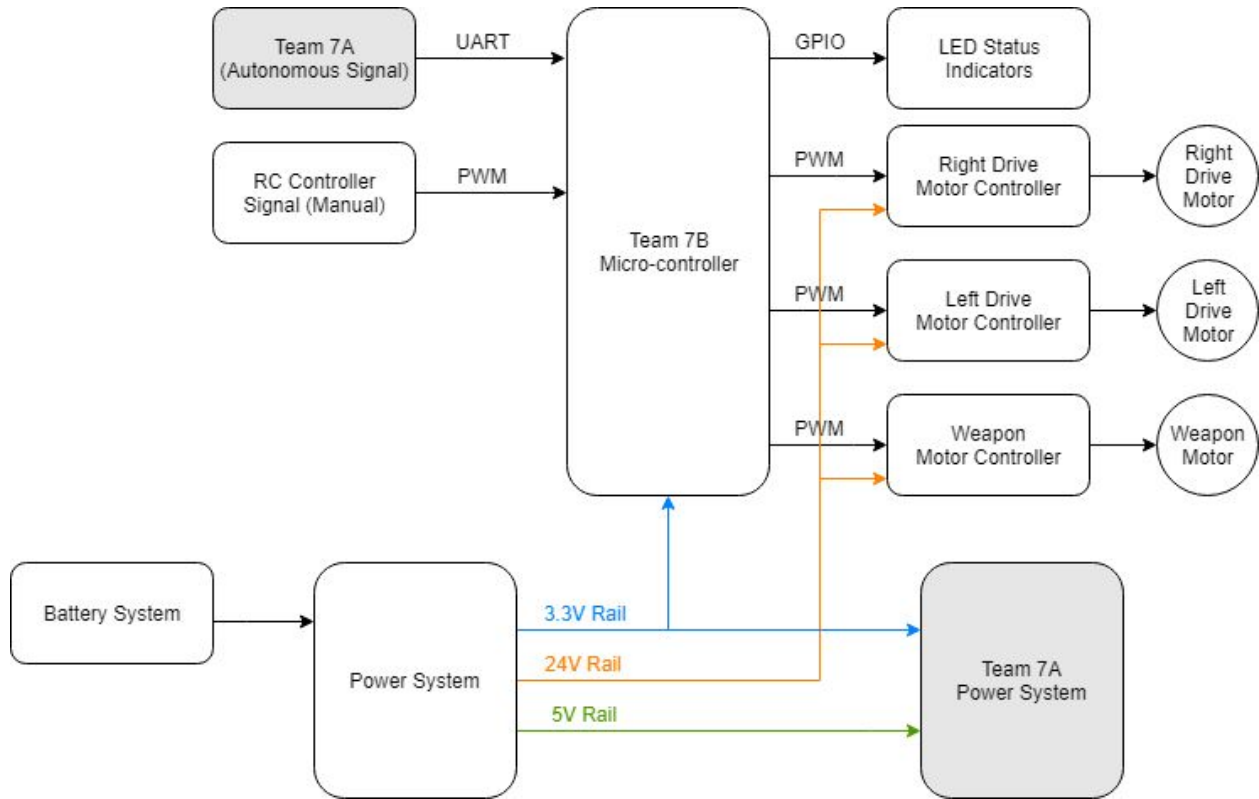


Figure 4: Level 2 Hardware Block Diagram for Control System

[TW]

The level 2 functional requirement table, indicating the third-level inputs and outputs of the fully autonomous combat robot, are shown in Tables 7, 8, 9, and 10 below.

Module	RC Receiver Board
Inputs	<ul style="list-style-type: none"> • Wireless Signal
Outputs	<ul style="list-style-type: none"> • PWM Signal
Functionality	Transfers the RC signal to the microcontroller.

Table 7: Level 2 RC Receiver Board Functional Requirement Table

Module	Team 7B Micro-controller
Inputs	<ul style="list-style-type: none"> • Team 7A Autonomous Signal • RC control signal (manual) • 3.3V (power)
Outputs	<ul style="list-style-type: none"> • GPIO (for status LEDs) • PWM (for motor control) • LED Control
Functionality	Accepts inputs from Team 7A's autonomous signal and the manual control signal from the RC receiver and translates them into motor control signals and LED status indicators..

Table 8: Level 2 Microcontroller Hardware Functional Requirement Table

Module	Weapon and Drive Motor Controller
Inputs	<ul style="list-style-type: none"> • PWM Signal • Forward/Reverse bit (hi/lo) • 24V (power)
Outputs	<ul style="list-style-type: none"> • Variable motor speed and torque.
Functionality	Converts PWM signals into variable motor speed control via voltage averaging and duty cycles.

Table 9: Level 2 Motor Controller Hardware Functional Requirement Table

Module	LED Status Indicators
Inputs	<ul style="list-style-type: none"> • GPIO
Outputs	<ul style="list-style-type: none"> • Visual Indication
Functionality	Accepts GPIO from microcontroller and gives a visual indication of various functions (power on, emergency stop, manual/autonomous mode).

Table 10: Level 2 LED Status Indicators Functional Requirement Table

[AS]

Figure 5 shows the level 2 hardware block diagram for the power system of the autonomous combat robot.

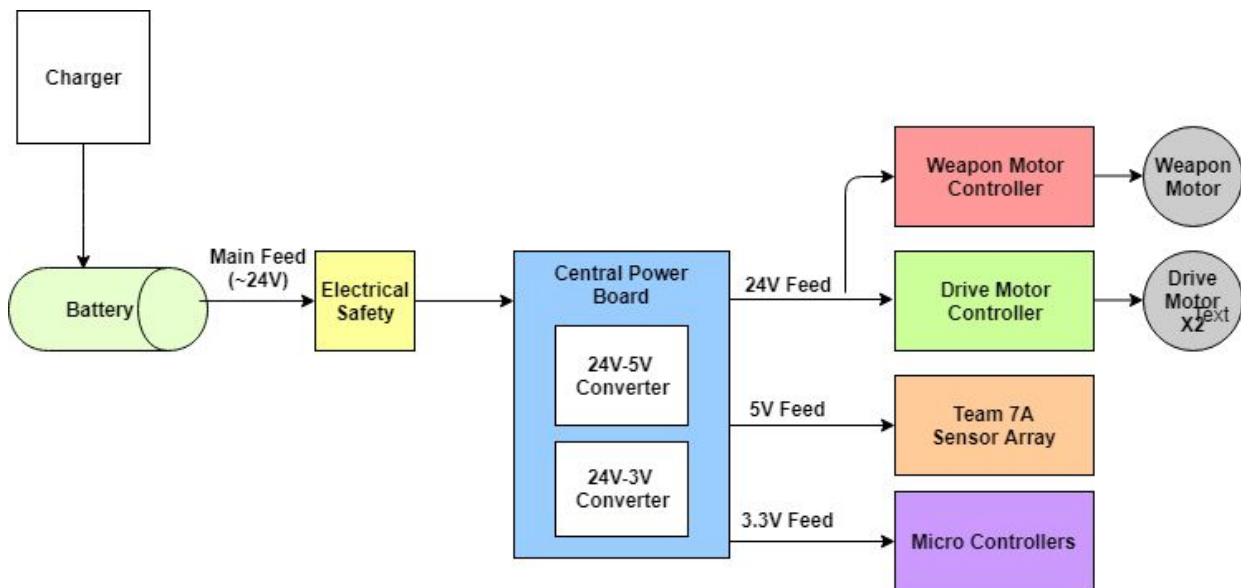


Figure 5: Level 2 Hardware Block Diagram for Power System

The level 2 functional requirement table, indicating the third-level inputs and outputs of the fully autonomous combat robot, are shown in Tables 11, 12, 13, and 14 below.

Module	Battery and Charger System
Inputs	<ul style="list-style-type: none"> ● Wall Outlet
Outputs	<ul style="list-style-type: none"> ● 24V DC Rail Output
Functionality	Charges the Battery Cells. Feeds 24V to the Power Distribution System. Includes Proper Fusing to Avoid Short Circuit Hazard

Table 11: Level 2 Battery and Charger System Functional Requirement Table

Module	Central Power Board
Inputs	<ul style="list-style-type: none"> ● 24V DC Rail
Outputs	<ul style="list-style-type: none"> ● 24V DC ● 5V DC ● 3.3V DC
Functionality	Efficiently converts 24V DC to 5V and 3.5V DC to supply the various sensors and controllers of the robot.

Table 12: Level 2 Central Power Board Functional Requirement Table

Module	Weapon and Drive Motor Controllers
Inputs	<ul style="list-style-type: none"> ● 24V DC ● Control Signal from Microcontroller
Outputs	<ul style="list-style-type: none"> ● Variable power input to motors
Functionality	Supplies power to the motors. Varies motor speeds depending on microcontroller input.

Table 13: Level 2 Weapon and Drive Motor Controllers Functional Requirement Table

Module	Motors
Inputs	<ul style="list-style-type: none"> ● Variable power inputs from motor controllers
Outputs	<ul style="list-style-type: none"> ● Torque ● Speed (RPM)
Functionality	Moves the robot (drive motors) Moves the weapon (weapon motors)

Table 14: Level 2 Motors Functional Requirement Table

[TW]

Figure 6 shows the level 3 hardware block diagram for the control system of the autonomous combat robot. Note that each motor has a DIO and PWM signal. This is a requirement of the motor controllers. The PWM signal sets the speed of the motors and the DIO sets the direction (forward or reverse). Table 15 shows what the four LEDs are for.

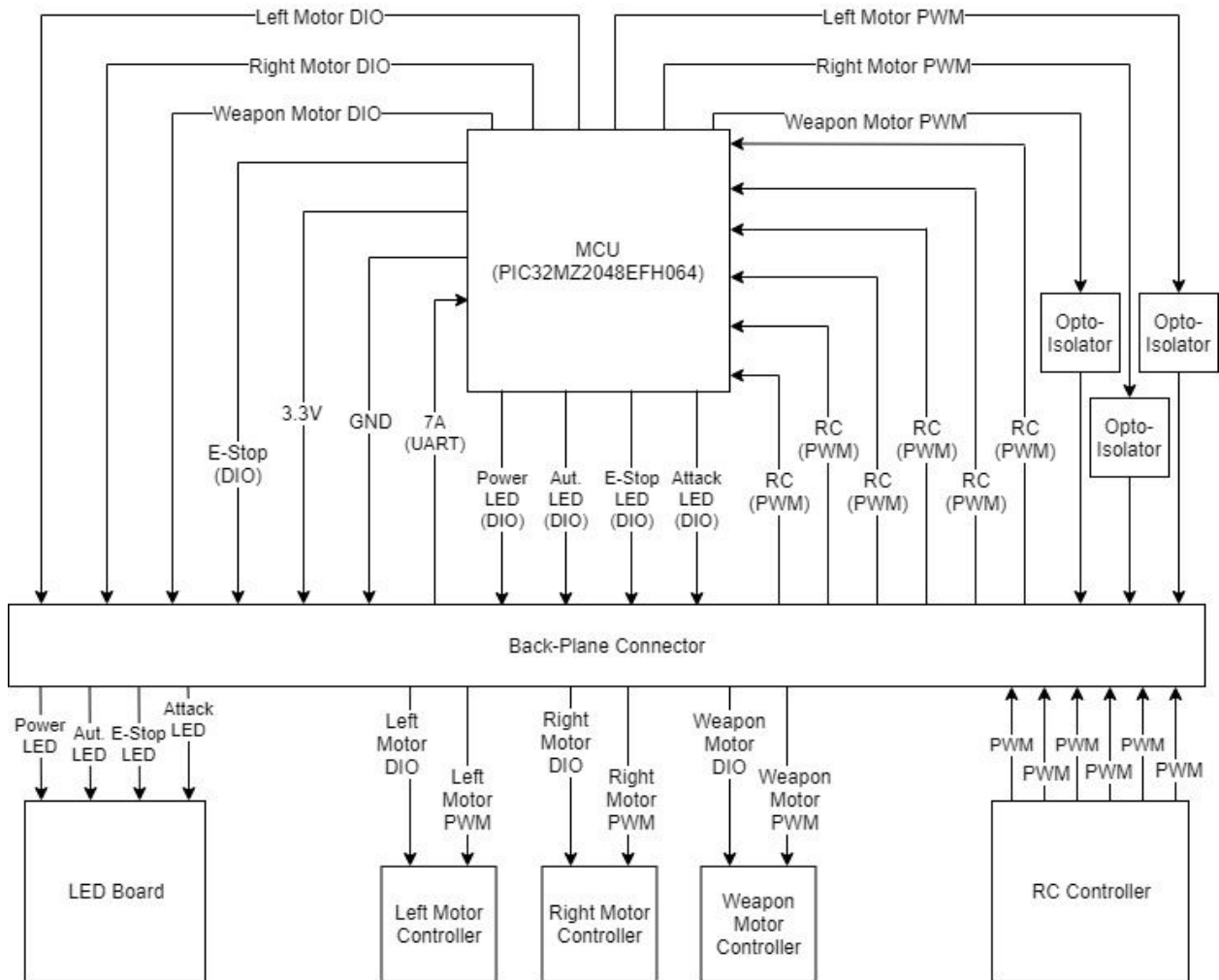


Figure 6: Level 3 Hardware Block Diagram for Control System

LED Signal Name	Purpose
Power LED	Lights if the combat bot is receiving power.
Aut. LED	Lights if the combat bot is in autonomy mode.
E-Stop LED	Lights if the e-stop button is pressed.
Attack LED	Lights if the robot is in autonomy mode and is in motion to attack an opponent.

Table 15: LED Indicators

[TT]

3.2.1.2 Hardware Overview and Schematics

Figures 7 and 8 show the schematic for the control system of the combat robot. The same microprocessor (PIC32MZ2048EFH064) and backplane connector as Team 7A were chosen for system compatibility and ease of integration. Opto-isolators (TCMT1103-OPTO) were placed in between the microprocessor and the motor controllers. The remainder of the circuitry includes necessary signal connections between the rest of the circuitry of the control system and the microprocessor, as well as the suggested microcontroller set-up from the datasheet. Table 16 shows the explanation for each signal name.

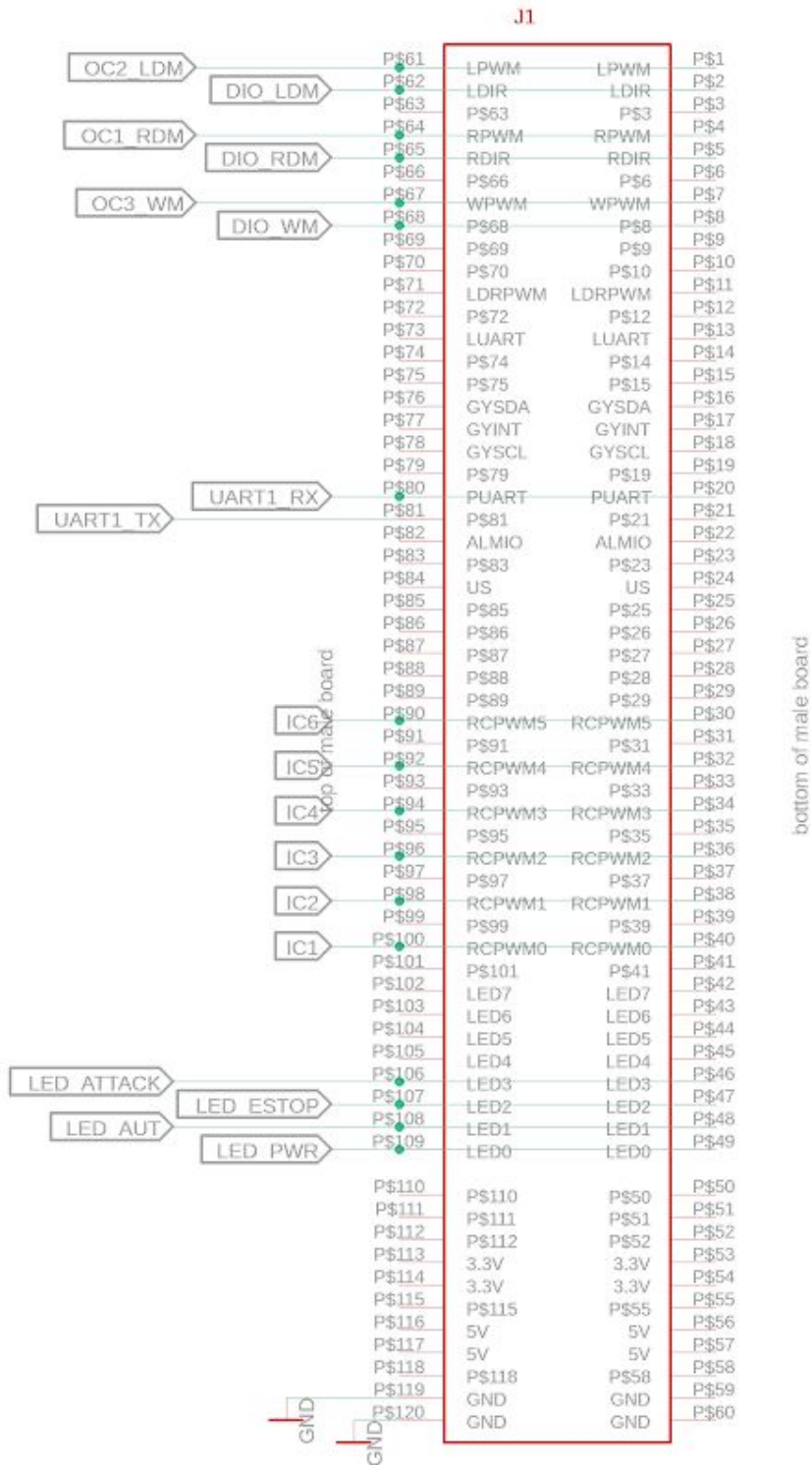


Figure 7: Control System Hardware Schematic - Backplane Connector

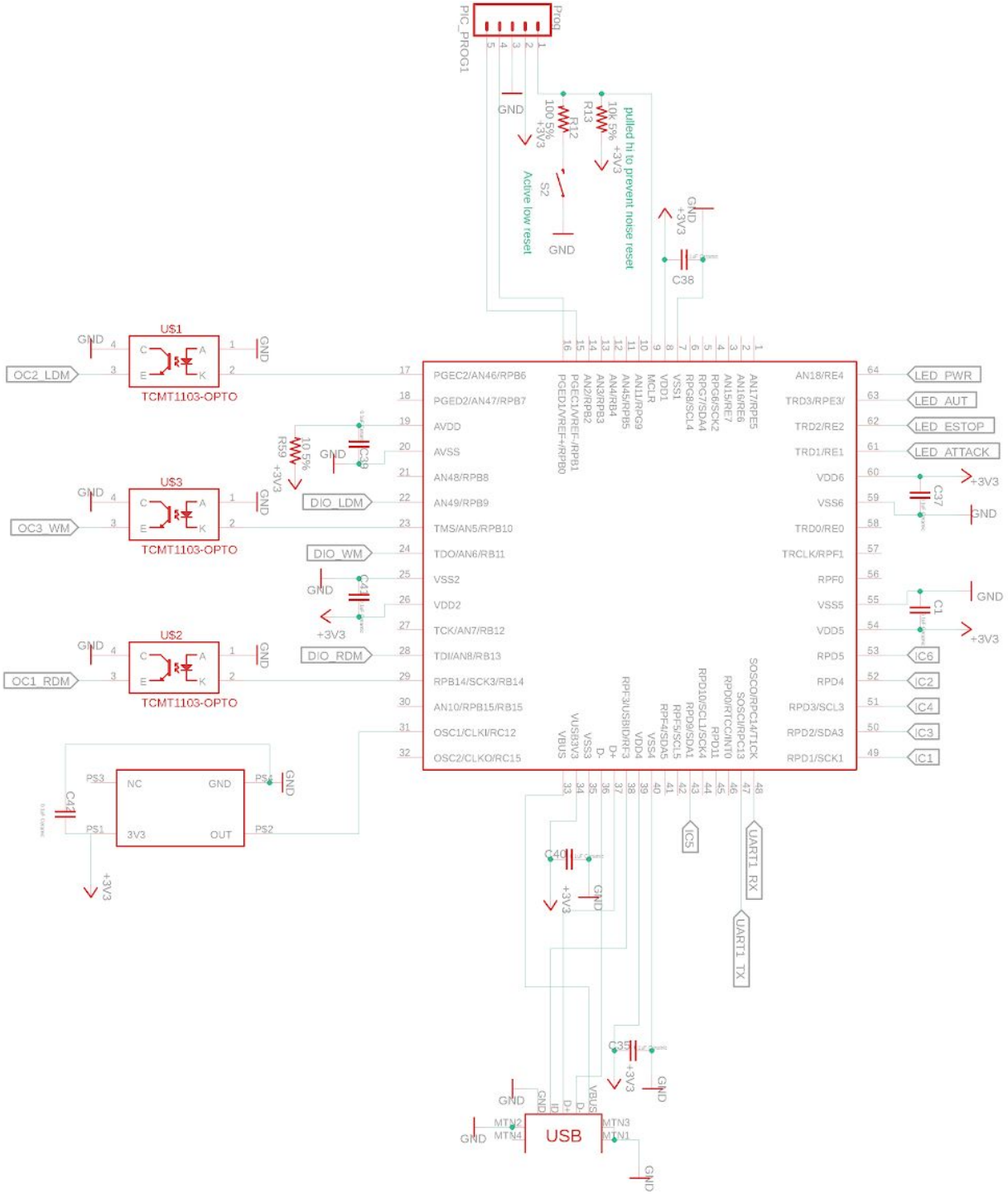


Figure 8: Control System Hardware Schematic - Microprocessor

Signal Name	Purpose
DIO_LDM	DIO output from PIC for left drive motor controller. Indicates direction of motor (forward or reverse).
DIO_RDM	DIO output from PIC for right drive motor controller. Indicates direction of motor (forward or reverse).
DIO_WM	DIO output from PIC for weapon motor controller. Indicates direction of motor (forward or reverse).
OC2_LDM	PWM output from PIC for left drive motor, done through the second output compare module. Indicates speed of motor.
OC1_RDM	PWM output from PIC for right drive motor, done through the first output compare module. Indicates speed of motor.
OC3_WM	PWM output from PIC for weapon motor, done through the third output compare module. Indicates speed of motor.
LED_PWR	DIO output from PIC for the power LED. Lights if the combat bot is receiving power.
LED_AUT	DIO output from PIC for the autonomy LED. Lights if the combat bot is in autonomy mode.
LED_ESTOP	DIO output from PIC for the e-stop LED. Lights if the e-stop button is pressed.
LED_ATTACK	DIO output from PIC for the attack LED. Lights if the robot is in autonomy mode and is in motion to attack an opponent.
IC1	PWM input to PIC from RC controller receiver.
IC2	PWM input to PIC from RC controller receiver.
IC3	PWM input to PIC from RC controller receiver.
IC4	PWM input to PIC from RC controller receiver.
IC5	PWM input to PIC from RC controller receiver.
IC6	PWM input to PIC from RC controller receiver.
UART_RX	UART input (receiving module) to PIC from team 7A. Will give autonomy instructions such as angle, drive mode, orientation, etc.

Table 16: Control System Signals

[TT]

Figures 9 and 10 are the hardware schematic for the power system and backplane. This board is responsible for supplying Team 7B's control system board, Team 7A's control system board, the LEDs, the RC receiver, and the sensor array with power. In addition, this is where all of the routing and connectors will be implemented to maintain a compact, efficient, and noise-free electrical system. This is all one board, but it is broken into two parts for readability.

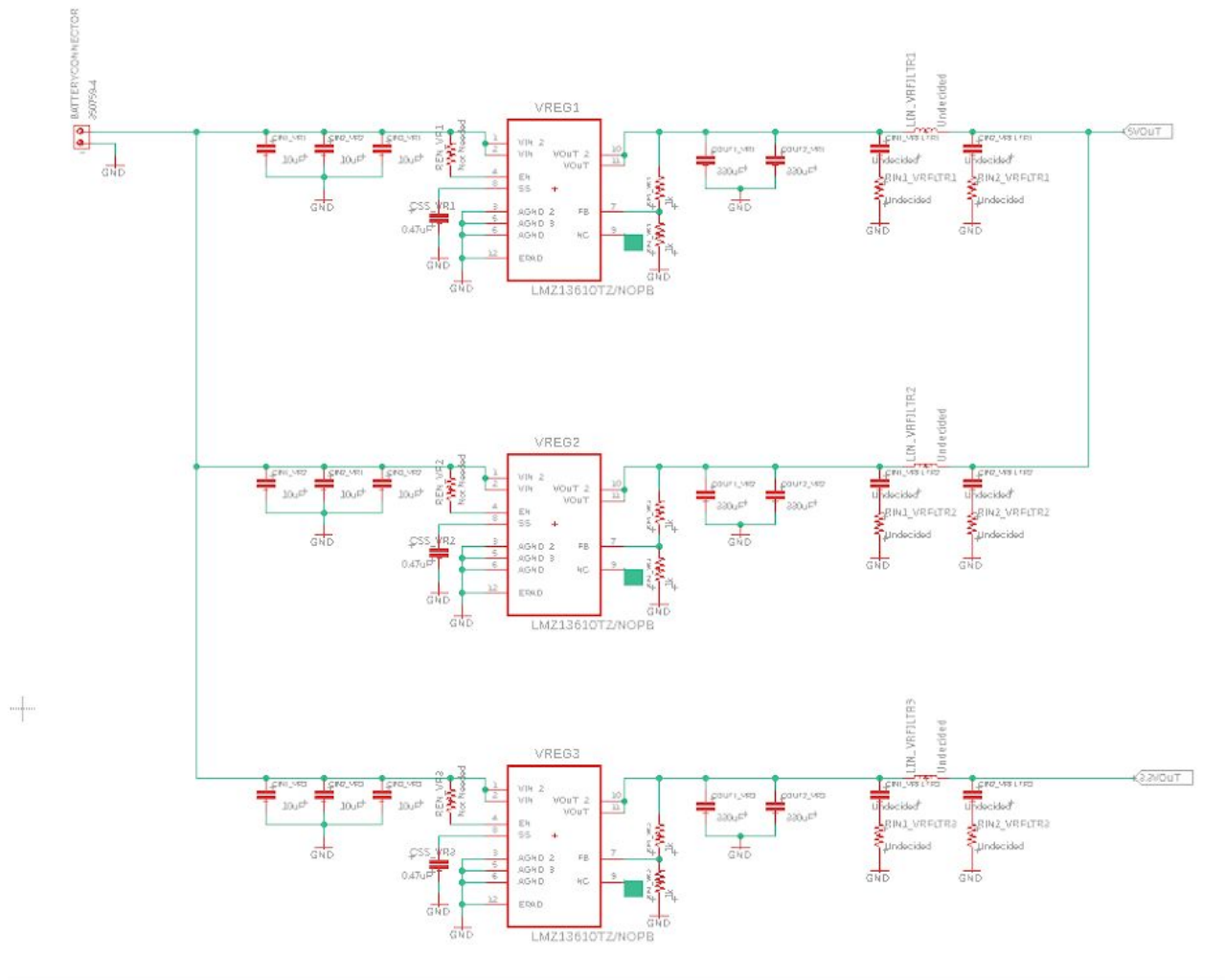


Figure 9: Power System and Backplane Schematic - Voltage Regulators

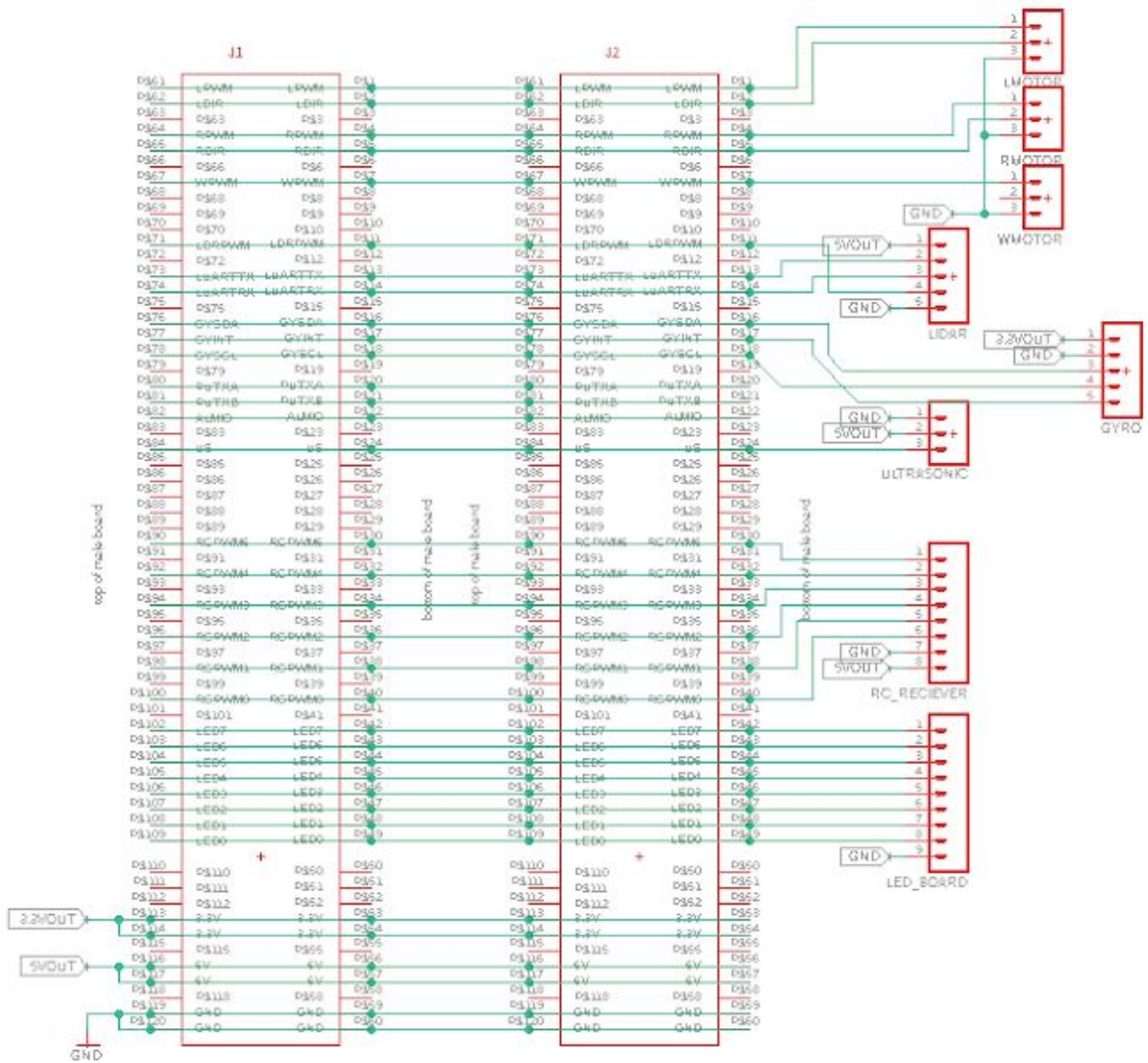


Figure 10: Power System and Backplane Schematic - Connectors

The voltage regulator portion of this board called for the design of DC/DC converters that could meet the following requirements:

- 1) Maintain 3.3V and 5V outputs as the battery rail varies from 20V-30V. This is due to the expected changes in battery voltage as the battery system charges and discharges.
- 2) Pass a minimal amount of noise to the sensitive electronics such as sensors and microcontrollers.
- 3) Regulate voltage efficiently.
- 4) Output enough current to supply all the electronics of Team 7A and Team 7B.

To achieve these requirements, the Texas Instruments LMZ13610TZ switching regulator was selected. While three-terminal devices were suggested for simplicity, their inefficient nature and the requirement of being able to run on battery power for 4 minutes at full load made them a poor choice. A big advantage of the TI regulators is their ability take a variable voltage input and maintain a constant voltage output with very low switching losses.

After opting for switching regulators, it was decided that three should be used - 1 3.3V regulator and 2 5V regulators. This was done due to the high anticipated current draw of the Lidar sensor from Team 7A.

Next, a resistive and capacitive network had to be designed based on the datasheet of the switching regulators in order to achieve the appropriate voltage output, voltage output rise time (power on cycle), and voltage output ripple. A sufficiently chosen feedback resistor network to pin 7 of these devices was crucial in deciding the output voltage, as the device is feedback-dependant. In addition, a biasing resistor between pins 1 and 2 and pin 4 had to be chosen to keep the device switching properly under varying loads.

Finally, in order to meet noise requirements, two 330uF output capacitors were chosen, and placeholders for a pi-filter were added. The output capacitors smooth the output voltage to avoid damaging electronics downstream, but they also act as low-pass filters. The pi filter on the output of the switching regulators is a placeholder that gives the ability to implement LR, RC, or LRC filters easily. While initial analysis has deemed additional filters unnecessary, further experimentation may call for them once the board is constructed. For that reason, the footprints are there.

Moving on to the connector schematic, the purpose of these connectors is to efficiently route power, io, and control, and communication signals where they need to go. While there is not much electrical analysis to be done on the connectors themselves, this segment of the board required a fair amount of thought to lay it out efficiently, and will require further geometric analysis and knowledge of IEEE standards when routing the board in the build and test phase of the project.

Figure 11 below shows the basic layout of the connector with pins labelled with their corresponding signals.

P\$61	LPWM	LPWM	P\$1	LEFT_DRIVE_MOTOR_PWM
P\$62	LDIR	LDIR	P\$2	LEFT_DRIVE_MOTOR_DIRECTION_BIT
P\$63	P\$63	P\$3	P\$3	
P\$64	RPWM	RPWM	P\$4	RIGHT_DRIVE_MOTOR_PWM
P\$65	RDIR	RDIR	P\$5	RIGHT_DRIVE_MOTOR_DIRECTION_BIT
P\$66	P\$66	P\$6	P\$6	
P\$67	WPWM	WPWM	P\$7	WEAPON_DRIVE_MOTOR_PWM
P\$68	P\$68	P\$8	P\$8	
P\$69	P\$69	P\$9	P\$9	
P\$70	P\$70	P\$10	P\$10	
P\$71	LDRPWM	LDRPWM	P\$11	LIDAR_SPEED_CONTROL_PWM
P\$72	P\$72	P\$12	P\$12	
P\$73	LUARTTX	LUARTTX	P\$13	LIDAR_UART_TX
P\$74	LUARTRX	LUARTRX	P\$14	LIDAR_UART_RX
P\$75	P\$75	P\$15	P\$15	
P\$76	GYSDA	GYSDA	P\$16	GYROSCOPE_SENSOR_DA
P\$77	GYINT	GYINT	P\$17	GYROSCOPE_SENSOR_INT
P\$78	GYACL	GYACL	P\$18	GYROSCOPE_SENSOR_CL
P\$79	P\$79	P\$19	P\$19	
P\$80	PUTXA	PUTXA	P\$20	PIC_UART_ATX_BRX
P\$81	PUTXB	PUTXB	P\$21	PIC_UART_BTXX_ARX
P\$82	ALMIO	ALMIO	P\$22	ALARM_BIT
P\$83	P\$83	P\$23	P\$23	
P\$84	US	US	P\$24	ULTRASONIC_SENSOR_DATA_LINE
P\$85	P\$85	P\$25	P\$25	
P\$86	P\$86	P\$26	P\$26	
P\$87	P\$87	P\$27	P\$27	
P\$88	P\$88	P\$28	P\$28	
P\$89	P\$89	P\$29	P\$29	
P\$90	RCPWM5	RCPWM5	P\$30	RC_RECIEVER_PWM5
P\$91	P\$91	P\$31	P\$31	
P\$92	RCPWM4	RCPWM4	P\$32	RC_RECIEVER_PWM4
P\$93	P\$93	P\$33	P\$33	
P\$94	RCPWM3	RCPWM3	P\$34	RC_RECIEVER_PWM3
P\$95	P\$95	P\$35	P\$35	
P\$96	RCPWM2	RCPWM2	P\$36	RC_RECIEVER_PWM2
P\$97	P\$97	P\$37	P\$37	
P\$98	RCPWM1	RCPWM1	P\$38	RC_RECIEVER_PWM1
P\$99	P\$99	P\$39	P\$39	
P\$100	RCPWM0	RCPWM0	P\$40	RC_RECIEVER_PWM0
P\$101	P\$101	P\$41	P\$41	
P\$102	LED7	LED7	P\$42	LED7
P\$103	LED6	LED6	P\$43	LED6
P\$104	LED5	LED5	P\$44	LED5
P\$105	LED4	LED4	P\$45	LED4
P\$106	LED3	LED3	P\$46	LED3
P\$107	LED2	LED2	P\$47	LED2
P\$108	LED1	LED1	P\$48	LED1
P\$109	LED0	LED0	P\$49	LED0
P\$110	P\$110	P\$50	P\$50	
P\$111	P\$111	P\$51	P\$51	
P\$112	P\$112	P\$52	P\$52	
P\$113	3.3V	3.3V	P\$53	3.3V_POWER_RAIL
P\$114	3.3V	3.3V	P\$54	3.3V_POWER_RAIL
P\$115	P\$115	P\$55	P\$55	
P\$116	5V	5V	P\$56	5V_POWER_RAIL
P\$117	5V	5V	P\$57	5V_POWER_RAIL
P\$118	P\$118	P\$58	P\$58	
P\$119	GND	GND	P\$59	GND
P\$120	GND	GND	P\$60	GND

bottom of male board

Figure 11: Backplane Connector Layout

Note that pins across from each other will be connected when soldered onto the board (ex. pin 120 and pin 60).

With the board design out of the way, the motors and motor controllers were to be selected. After considering a few different options for motors, the Ampflow E30-150 motor was selected for the weapon motors. This motor has been selected because it produces 85 oz-in of torque nominally, and over 700 oz-in of torque in a stall (note that the stall condition is not likely to be reached, but it IS capable of producing far more torque than acceleration calculations accounted for). This motor is also capable of a top speed of 5600 rpm when given a 24V signal. This meets our requirements of 5000 rpm for the desired top speed of 10 mph. This motor has also been selected for the weapon for simplicity's sake. Ideally, a higher torque motor could be used for the weapon for more force ($f=m*a$ where acceleration is based on torque), but for budgetary reasons, a higher torque motor may not be possible to use.

After selecting the motors, the motor controller had to be selected. Motor torque is based directly off of current while motor speed is based directly off of voltage. Looking at the motor datasheets, it was determined that the nominal current draw of the motors could be 10A, but the stall current could be as high as 60A. To accommodate this, any motor controller selected had to be able to pass enough current to achieve a good torque rating (roughly ~30A maximum) and able to pass or limit the full draw of the motor. In addition, a motor controller has to be able to withstand the full rail voltage (nominally 24V). For these reasons, the Cytron MD30C was selected. This motor controller can pass up to 30A and will current limit to safe levels (30A) if the motor attempts to draw more current. It is also capable of accepting a 30V input, which will allow it to handle the rail voltage without failure. In addition, the motor controller is easy to

interface with - taking only a PWM signal (speed control) and a 3.3V logic “hi” or “lo” signal (for forward or reverse) from a microchip. The MD30C is also relatively inexpensive (\$31).

[TT]

3.2.2 Software Overview

The software includes choosing between autonomous or manual modes, which was decided for safety reasons. Once chosen, the software must translate autonomous/manual mode signals into motion commands. Autonomous mode signals will be provided by the Sensing and Navigation Team in the form of UART. They will go through a look-up table and then be converted into appropriate signals for motor control. Manual mode signals will be provided by the RC controller in the form of PWM. They will be measured, scaled and ultimately converted into pwm signals for motor control. Furthermore, the software is responsible for accounting for emergency stop, visual status indicators, and robot orientation. The below flowcharts represent an overview of the software design of the combat bot.

[AS, TT, TW]

3.2.2.1 Software Flowcharts

The below sections show the software flowcharts for the system.

[TT]

3.2.2.1.1 Level 0 Software Flowchart

The level 0 software flowchart indicates the top-level inputs and outputs of the fully autonomous combat robot. Figure 12 shows the level 0 flowchart for DT07B's part of the autonomous combat robot.

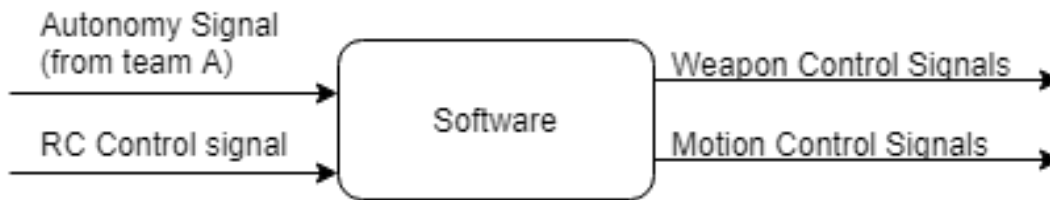


Figure 12: Level 0 Software Flowchart

[AS]

3.2.2.1.2 Level 1 Software Flowchart

The level 1 block diagram is an expansion of the level 0 diagram. Figures 13, 14, and 15 show the level 1 flowcharts for DT07B's part of the autonomous combat robot.

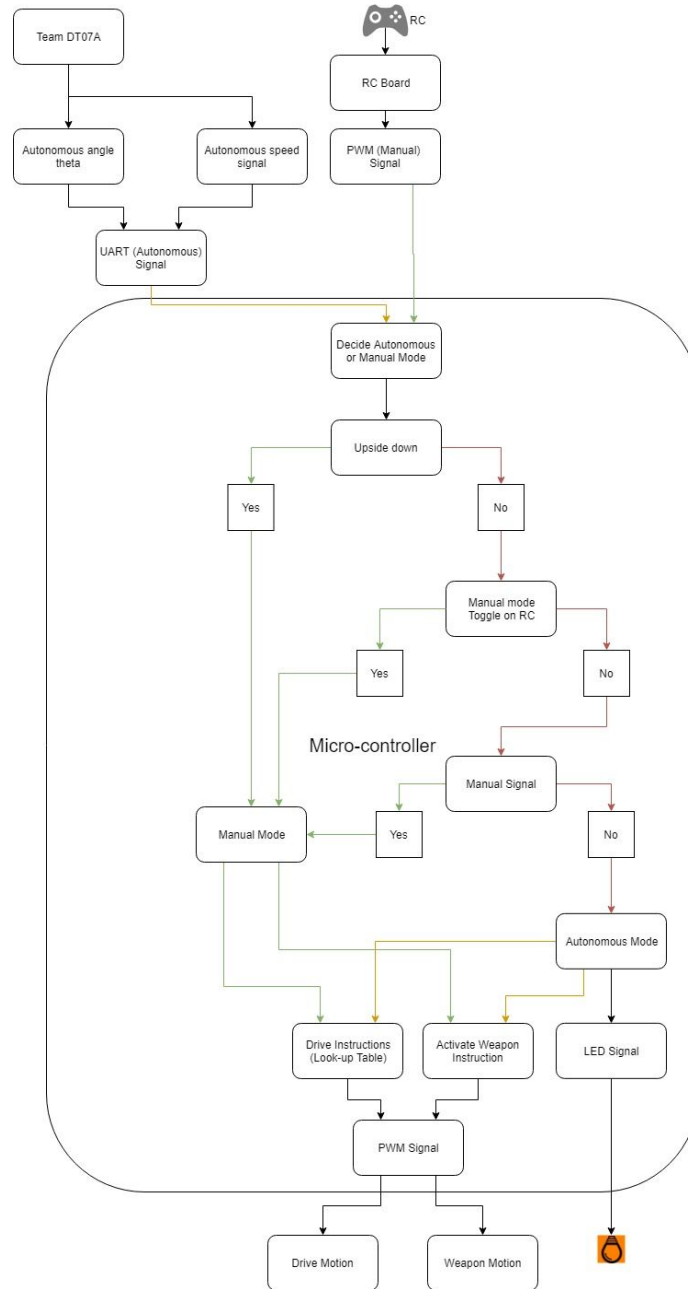


Figure 13: Level 1 Software Flowchart

[AS, TT]

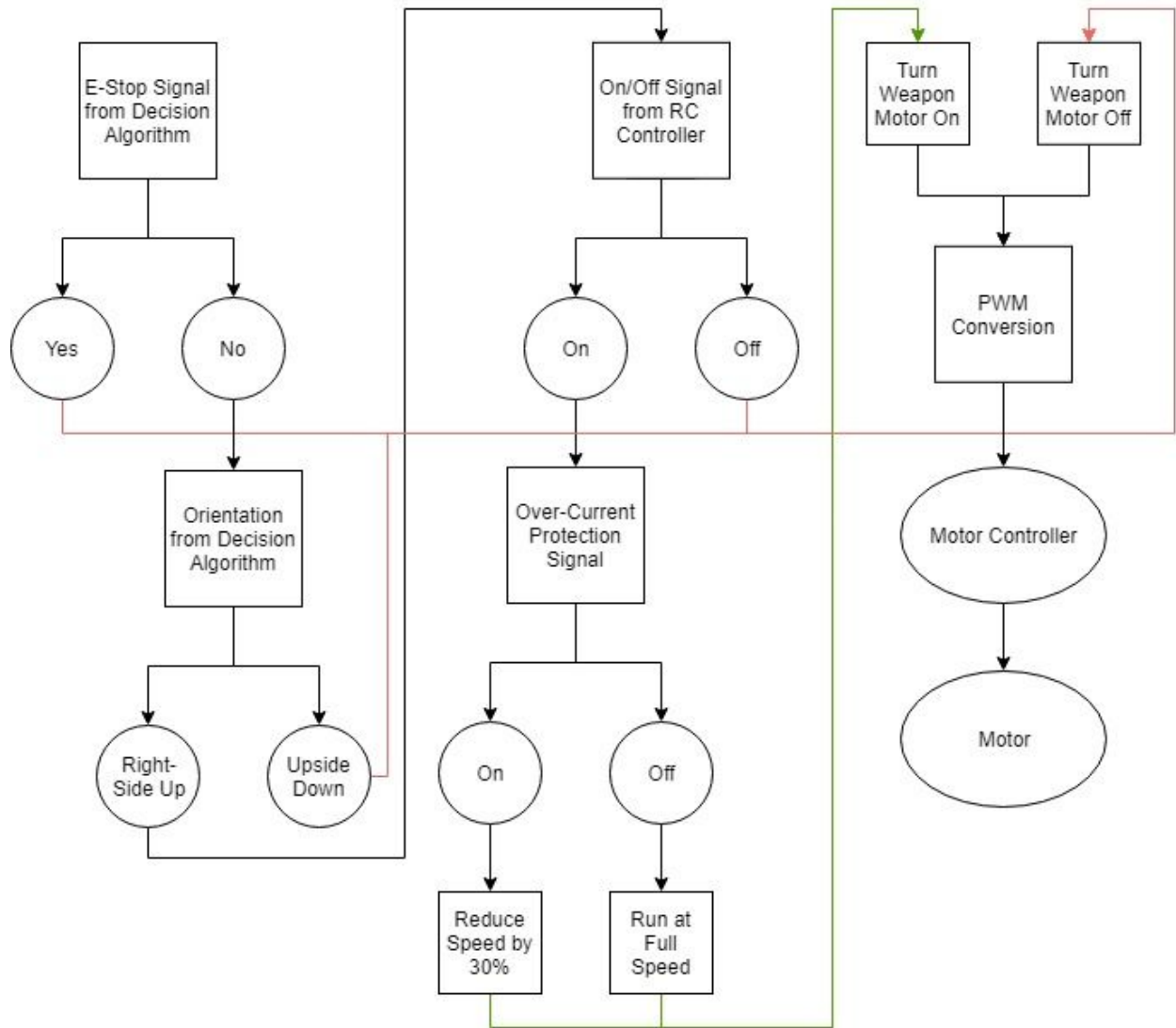


Figure 14: Level 1 Weapon Control Flowchart

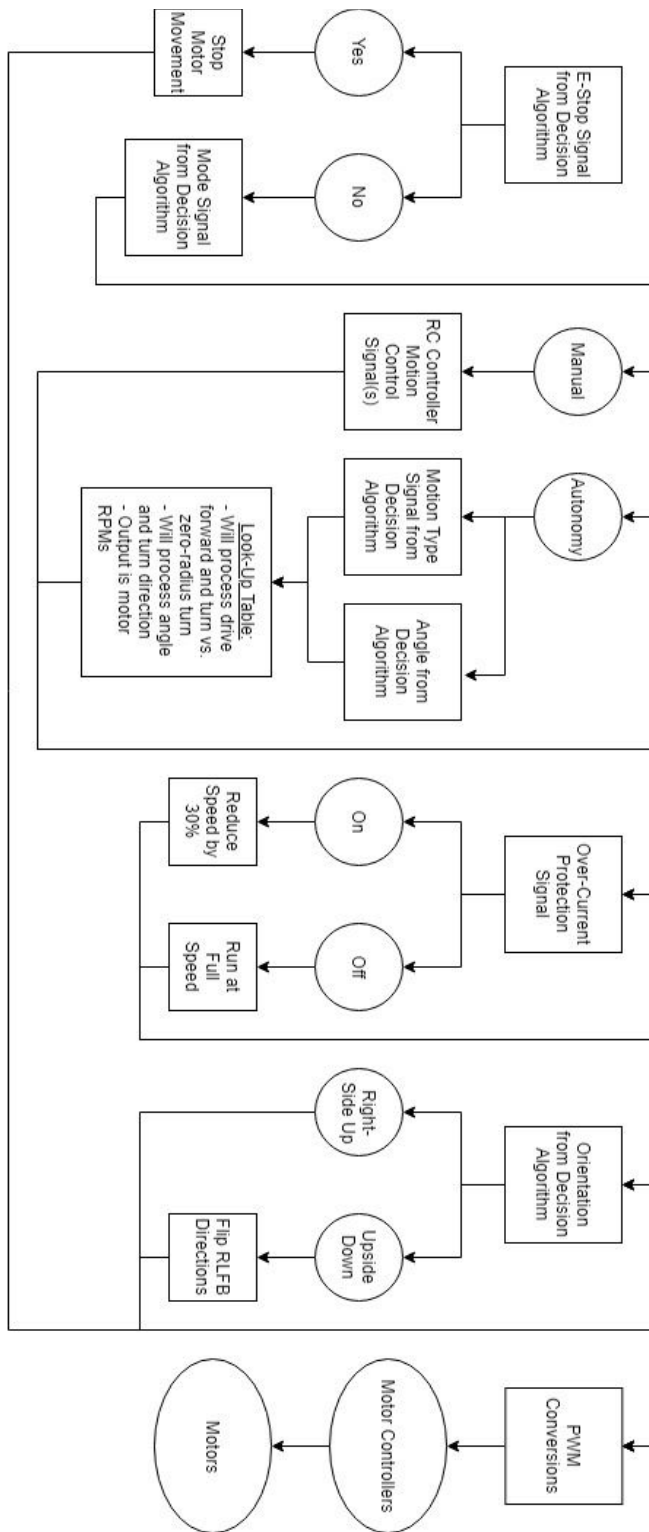


Figure 15: Level 1 Motion Control Flowchart

[TT]

3.2.2.2 Pseudocode

Below shows the pseudocode for the decision and actualization algorithms.

[TT]

```
//receive/accept 3 UART signals from DT07A

//signal 1: angle theta

//signal 2: speed

//signal 3: are we upside down or not?

//receive/accept 1 PWM signal from the RC Controller receiver

//Decide Autonomous or Manual mode

//    Manual mode function

//    Use Manual mode if:

//        1: Robot is upside down

//        OR

//        2: RC controller toggle switch is flipped to manual mode

//        OR

//        3: RC controller toggle switch is flipped to autonomous mode BUT a manual

//            signal is received (Manual Override)

//            if both Autonomous and Manual signals are received, then listen to the

//                manual signal

//Insert Manual mode function here

//    Autonomous mode function

//    Use Autonomous mode if:
```

```

//          1: Robot is not upside down
//          AND
//          2: RC controller toggle switch is flipped to autonomous mode
//          AND
//          3: No manual signal is received

//Insert Autonomous mode function here

//Insert Weapon Activation function here

//Insert Drive Instructions function here

//-----

//Main Function

//Initialize variables

//    if in autonomous mode:

//          1: Turn on Autonomous mode LED indicator
//          2: Call Weapon Activation function
//          3: Call Drive Instructions function

//    elseif in manual mode:

//          1: Call Weapon Activation function
//          2: Call Drive Instructions function

//    else

//          Activate Watchdog reset

//

```

```
//Turn on LEDs

//    if in autonomous mode:

//        Turn on autonomous mode LED

//    if robot is on:

//        Turn on robot power LED (Manual mode enabled)

//    if robot detects enemy:

//        Turn on enemy detected LED

//    if robot loses communication:

//        Turn on E-stop LED

//

//Create a PWM signal to send to the Drive and Weapon motors

//Insert main function here

//Interrupts

//    Watchdog reset

//    Emergency stop (from RC controller) if:

//        Connection between robot and controller is lost
```

[AS]

```
#include <iostream>

using namespace std;

// Initialize functions

bool decideMode(bool, bool);
```

```
void command_drive_motors(bool, bool, bool, bool, double, int);
```

```
void command_weapon_motor(bool, bool, bool, bool);
```

```
int main(){
```

```
    // Initialize variables
```

```
    bool aut_mode = 0;           // Indicates if autonomy mode is in use
```

```
    bool man_toggle = 1;        // Indicates if manual mode toggle button was pressed
```

```
    bool man_signal = 1;        // Indicates if receiving a manual mode signal
```

```
    bool e_stop = 0;            // Indicates if e-stop activated
```

```
    bool upside_down = 0;       // Indicates if bot is upside down
```

```
    bool weapon_off_toggle = 0; // Indicates if weapon is turned off
```

```
    bool oc_weapon = 0;         // Indicates if weapon motor is in overcurrent mode
```

```
    bool oc_left_motor = 0;     // Indicates if left drive motor is in overcurrent mode
```

```
    bool oc_right_motor = 0;    // Indicates if right drive motor is in overcurrent mode
```

```
    // Global variables
```

```
    double angle;
```

```
    int motion_type;
```

```
    // Set power LED
```

```
    while (1) {
```

```

// Read angle, speed, orientation, and other errors from autonomy (DT07A)

// Set variables

    // upside_down;

// Read data from RC controller

// Set variables

    // e_stop

    // LED for status of RC controller

// Decide between manual and autonomy mode

aut_mode = decideMode(man_toggle, man_signal);

// Data conversion so that autonomy and PWM can be fed into the same calculations/table,
and edit BD

// Command drive motors

command_drive_motors(e_stop, upside_down, oc_left_motor, oc_right_motor, angle,
motion_type);

// Turn weapon motor on or off

command_weapon_motor(e_stop, upside_down, weapon_off_toggle, oc_weapon);
}

return 0;
}

```

```

// Decide between manual and autonomy mode

bool decideMode(bool man_toggle_1, bool man_signal_1) {

    // Initialize variables

    bool aut_mode_1 = 0;

    bool aut_LED = 0;

    // If RC manual mode toggle button is pressed, set manual mode

    if (man_toggle_1 == 1) {

        aut_mode_1 = 0;

    }

    // Otherwise, if receiving a manual signal, set manual mode

    else if (man_signal_1 == 1) {

        aut_mode_1 = 0;

    }

    // Otherwise, set autonomous mode

    else {

        aut_mode_1 = 1;

    }

    // Convert autonomy/manual mode into angle and motion type that can be processed the same
way for both cases

```



```

// Set angle

// Set motion_type

// Set mode LED pin to indicate if in autonomy mode
aut_LED = aut_mode_1;

// Turn autonomy LED on/off

// Turn attacking LED on/off

return aut_mode_1;
}

// Command drive motors
void command_drive_motors(bool e_stop_1, bool upside_down_1, bool oc_left_motor_1, bool
oc_right_motor_1, double angle_1, int motion_type_1) {
// Initialize variables

double left_motor_speed = 0; // Duty cycle

int left_motor_dir = 1; // Forward

double right_motor_speed = 0; // Duty cycle

int right_motor_dir = 1; // Forward

double temp_1 = 0; // For flipping robot directions

double temp_2 = 0; // For flipping robot directions

```

```

// If e-stop button is pressed, turn motors off

if (e_stop_1 == 1) {
    left_motor_speed = 0;
    right_motor_speed = 0;
}

// Otherwise...

else {

    // Put angle_1 and motion_type_1 through lookup table to set motor speed and directions

    // Set left_motor_speed

    // Set right_motor_speed

    // If over current protection needed, run motor at 70% of full speed

    if (oc_left_motor == 1) {
        left_motor_speed = 0.7 * left_motor_speed;
    }

    else if (oc_right_motor == 1) {
        right_motor_speed = 0.7 * right_motor_speed;
    }

    // If upside down, flip motor direction

    if (upside_down_1 == 1) {

```

```

// Flip left and right directions
temp_1 = left_motor_speed;
left_motor_speed = right_motor_speed;
right_motor_speed = temp_1;

// Flip forward and reverse directions
temp_2 = left_motor_dir;
left_motor_dir = right_motor_dir;
right_motor_dir = temp_2;
}
}

// Send direction and PWM to motor controller for left and right motors
return;
}

// Turn weapon motor on or off
void command_weapon_motor(bool e_stop_1, bool upside_down_1, bool weapon_off_toggle_1,
bool oc_weapon_1) {
// Initialize variables
bool direction = 1; // Forward
int motor_speed = 0; // Duty cycle

```

```
// If e-stop button is pressed, turn motor off
if (e_stop_1 == 1) {
    motor_speed = 0;
}

// Otherwise, if upside down, turn motor off
else if (upside_down_1 == 1) {
    motor_speed = 0;
}

// Otherwise, if RC weapon off toggle button is pressed, turn motor off
else if (weapon_off_toggle_1 == 1) {
    motor_speed = 0;
}

// Otherwise, if over current protection needed, run motor at 70% of full speed
else if (oc_weapon_1 == 1) {
    motor_speed = 0.7;
}

// Otherwise, run motor at full speed
else {
    motor_speed = 1;
}

// Send direction and PWM to motor controller
```

```
return;  
}
```

[TT]

3.2.3 Mechanical Overview

Through research trade studies (see Appendix) and video research of various combat robotics competitions, the team has decided to design a hybrid between the wedge and drum weapon systems. The decision to use a wedge in addition to a drum was made primarily because wedge bots have an advantage over many because they can get under the opponent robot and avoid the opponent's weapon or disabling them by flipping them. The drawback of a wedge alone is if the enemy robot is designed to drive on both top and bottom the wedge is not likely to disable the robot. Wedges alone generally do not inflict critical damage.

The proposed design will add a drum weapon mechanism to the wedge design. This will allow the robot to inflict critical damage to the underside of the opponent's robot. The team's hybrid design will have the robustness and defensive capabilities of a wedge, but have the damaging capabilities of a drum weapon system. This is because, as mentioned previously, the wedge is good primarily for avoiding the opponent's weapons and flipping the opponent but not inflicting the damage needed to destroy or disable the opponent.

Using a wedge-drum hybrid will allow for all of the benefits of the wedge, but will also provide a mechanism which will be able to actually disable or destroy the opposing robot. This hybrid design will support the implementation of the intercept or escape algorithms.

[CH, TT, TW]

Figure 16 and 17 below shows a rendition of the mechanical design. The team plans on moving much of the body of the LiDAR sensor into the chassis to protect them. The drum spinner has been made a small width to enable fast speed up times and increase impact delivery force by minimizing the distribution of the impact.

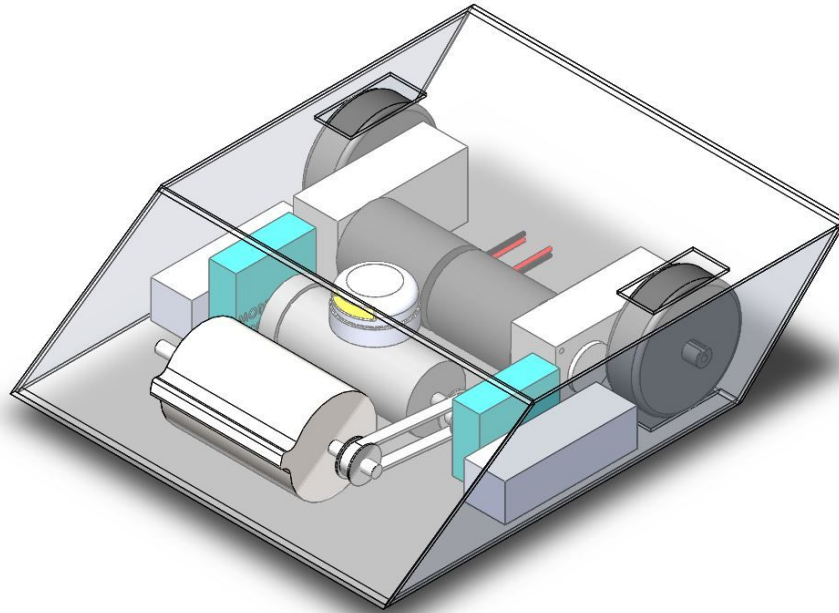


Figure 16: Mechanical Design - Isometric View

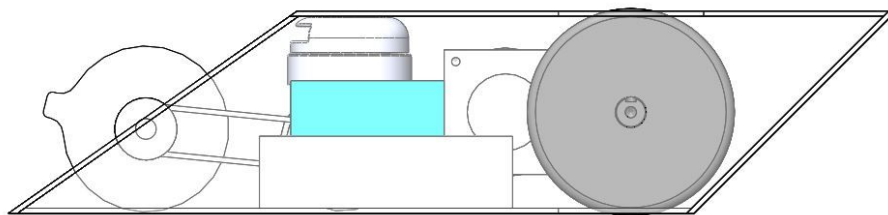


Figure 17: Mechanical Design - Planar View

[FA, CH, AS, TT, TW]

4. Parts List

Qty	Value	Device	Parts	Description
8	0.1uF	CAP0805	C1, C35, C37, C38, C39, C40, C41, C42	Capacitor
1		PCIMALE	J1	Card Edge Connector
1		Pic_Prog 1	Header 5	Pic Programming Header
1	100	R-US_R0 805	R12	Resistor
1	10k	R-US_R0 805	R13	Resistor
1	10	R-US_R0 805	R59	Resistor
3		TCMT11 03-OPTO	U\$1, U\$2, U\$3	Optoisolator- NPN output
1		PIC32MZ 2048EFH 064	U\$5	Microprocessor, TQFP64
1		SWCH-11 966	S2	Switch
1		KC2520B	U\$9	KC2520B
1		USB-MIN IM-5PIN	U\$11	Mini-USB "B" connector with 5th pin broken

Table 17: Control Board BOM

[TT]

Qty	Value	Device	Parts	Description
4		M03	LMOTOR, RMOTOR, ULTRASONIC, WMOTOR	AMP QUICK CONNECTOR
2		M05	GYRO, LIDAR	AMP QUICK CONNECTOR
1		M08	RC_RECIEVER	AMP QUICK CONNECTOR
1		M09	LED_BOARD	AMP QUICK CONNECTOR
2		PCIFEMALE	J1, J2	Card Edge Connector
3	0.47uF	CAP0805	CSS_VR1, CSS_VR2, CSS_VR3	Capacitor
9	10uF	CAP0805	CIN1_VR1, CIN1_VR2, CIN1_VR3, CIN2_VR1, CIN2_VR2, CIN2_VR3, CIN3_VR1, CIN3_VR2, CIN3_VR3	Capacitor
6	1k	R-US_R0805	RF1_VR1, RF1_VR2, RF1_VR3, RF2_VR1, RF2_VR2, RF2_VR3	RESISTOR, American symbol
6	330uF	CAP0805	COU1_VR1, COU1_VR2, COU1_VR3, COU2_VR1, COU2_VR2, COU2_VR3	Capacitor
1	350759-4	350759-4	BATTERYCONNECTOR	Universal MATE-N-LOK .250 Centerline, 600 V, 19 - 36 A max"
3	LMZ13610TZ/NOPB	LMZ13610TZ/NOPB	VREG1, VREG2, VREG3	BUCK, SYNC, ADJ, 10A, TO-PMOD-11
3	1k	R-US_R0805	REN_VR1, REN_VR2, REN_VR3	RESISTOR, American symbol
6	0.047uF	CAP0805	CIN1_VRFLTR1, CIN1_VRFLTR2, CIN1_VRFLTR3, CIN2_VRFLTR1, CIN2_VRFLTR2, CIN2_VRFLTR3	Capacitor
3	0.001uH	INDUCTOR-0805-3.3UH	LIN_VRFILTR1, LIN_VRFILTR2, LIN_VRFILTR3	Inductors
6	1K	R-US_R0805	RIN1_VRFLTR1, RIN1_VRFLTR2, RIN1_VRFLTR3, RIN2_VRFLTR1, RIN2_VRFLTR2, RIN2_VRFLTR3	RESISTOR, American symbol

Table 18: Power Board/Backplane BOM [TW]

Qty	Suggested Vendor	Vendor Part Number	Description
3	AmpFlow	E30-150	Weapon and Drive Motors
2	Cytron	MDS40B	Motor Controllers - Drive
1	Cytron	MD30C	Motor Controllers - Weapon
2	Revolectrix	B435 YS5000-6S-XP	Batteries
2	Turnigy	9466000015-0	Charger

Table 19: General Electrical System BOM

[AS, TT]

5. Project Schedules

ID	Task Name	Duration	Start	Finish	Resource Names
1	SDP1 Fall 2018				
2	Project Design				
3	Preliminary report	11 days	Thu 9/6/18	Sun 9/16/18	
4	Cover page	11 days	Thu 9/6/18	Sun 9/16/18	Tanya Tebcherani
5	T of C, L of T, L of F	11 days	Thu 9/6/18	Sun 9/16/18	Tanya Tebcherani
6	Need	11 days	Thu 9/6/18	Sun 9/16/18	Andrew Szabo
7	Objective	11 days	Thu 9/6/18	Sun 9/16/18	Tristin Weber
8	Background	11 days	Thu 9/6/18	Sun 9/16/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
9	Marketing Requirements	11 days	Thu 9/6/18	Sun 9/16/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
10	Objective Tree	11 days	Thu 9/6/18	Sun 9/16/18	Andrew Szabo
11	Block Diagrams Level 0, 1, ... w/ FR tables	11 days	Thu 9/6/18	Sun 9/16/18	
12	Hardware modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18	Tristin Weber,Tanya Tebcherani
13	Software modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18	Andrew Szabo,Tanya Tebcherani
14	Mechanical Sketch	11 days	Thu 9/6/18	Sun 9/16/18	Mechanical Team
15	Team Information	11 days	Thu 9/6/18	Sun 9/16/18	Tanya Tebcherani,Andrew Szabo,Tristin Weber
16	References	11 days	Thu 9/6/18	Sun 9/16/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
17	Preliminary Parts Request Form	11 days	Thu 9/6/18	Sun 9/16/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
18	Midterm Report	35 days	Thu 9/6/18	Wed 10/10/18	
19	Design Requirements Specification	14 days	Mon 9/17/18	Sun 9/30/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
20	Midterm Design Gantt Chart	14 days	Mon 9/17/18	Sun 9/30/18	Andrew Szabo
21	Design Calculations	24 days	Mon 9/17/18	Wed 10/10/18	
22	Electrical Calculations	24 days	Mon 9/17/18	Wed 10/10/18	
23	Communication	24 days	Mon 9/17/18	Wed 10/10/18	
24	Computing	24 days	Mon 9/17/18	Wed 10/10/18	
25	Control Systems	24 days	Mon 9/17/18	Wed 10/10/18	
26	Power, Voltage, Current	24 days	Mon 9/17/18	Wed 10/10/18	Tristin Weber
27	Electromagnetic Radiation	24 days	Mon 9/17/18	Wed 10/10/18	
28	Thermal	24 days	Mon 9/17/18	Wed 10/10/18	
29	Mechanical Calculations	24 days	Mon 9/17/18	Wed 10/10/18	Mechanical Team
30	Structural Considerations	24 days	Mon 9/17/18	Wed 10/10/18	
31	System Dynamics	24 days	Mon 9/17/18	Wed 10/10/18	
32	Block Diagrams Level 0 w/ FR tables & To	7 days	Mon 9/17/18	Sun 9/23/18	
33	Hardware modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18	Tristin Weber,Andrew Szabo,Tanya Tebcherani
34	Software modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18	Andrew Szabo
35	Block Diagrams Level 1 w/ FR tables & To	7 days	Mon 9/24/18	Sun 9/30/18	
36	Hardware modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18	Andrew Szabo
37	Software modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18	Andrew Szabo,Tanya Tebcherani
38	Block Diagrams Level 2 w/ FR tables & To	10 days	Mon 10/1/18	Wed 10/10/18	
39	Hardware modules (identify designer)	10 days	Mon 10/1/18	Wed 10/10/18	Tristin Weber
40	Software modules (identify designer)	10 days	Mon 10/1/18	Wed 10/10/18	Andrew Szabo,Tanya Tebcherani
41	Midterm Design Presentations Part 1	1 day	Thu 10/11/18	Thu 10/11/18	
42	Midterm Design Presentations Part 2	1 day	Thu 10/18/18	Thu 10/18/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
43	Project Poster	14 days	Mon 10/8/18	Sun 10/21/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
44	Secondary Parts Request Form	21 days	Mon 9/17/18	Sun 10/7/18	
45	Final Design Report	52 days	Mon 10/8/18	Wed 11/28/18	
46	Abstract	52 days	Mon 10/8/18	Wed 11/28/18	Andrew Szabo
47	Software Design	31 days	Mon 10/8/18	Wed 11/7/18	
48	Modules 1...n	31 days	Mon 10/8/18	Wed 11/7/18	
49	Pseudo Code	31 days	Mon 10/8/18	Wed 11/7/18	Andrew Szabo,Tanya Tebcherani
50	Hardware Design	31 days	Mon 10/8/18	Wed 11/7/18	
51	Modules 1...n	31 days	Mon 10/8/18	Wed 11/7/18	
52	Simulations	31 days	Mon 10/8/18	Wed 11/7/18	Tristin Weber,Tanya Tebcherani
53	Schematics	31 days	Mon 10/8/18	Wed 11/7/18	Tristin Weber,Tanya Tebcherani
54	Parts Lists	52 days	Mon 10/8/18	Wed 11/28/18	
55	Parts list(s) for Schematics	52 days	Mon 10/8/18	Wed 11/28/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
56	Materials Budget list	52 days	Mon 10/8/18	Wed 11/28/18	Andrew Szabo,Tanya Tebcherani,Tristin Weber
57	Proposed Implementation Gantt Chart	52 days	Mon 10/8/18	Wed 11/28/18	Andrew Szabo
58	Conclusions and Recommendations	52 days	Mon 10/8/18	Wed 11/28/18	Andrew Szabo,Tanya Tebcherani
59	Final Design Presentations Part 1	1 day	Thu 11/8/18	Thu 11/8/18	
60	Final Design Presentations Part 2	1 day	Thu 11/15/18	Thu 11/15/18	
61	Secondary Parts Request Form	14 days	Thu 10/4/18	Wed 10/17/18	
62	Final Parts Request Form	56 days	Mon 10/8/18	Sun 12/2/18	Andrew Szabo

Figure 18: Gantt Chart Fall 2018

ID	Task Name	Duration	Start	Finish	Resource Names
1	SDP11 Implementation 2018	103 days	Mon 1/14/19	Fri 4/26/19	
2	Revise Gantt Chart	14 days	Mon 1/14/19	Sun 1/27/19	Andrew Szabo
3	Implement Project Design	96 days	Mon 1/14/19	Fri 4/19/19	
4	Hardware Implementation	56 days	Mon 1/14/19	Sun 3/10/19	
5	Breadboard Components	13 days	Mon 1/14/19	Sat 1/26/19	Tanya Tebcherani,Tristin Weber
6	Layout and Generate PCB(s)	14 days	Sun 1/27/19	Sat 2/9/19	Tanya Tebcherani,Tristin Weber
7	Assemble Hardware	7 days	Sun 2/10/19	Sat 2/16/19	Tanya Tebcherani,Tristin Weber
8	Test Hardware	14 days	Sun 2/17/19	Sat 3/2/19	Tanya Tebcherani,Tristin Weber
9	Revise Hardware	14 days	Sun 2/17/19	Sat 3/2/19	Tanya Tebcherani,Tristin Weber
10	<i>MIDTERM: Demonstrate Hardware</i>	5 days	Sun 3/3/19	Thu 3/7/19	Tanya Tebcherani,Tristin Weber
11	SDC & FA Hardware Approval	0 days	Fri 3/8/19	Fri 3/8/19	Tanya Tebcherani,Tristin Weber
12	Software Implementation	56 days	Mon 1/14/19	Sun 3/10/19	
13	Develop Software	27 days	Mon 1/14/19	Sat 2/9/19	Andrew Szabo,Tanya Tebcherani
14	Test Software	21 days	Sun 2/10/19	Sat 3/2/19	Andrew Szabo,Tanya Tebcherani
15	Revise Software	21 days	Sun 2/10/19	Sat 3/2/19	Andrew Szabo,Tanya Tebcherani
16	<i>MIDTERM: Demonstrate Softw</i>	5 days	Sun 3/3/19	Thu 3/7/19	Andrew Szabo,Tanya Tebcherani
17	SDC & FA Software Approval	0 days	Fri 3/8/19	Fri 3/8/19	Andrew Szabo,Tanya Tebcherani
18	System Integration	42 days	Sat 3/9/19	Fri 4/19/19	
19	Assemble Complete System	14 days	Sat 3/9/19	Fri 3/22/19	Andrew Szabo,Tanya Tebcherani,Tristin Weber
20	Test Complete System	21 days	Sat 3/23/19	Fri 4/12/19	Andrew Szabo,Tanya Tebcherani,Tristin Weber
21	Revise Complete System	21 days	Sat 3/23/19	Fri 4/12/19	Andrew Szabo,Tanya Tebcherani,Tristin Weber
22	<i>Demonstration of Complete System</i>	7 days	Sat 4/13/19	Fri 4/19/19	Andrew Szabo,Tanya Tebcherani,Tristin Weber
23	Develop Final Report	99 days	Mon 1/14/19	Mon 4/22/19	
24	Write Final Report	99 days	Mon 1/14/19	Mon 4/22/19	Andrew Szabo,Tanya Tebcherani,Tristin Weber
25	Submit Final Report	0 days	Mon 4/22/19	Mon 4/22/19	Tanya Tebcherani
26	Spring Recess	7 days	Mon 3/25/19	Sun 3/31/19	
27	Project Demonstration and Presentation	0 days	Fri 4/26/19	Fri 4/26/19	Andrew Szabo,Tanya Tebcherani,Tristin Weber

Figure 19: Gantt Chart Spring 2019

[AS]

	AJ Szabo	Tanya Tebcherani	Tristin Weber
9/23	<ul style="list-style-type: none"> - Researched lidars and found a few possible lidars we could use * Scanse lidar * rplidar - Discussed motor control with Tristin - Edited preliminary report, defining team B - Brainstorming on software block diagrams - Looking into how to make (or buy if possible) a watchdog board - Worked with SOuRCe to create an on campus student organization * We are officially a student organization as of Tuesday, 9/25/18, and we have an orgsync page * In the process of requesting funding for travel, parts, etc. 	<ul style="list-style-type: none"> - Previously worked on sensor selection (lidar and proximity) - Edited preliminary report's marketing requirements and block diagrams with the team - Brainstorming on further hardware block diagram breakdown * Level 2 and level 3, to be completed when able to discuss as a team * Preliminary list of pins/features necessary in MCU, and what parts are needed to interface with the various devices -- will be continued to the point where parts can be chosen and calculations can be performed - Very low-level brainstorming of some software needed in the form of state machine diagrams 	<ul style="list-style-type: none"> - Selected potential batteries to supply the robot with power. Created excel sheet to estimate runtime and risk of overcurrent. Found several batteries that meet requirements - Working with mechanical team to find motors that meet their torque requirements but stay within power limitations of batteries - Researching motor controller for weapon motor. Drive motors have controllers built into them.
9/30	<ul style="list-style-type: none"> - Met on Saturday with Tristin and Tanya to work on the Midterm Design Report * Completed Software and Hardware Block diagrams and Functional Requirement Tables * Completed Design Requirements - Created a software diagram (pseudocode) rough draft showing the structure of what I will be coding 	<ul style="list-style-type: none"> - Met with Team 7B to work on midterm design report - Assisted in software block diagram - Split up team tasks - Began reformatting of report - Created rough draft of software block diagram for drive/weapon motor algorithms - Brainstormed/noted actual code implementation idea for drive/weapon motor algorithms 	<ul style="list-style-type: none"> - Met with Dr. Elbuluk to learn more about DC-DC converters for the power system - Produced power system block diagram and pondered electrical safety implementation - Assisted in software block diagram - Met with Team 7A to finalize their "power budget" for how many watts their system is allowed to produce - Formulated power system design requirements
10/7	<ul style="list-style-type: none"> - Edited the decision algorithm pseudocode block diagram based on changes from Team 7A - Helped make changes to decision algorithm - Worked on midterm report and PowerPoint presentation 	<ul style="list-style-type: none"> - Recreated weapon and motion control algorithms based on changes from Team 7A - Helped make changes to decision algorithm - Worked on midterm report and presentation 	<ul style="list-style-type: none"> - Discussed software algorithms - Revisited motor selection
10/14	<ul style="list-style-type: none"> - Helped create poster - Practiced midterm presentation - Helped with the midterm Report - Created a software flowchart (used in the Midterm report and presentation) 	<ul style="list-style-type: none"> - Helped create poster (specifically, wrote software theory of operation) - Practiced for presentation - Worked on how the software algorithm will actually be implemented in code (potentially using binary code and case statements with interrupts as needed) - Researched encoders and how to pick one 	<ul style="list-style-type: none"> - Picked Batteries - Hardware Diagrams - Midpoint Presentation/Report - EE Rep. To COE budget Presentation - Greg's powerpoint
10/21	<ul style="list-style-type: none"> - Worked on Project poster - Prepared for midterm presentation 	<ul style="list-style-type: none"> - Worked on Project poster - Prepared for midterm presentation 	<ul style="list-style-type: none"> - Worked on Project poster - Prepared for midterm presentation
10/28	<ul style="list-style-type: none"> - RC Controller Research 	<ul style="list-style-type: none"> - Updated software theory of operation on 	<ul style="list-style-type: none"> - Design Requirements

	<ul style="list-style-type: none"> - Software “Skeleton” Code - Familiarize with UART 	<ul style="list-style-type: none"> poster - Researched encoders/how they work - Researched designing a filter to convert PWM to DC voltage - to the point where it can be designed 	<ul style="list-style-type: none"> - Switching Converter Lookup - Found Power MOSFETS - Selected Microcontroller - Drive Motor Controllers
11/4	<ul style="list-style-type: none"> - RC Controller Research - Software “Skeleton” Code - Familiarize with UART 	<ul style="list-style-type: none"> - Created control system schematic block diagram - Began schematic for control system - Updated and formatted midterm report - Confirmed team member roles 	<ul style="list-style-type: none"> - Design Requirements - Switching Converter Lookup - Found Power MOSFETS - Selected Microcontroller - Drive Motor Controllers
11/11	<ul style="list-style-type: none"> - RC Controller Research - Software “Skeleton” Code - Familiarize with UART 	<ul style="list-style-type: none"> - Working on finalizing control system schematic and connectors - Test! 	<ul style="list-style-type: none"> - Design Requirements - Switching Converter Simulations - Power System Schematics - Backplane layout - Drive Motor Controller Interfacing
11/20	<ul style="list-style-type: none"> - Worked on the Final Written Report 	<ul style="list-style-type: none"> - Worked on the Final Written Report 	<ul style="list-style-type: none"> - Worked on the Final Written Report

Table 20: Week by Week Updates (9/23 - 11/20)

[AS, TT, TW]

6. Design Team Information

Andrew Szabo, Computer Engineer - Software Lead.

- LED software
- Decision algorithm software
- RC controller interface hardware and software

Tanya Tebcherani, Electrical Engineer - Team Lead, Archivist.

- Control algorithm software
- E-stop software
- Embedded controller system hardware

Tristin Weber, Electrical Engineer - Hardware Lead.

- LED hardware
- Overcurrent protection hardware (and other safety features)
- Power supply design
- Motor controller interface design
- Motor and battery selection

[AS, TT, TW]

7. Conclusion and Recommendations

In conclusion, the majority of the design of the combat robot has been completed. The team has created various hardware and software block diagrams showing the theory behind the team's plan of implementation. Parts have been chosen and parts request forms have been submitted. Schematics and pseudocode have also been created. For the final semester of senior design, Team 7B is ready to move on to physically building the robot in terms of hardware, and programming the PIC in terms of software. Finally, Team 7B must also merge their work with Team 7A and the mechanical team to create a functioning combat robot.

[AS, TT]

8. References

- [1] S. Bachand-Amirault, *Types of Battlebots*. [Online]. Available: <https://sbainvent.com/battlebot-design/battlebot-types.php>. [Accessed: 20-Apr-2018].
- [2] *lidar-uk.com*. [Online]. Available: <http://www.lidar-uk.com/how-lidar-works/>. [Accessed: 20-Apr-2018].
- [3] “What is an Ultrasonic Sensor?,” *Ultrasonic Sensor | What is an Ultrasonic Sensor?*[Online]. Available: http://education.rec.ri.cmu.edu/content/electronics/boe/ultrasonic_sensor/1.html. [Accessed: 20-Apr-2018].
- [4] T. A. Kinney and Baumer Electric, “Proximity Sensors Compared: Inductive, Capacitive, Photoelectric, and Ultrasonic,” *Machine Design*, 13-Mar-2017. [Online]. Available: <http://www.machinedesign.com/sensors/proximity-sensors-compared-inductive-capacitive-photoelectric-and-ultrasonic>. [Accessed: 20-Apr-2018].
- [5] W. by AZoSensors, “What is a Photoelectric Sensor?,” *AZoSensors.com*, 15-Jun-2015. [Online]. Available: <https://www.azosensors.com/article.aspx?ArticleID=311>. [Accessed: 20-Apr-2018].

[6] "Photoelectric sensors," *Balluff*. [Online]. Available:

<https://www.balluff.com/local/us/products/sensors/photoelectric-sensors/>. [Accessed: 20-Apr-2018].

[7] "CN107140047A - Competitive foot combat robot," *Google Patents*. [Online]. Available:

<https://patents.google.com/patent/CN107140047A/en?q=competition&oq=combat+robot+competition>. [Accessed: 20-Apr-2018].

[8] "WO2017143567A1 - Fighting robot," *Google Patents*. [Online]. Available:

<https://patents.google.com/patent/WO2017143567A1/en?q=competition&oq=combat+robot+competition>. [Accessed: 20-Apr-2018].

[9] "KR20050055822A - The weapon system of a battlebot using pneumatic circuit," *Google Patents*. [Online]. Available:

<https://patents.google.com/patent/KR20050055822A/en?q=robot&oq=competition+combat+robot&page=2>. [Accessed: 20-Apr-2018].

[10] V. Magnier, D. Gruyer and J. Godelle, "Automotive LIDAR objects detection and classification algorithm using the belief theory," *2017 IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, CA, 2017, pp. 746-751. <https://ieeexplore.ieee.org/document/7995806/>

[11] A. Börcs, B. Nagy and C. Benedek, "Instant Object Detection in Lidar Point Clouds," in *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 7, pp. 992-996, July 2017.. Available:

<https://ieeexplore.ieee.org/document/7927715/>

[12] Renishaw. (2018). *Renishaw: Application note: Optical encoders and LiDAR scanning*.

[Online] Available: <http://www.renishaw.com/en/optical-encoders-and-lidar-scanning--39244>

[Accessed 25 May 2018].

[13] Wong, W. (2018). *Safe Robots Rely On Sensors*. [Online] Electronic Design. Available:

<http://www.electronicdesign.com/embedded/safe-robots-rely-sensors> [Accessed 25 May 2018].

[14] P. Agharkar and F. Bullo, "Vehicle routing algorithms to intercept escaping targets," *2014 American Control Conference*, Portland, OR, 2014, pp. 952-957.

Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6858759>

[15] *Combat Robot Rules*. [Online]. Available: <http://robogames.net/rules/combat.php>.

[Accessed: 20-Apr-2018].

9. Appendices

9.1 Software Code

9.1.1 Header Files

9.1.1.1 config.h

```
// Configuration Bits

// DEVCFG3
#pragma config USERID = 0xFFFF // Enter Hexadecimal value (Enter Hexadecimal value)
#pragma config FMIIEN = ON // Ethernet RMII/MII Enable (MII Enabled)
#pragma config FETHIO = ON // Ethernet I/O Pin Select (Default Ethernet I/O)
#pragma config PGL1WAY = ON // Permission Group Lock One Way Configuration (Allow only one reconfiguration)
#pragma config PMDL1WAY = ON // Peripheral Module Disable Configuration (Allow only one reconfiguration)
#pragma config IOL1WAY = ON // Peripheral Pin Select Configuration (Allow only one reconfiguration)
#pragma config FUSBIDIO = ON // USB USBID Selection (Controlled by the USB Module)

// DEVCFG2
#pragma config FPLLIDIV = DIV_2 // System PLL Input Divider (2x Divider)
#pragma config FPLL RNG = RANGE_8_16_MHZ // System PLL Input Range (8-16 MHz Input)
#pragma config FPLLCLK = PLL_FRC // System PLL Input Clock Selection (FRC is input to the System PLL)
#pragma config FPLLMULT = MUL_4 //128 // System PLL Multiplier (PLL Multiply by 128)
#pragma config FPLLODIV = DIV_2 // System PLL Output Clock Divider (2x Divider)
#pragma config UPLLFSEL = FREQ_24MHZ // USB PLL Input Frequency Selection (USB PLL input is 24 MHz)

// DEVCFG1
#pragma config FNOSC = FRCDIV // Oscillator Selection Bits (Fast RC Osc w/Div-by-N (FRCDIV))
#pragma config DMTINTV = WIN_127_128 // DMT Count Window Interval (Window/Interval value is 127/128 counter value)
#pragma config FSOSCEN = ON // Secondary Oscillator Enable (Enable SOSC)
#pragma config IESO = ON // Internal/External Switch Over (Enabled)
#pragma config POSCMOD = HS // Primary Oscillator Configuration (HS osc mode)
#pragma config OSCIOFNC = OFF // CLKO Output Signal Active on the OSCO Pin (Disabled)
#pragma config FCKSM = CSDCMD // Clock Switching and Monitor Selection (Clock Switch Enabled, FSCM Enabled)
#pragma config WDTPS = PS1048576 // Watchdog Timer Postscaler (1:1048576)
#pragma config WDTSPGM = STOP // Watchdog Timer Stop During Flash Programming (WDT stops during Flash programming)

#pragma config WINDIS = NORMAL // Watchdog Timer Window Mode (Watchdog Timer is in non-Window mode)
#pragma config FWDTEN = ON // Watchdog Timer Enable (WDT Enabled)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window Size (Window size is 25%)
#pragma config DMTCNT = DMT31 // Deadman Timer Count Selection (2^31 (2147483648))
#pragma config FDMTEN = ON // Deadman Timer Enable (Deadman Timer is enabled)

// DEVCFG0
#pragma config DEBUG = OFF // Background Debugger Enable (Debugger is disabled)
#pragma config JTAGEN = ON // JTAG Enable (JTAG Port Enabled)
#pragma config ICESEL = ICS_PGx1 // ICE/ICD Comm Channel Select (Communicate on PGEC1/PGED1)
#pragma config TRCEN = ON // Trace Enable (Trace features in the CPU are enabled)
#pragma config BOOTISA = MIPS32 // Boot ISA Selection (Boot code and Exception code is MIPS32)
#pragma config FECCON = OFF_UNLOCKED // Dynamic Flash ECC Configuration (ECC and Dynamic ECC are disabled (FECCON bits are writable))

#pragma config FSLEEP = OFF // Flash Sleep Mode (Flash is powered down when the device is in Sleep mode)
#pragma config DBGPER = PG_ALL // Debug Mode CPU Access Permission (Allow CPU access to all permission regions)
#pragma config SMCLR = MCLR_NORM // Soft Master Clear Enable bit (MCLR pin generates a normal system Reset)
```

```
#pragma config SOSC_GAIN = GAIN_2X // Secondary Oscillator Gain Control bits (2x gain setting)
#pragma config SOSCBOOST = ON // Secondary Oscillator Boost Kick Start Enable bit (Boost the kick start of the oscillator)
#pragma config POSC_GAIN = GAIN_2X // Primary Oscillator Gain Control bits (2x gain setting)
#pragma config POSCBOOST = ON // Primary Oscillator Boost Kick Start Enable bit (Boost the kick start of the oscillator)
#pragma config EJTAG_BEN = NORMAL // EJTAG Boot (Normal EJTAG functionality)

// DEVCP0
#pragma config CP = OFF // Code Protect (Protection Disabled)

// SEQ3
#pragma config TSEQ = 0xFFFF // Boot Flash True Sequence Number (Enter Hexadecimal value)
#pragma config CSEQ = 0xFFFF // Boot Flash Complement Sequence Number (Enter Hexadecimal value)

// DEVADC0
#pragma config ADCFG = 0x0FFFFFFF // Enter Hexadecimal value (Enter Hexadecimal value)

// DEVSNO
#pragma config SN = 0x0FFFFFFF // Enter Hexadecimal value (Enter Hexadecimal value)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.
```

9.1.1.2 defines.h

```
#define SYS_CLK 8000000L
#define PWM_FREQ 20000
```

9.1.1.3 functions.h

```
// UART4
void init_UART4(void);
char rx4(void);

// IC1_y
void init_IC1(void);
float read_buf_IC1(void);
float calc_perc_IC1(float);
float y_out(void);

// IC2_x
void init_IC2(void);
float read_buf_IC2(void);
float calc_perc_IC2(float);
float x_out(void);

// IC3_w
void init_IC3(void);
float read_buf_IC3(void);
float w_out(void);

// IC4_estop
void init_IC4(void);
float read_buf_IC4(void);
float estop_out(void);

// IC5_am
void init_IC5(void);
float read_buf_IC5(void);
float aut_out(void);

// IC_Misc
void init_IC(void);
float calc_ms(float);

// OC1_RDM_PWM
void init_rdm_pwm(void);
void dc_rdm_pwm(float);
void dig_rdm_pwm(int);

// OC2_LDM_PWM
```

```
void init_ldm_pwm(void);
void dc_ldm_pwm(float);
void dig_ldm_pwm(int);

// OC3_WM_PWM
void init_wm_pwm(void);
void dc_wm_pwm(float);
void dig_wm_pwm(int);

// Auto_Mode
void autonomous_control(short int, short int);

// Man_Mode
void manual_RC_control(float, float);

// Misc
void init_osc(void);
void delay(int);
void estop(void);

// LEDs
void init_LEDs(void);
void LED_attack(int);
void LED_estop(int);
void LED_autonomy(int);
void LED_power(int);
```

9.1.2 Source Files

9.1.2.1 Auto_Mode.c

```
// Include Header Files
#include <stdlib.h>
#include <math.h>    //for fabs, sin, cos functions
#include "functions.h"
#include "defines.h"

void autonomous_control(short int th, short int dr) {
    float theta = th;
    float drive = dr;
    float rad;

    float A = 0;
    float B = 0;
    float R = 0;
    float L = 0;

    float X;
    float Y;

    float rm_speed;
    float lm_speed;

    // Convert theta to radians
    rad = ((M_PI/180) * theta) - M_PI/2;

    // Calculate X & Y from angle
    // if (theta < -90 || theta > 90) {
    if (theta < 0 || theta > 180) {
        X = 0;
        Y = 0;
    }
    //else if (theta >= -90 && theta <= -45) {
    else if (theta >= 0 && theta <= 45) {
        X = -100;
        Y = 100 * tan(M_PI/2 + rad);
    }
}
```

```

//else if (theta > -45 && theta < 0) {
else if (theta > 45 && theta < 90) {
    X = -100 * tan(-1 * rad);
    Y = 100;
}
//else if (theta >= 0 && theta <= 45) {
else if (theta >= 90 && theta <= 135) {
    X = 100 * tan(rad);
    Y = 100;
}

//else if (theta > 45 && theta <= 90) {
else if (theta > 135 && theta <= 180) {
    X = 100;
    Y = 100 * tan(M_PI/2 - rad);
}

//NEED TO ADD A FEW STEPS FOR SCALING IN HERE
X = (X)*(-1);

// If going backwards, flip y-axis
if (drive == 180) {
    Y = (Y)*(-1);
}

A = (100-fabs(X)) * (Y/100) + Y; //intermittent calculations
B = (100-fabs(Y)) * (X/100) + X;
R = (A + B) / 2;           //% Left and % Right Found Here
L = (A - B) / 2;

//Determine motor speeds and output to global variable
rm_speed = (R * 0.01);
lm_speed = (L * 0.01);

// Output PWM
dc_rdm_pwm(rm_speed);
dc_ldm_pwm(lm_speed);
}

```


9.1.2.2 IC1_y.c

```
#include <p32xxxx.h>
#include "functions.h"

// Initialize IC1
void init_IC1(void) {
    // Initialize RPD1 for IC1
    TRISDbits.TRISD1 = 1;    // Input
    IC1Rbits.IC1R = 0b0000; // IC1 on RPD1 (P51), elevator

    // Initialize IC1
    IC1CONbits.SIDL = 0;
    IC1CONbits.FEDGE = 1;
    IC1CONbits.C32 = 0;
    IC1CONbits.ICTMR = 0;
    IC1CONbits.ICI = 0b01; //
    IC1CONbits.ICM = 0b110; // change to 001 for not simple mode // Capture every rising and
                                                                    falling edge

    IC1CONbits.ON = 1;

    IFS0CLR = _IFS0_IC1IF_MASK;
    IPC1bits.IC1IP = 6;    // Priority (highest?)
    IPC1bits.IC1IS = 0;    // Sub priority?
    IEC0bits.IC1IE = 1;    // Enable IC1 interrupts
}

// Read IC1 buffer
float read_buf_IC1(void) {
    float ic1_1;
    float ic1_2;
    float ic1_t;

    // Read timer value at rising edge
    ic1_1 = IC1BUF;
    // Read timer value at falling edge
    ic1_2 = IC1BUF;
    // Clear interrupt flag
    IFS0CLR = _IFS0_IC1IF_MASK;
    // Calculate time high
    ic1_t = ic1_2 - ic1_1;

    return ic1_t;
}
```

```

}

// Convert time high (ms) to percent (range from -100 to 100%)
float calc_perc_IC1(float ms_val_IC1) {
    float perc_IC1;

    // (ms value - center value) / [(highest ms value - center ms value) / 100]
    // Only one equation is needed because lowest and highest ms values from the center are the
    // same (symmetric)
    perc_IC1 = (ms_val_IC1-1.5)/0.006;

    if (perc_IC1 < 1 && perc_IC1 > -1) {
        perc_IC1 = 0.001;
    }

    return perc_IC1;
}

// Output percent for y-axis on RC controller
float y_out(void) {
    float tmr_val_IC1;
    float tmr_val_ms_IC1;
    float y_output;

    // If flag is high
    if (IFS0bits.IC1IF == 1) {
        // Read buffer for time high
        tmr_val_IC1 = read_buf_IC1();

        // If timer didn't overflow
        if (tmr_val_IC1 > 0) {
            // Convert time high (timer value) to ms
            tmr_val_ms_IC1 = calc_ms(tmr_val_IC1);
            // Convert ms to %
            y_output = (calc_perc_IC1(tmr_val_ms_IC1));

            return y_output;
        }
    }
}

```

9.1.2.3 IC2_x.c

```
#include <p32xxxx.h>
#include "functions.h"

// Initialize IC2
void init_IC2(void) {
    // Initialize RPD4 for IC2
    TRISDbits.TRISD4 = 1;    // Input
    IC2Rbits.IC2R = 0b0100; // IC2 on RPD4 (P40), aileron

    // Initialize IC1
    IC2CONbits.SIDL = 0;
    IC2CONbits.FEDGE = 1;
    IC2CONbits.C32 = 0;
    IC2CONbits.ICTMR = 0;
    IC2CONbits.ICI = 0b01; //
    IC2CONbits.ICM = 0b110; // change to 001 for not simple mode // Capture every rising and
    falling edge
    IC2CONbits.ON = 1;

    IFS0CLR = _IFS0_IC2IF_MASK;
    IPC2bits.IC2IP = 5; // Priority (highest?)
    IPC2bits.IC2IS = 0; // Sub priority?
    IEC0bits.IC2IE = 1; // Enable IC2 interrupts
}

// Read IC2 buffer
float read_buf_IC2(void) {
    float ic2_1;
    float ic2_2;
    float ic2_t;

    // Read timer value at rising edge
    ic2_1 = IC2BUF;
    // Read timer value at falling edge
    ic2_2 = IC2BUF;
    // Clear interrupt flag
    IFS0CLR = _IFS0_IC2IF_MASK;
    // Calculate time high
    ic2_t = ic2_2 - ic2_1;

    return ic2_t;
}
```

```

}

// Convert time high (ms) to percent (range from -100 to 100%)
float calc_perc_IC2(float ms_val_IC2) {
    float perc_IC2;

    // If joystick above center
    if (ms_val_IC2 > 1.55) {
        // (ms value - center value) / [(highest ms value - center ms value) / 100]
        perc_IC2 = (ms_val_IC2-1.55)/0.0055;
    }
    // If joystick below center
    else {
        // (ms value - center value) / [(lowest ms value - center ms value) / 100]
        perc_IC2 = (ms_val_IC2-1.55)/0.0066;
    }

    if (perc_IC2 < 1 && perc_IC2 > -1) {
        perc_IC2 = 0.001;
    }

    return perc_IC2;
}

// Output percent for y-axis on RC controller
float x_out(void) {
    float tmr_val_IC2;
    float tmr_val_ms_IC2;
    float x_output;

    // If flag is high
    if (IFS0bits.IC2IF == 1) {
        // Read buffer for time high
        tmr_val_IC2 = read_buf_IC2();
    }
}

```

```
// If timer didn't overflow
if (tmr_val_IC2 > 0) {
    // Convert time high (timer value) to ms
    tmr_val_ms_IC2 = calc_ms(tmr_val_IC2);
    // Convert ms to %
    x_output = calc_perc_IC2(tmr_val_ms_IC2);

    return x_output;
}
}
```

9.1.2.4 IC3_w.c

```
#include <p32xxxx.h>
#include "functions.h"

// Initialize IC3
void init_IC3(void) {
    // Initialize RPD2 for IC3
    TRISDbits.TRISD2 = 1;    // Input
    IC3Rbits.IC3R = 0b0000; // IC3 on RPD2 (P6), gear

    // Initialize IC3
    IC3CONbits.SIDL = 0;
    IC3CONbits.FEDGE = 1;
    IC3CONbits.C32 = 0;
    IC3CONbits.ICTMR = 0;
    IC3CONbits.ICI = 0b01; //
    IC3CONbits.ICM = 0b110; // change to 001 for not simple mode // Capture every rising and
falling edge
    IC3CONbits.ON = 1;

    IFS0CLR = _IFS0_IC3IF_MASK;
    IPC3bits.IC3EIP = 4; // Priority (highest?)
    IPC3bits.IC3EIS = 0; // Sub priority?
    IEC0bits.IC3IE = 1; // Enable IC1 interrupts
}

// Read IC3 buffer
float read_buf_IC3(void) {
    float ic3_1;
    float ic3_2;
    float ic3_t;

    // Read timer value at rising edge
    ic3_1 = IC3BUF;
    // Read timer value at falling edge
    ic3_2 = IC3BUF;
    // Clear interrupt flag
    IFS0CLR = _IFS0_IC3IF_MASK;
    // Calculate time high
    ic3_t = ic3_2 - ic3_1;

    return ic3_t;
}
```

```

}

// Output percent for y-axis on RC controller
float w_out(void) {
    float tmr_val_IC3;

    // If flag is high
    if (IFS0bits.IC3IF == 1) {
        // Read buffer for time high
        tmr_val_IC3 = read_buf_IC3();

        // If timer didn't overflow
        if (tmr_val_IC3 > 0) {
            if (tmr_val_IC3 > 4300 && tmr_val_IC3 < 4350) {
                return 1;
            }
            else if (tmr_val_IC3 > 7700 && tmr_val_IC3 < 7750) {
                return 2;
            }
        }
    }
}
}

```

9.1.2.5 IC4_estop.c

```
#include <p32xxxx.h>
#include "functions.h"

// Initialize IC4
void init_IC4(void) {
    // Initialize RPD3 for IC4
    TRISDbits.TRISD3 = 1;    // Input
    IC4Rbits.IC4R = 0b0000; // IC4 on RPD3 (P67), throttle

    // Initialize IC4
    IC4CONbits.SIDL = 0;
    IC4CONbits.FEDGE = 1;
    IC4CONbits.C32 = 0;
    IC4CONbits.ICTMR = 0;
    IC4CONbits.ICI = 0b01; //
    IC4CONbits.ICM = 0b110; // change to 001 for not simple mode // Capture every rising and
falling edge
    IC4CONbits.ON = 1;

    IFS0CLR = _IFS0_IC4IF_MASK;
    IPC5bits.IC4IP = 3; // Priority (highest?)
    IPC5bits.IC4IS = 0; // Sub priority?
    IEC0bits.IC4IE = 1; // Enable IC4 interrupts
}

// Read IC4 buffer
float read_buf_IC4(void) {
    float ic4_1;
    float ic4_2;
    float ic4_t;

    // Read timer value at rising edge
    ic4_1 = IC4BUF;
    // Read timer value at falling edge
    ic4_2 = IC4BUF;
    // Clear interrupt flag
    IFS0CLR = _IFS0_IC4IF_MASK;
    // Calculate time high
    ic4_t = ic4_2 - ic4_1;

    return ic4_t;
}
```



```

}

// Output percent for y-axis on RC controller
float estop_out(void) {
    float tmr_val_IC4;

    // If flag is high
    if (IFS0bits.IC4IF == 1) {
        // Read buffer for time high
        tmr_val_IC4 = read_buf_IC4();

        // If timer didn't overflow
        if (tmr_val_IC4 > 0) {
            if (tmr_val_IC4 >= 7080) {
                return 1;
            }
            else if (tmr_val_IC4 <= 4960) {
                return 2;
            }
            else {
                return 3;
            }
        }
    }
}

```

9.1.2.6 IC5_am.c

```
#include <p32xxxx.h>
#include "functions.h"

// Initialize IC5
void init_IC5(void) {
    // Initialize RPD9 for IC5
    TRISDbits.TRISD9 = 1;    // Input
    IC5Rbits.IC5R = 0b0000; // IC5 on RPD9 (P70) rutter

    // Initialize IC5
    IC5CONbits.SIDL = 0;
    IC5CONbits.FEDGE = 1;
    IC5CONbits.C32 = 0;
    IC5CONbits.ICTMR = 0;
    IC5CONbits.ICI = 0b01; //
    IC5CONbits.ICM = 0b110; // change to 001 for not simple mode // Capture every rising and
                                                                    falling edge

    IC5CONbits.ON = 1;

    IFS0CLR = _IFS0_IC5IF_MASK;
    IPC6bits.IC5EIP = 2; // Priority (highest?)
    IPC6bits.IC5EIS = 0; // Sub priority?
    IEC0bits.IC5IE = 1; // Enable IC4 interrupts
}

// Read IC5 buffer
float read_buf_IC5(void) {
    float ic5_1;
    float ic5_2;
    float ic5_t;

    // Read timer value at rising edge
    ic5_1 = IC5BUF;
    // Read timer value at falling edge
    ic5_2 = IC5BUF;
    // Clear interrupt flag
    IFS0CLR = _IFS0_IC5IF_MASK;
    // Calculate time high
    ic5_t = ic5_2 - ic5_1;

    return ic5_t;
}
```

```

}

// Output percent for y-axis on RC controller
float aut_out(void) {
    float tmr_val_IC5;

    // If flag is high
    if (IFS0bits.IC5IF == 1) {
        // Read buffer for time high
        tmr_val_IC5 = read_buf_IC5();

        // If timer didn't overflow
        if (tmr_val_IC5 > 0) {

            //autonomy = right on joystick
            if (tmr_val_IC5 >= 7160) {
                return 2;
            }
            //manual = left on joystick
            else if (tmr_val_IC5 <= 4570) {
                return 1;
            }
            else {
                return 3;
            }
        }
    }
}

```

9.1.2.7 IC_Misc.c

```
#include <p32xxxx.h>
#include "defines.h"

void init_IC(void) {
    // Initialize TMR3
    T3CONbits.SIDL = 0;
    T3CONbits.TGATE = 0;
    T3CONbits.TCKPS = 0b000; // 1:1 prescale value (2 MHz / 1 = timer ticks at 2 MHz)
    T3CONbits.ON = 1;

    PR3 = 0xFFFF;

    // Multi-vector interrupts
    INTCONbits.MVEC = 1;

    init_IC1();
    init_IC2();
    init_IC3();
    init_IC4();
    init_IC5();
}

float calc_ms(float t_val) {
    float ms_val;
    int long tmr_clk;

    tmr_clk = SYS_CLK / 2;
    ms_val = t_val/tmr_clk * 1000;

    return ms_val;
}
```

9.1.2.8 LEDs.c

```
// Include Header Files
#include <p32xxxx.h>
#include "functions.h"

//Global Variables
extern float wm_current_speed;
extern float filt_aut;

void init_LEDs(void) {
    // Initialize digital output
    TRISEbits.TRISE1 = 0;    // Output (P61)
    TRISEbits.TRISE2 = 0;    // Output (P62)
    TRISEbits.TRISE3 = 0;    // Output (P63)
}

// Turn E-stop LED on
void LED_estop(int on){
    if (on == 1) {
        PORTEbits.RE2 = 0;
    }
    else if (on == 0) {
        PORTEbits.RE2 = 1;
    }
}

// Turn Autonomy LED on
void LED_autonomy(int on){
    if (on == 1) {
        PORTEbits.RE3 = 0;
    }
    else if (on == 0) {
        PORTEbits.RE3 = 1;
    }
}

// Turn Power LED on
void LED_power(int on){
    if (on == 1) {
        PORTEbits.RE1 = 0;
    }
    else if (on == 0) {
        PORTEbits.RE1 = 1;
    }
}
```

9.1.2.9 Man_Mode.c

```
// Include Header Files
#include <stdlib.h>
#include <math.h>    //for fabs()
#include "functions.h"

extern float upside_down;

//Convert Controller Right Joystick to 2-Wheel Motion
void manual_RC_control(float X, float Y) {
    float A = 0;
    float B = 0;
    float R = 0;
    float L = 0;

    float rm_speed;
    float lm_speed;

    // If upside down, flip y-axis
    if (upside_down == 1) {
        Y = Y*(-1);
    }

    if (Y < -60){        //Fixes reverse turns. - Tristin
        X = X*(-1);
    }
    //NEED TO ADD A FEW STEPS FOR SCALING IN HERE
    X = (X)*(-1);

    A = (100-fabs(X)) * (Y/100) + Y;    //intermittent calculations
    B = (100-fabs(Y)) * (X/100) + X;
    R = (A + B) / 2;                    //% Left and % Right Found Here
    L = (A - B) / 2;

    //Determine motor speeds and output to global variable
    rm_speed = (R * 0.01);
    lm_speed = (L * 0.01);

    // Output PWM
    dc_rdm_pwm(rm_speed);
    dc_ldm_pwm(lm_speed);
}
```

9.1.2.10 Misc.c

```
#include <p32xxxx.h>
#include <stdlib.h>
#include <math.h>    //for fabs()
#include "defines.h"
extern float rdm_current_speed;
extern float ldm_current_speed;
extern float wm_current_speed;
extern float rdm_inc_val;
extern float ldm_inc_val;
extern float wm_inc_val;
extern float filt_aut;

// Initialize oscillator to 8MHz
void init_osc(void) {
    OSCCONbits.COSC = 0b111; // 8 MHz clock, maybe?
    OSCTUN = 0;           // Keep it at 8 MHz, don't tune it up or down
}

// Delay of unknown time
void delay(int num) {
    int i = 0;

    while (i < 125000 * num) {
        i++;
    };
}

void estop(void) {
    float duty = 0;

    LED_estop(1);

    while (rdm_current_speed != 0) {
        //added by tristin for soft start
        if (duty < (rdm_current_speed - rdm_inc_val/100)) {
            rdm_current_speed = rdm_current_speed - rdm_inc_val/100;
        }
        else if (duty > (rdm_current_speed + rdm_inc_val/100)){
            rdm_current_speed = rdm_current_speed + rdm_inc_val/100;
        }
    }
}
```

```

else {
    rdm_current_speed = duty;
}

OC1RS = (PR2 + 1) * (fabs(rdm_current_speed));
}

while(ldm_current_speed != 0) {
    //added by tristin for soft start
    if (duty < (ldm_current_speed - ldm_inc_val/100)) {
        ldm_current_speed = ldm_current_speed - ldm_inc_val/100;
    }
    else if (duty > (ldm_current_speed + ldm_inc_val/100)){
        ldm_current_speed = ldm_current_speed + ldm_inc_val/100;
    }
    else {
        ldm_current_speed = duty;
    }

    OC2RS = (PR2 + 1) * (fabs(ldm_current_speed));
}
OC9RS = 0;
/*
while(wm_current_speed != 0) {

    //added by tristin for soft start
    if (duty < (wm_current_speed - wm_inc_val)) {
        wm_current_speed = wm_current_speed - wm_inc_val;
    }
    else if (duty > (wm_current_speed + wm_inc_val)){
        wm_current_speed = wm_current_speed + wm_inc_val;
    }
    else {
        wm_current_speed = duty;
    }

    OC9RS = (PR2 + 1) * (fabs(wm_current_speed));

}*/
}

```


9.1.2.11 newmain1.c

```
// Include Header Files
#include <proc/p32mz2048efh100.h>

#include "config.h"
#include "defines.h"
#include "functions.h"

//Global Variables
float estp;
float upside_down = 0; // Start right-side up (0 or 1)
float filt_aut = 1; // Start in manual mode (1 or 2)
float spec = 0;

float rdm_current_speed = 0;
float ldm_current_speed = 0;
float wm_current_speed = 0;

float rdm_inc_val;
float ldm_inc_val;
float wm_inc_val;

int main(void) {
    // RC Controller Variables
    float RC_x;
    float RC_y;
    float RC_w;
    float RC_aut = 0;
    float RC_estop = 1; // (1 or 2)?

    // Filter variables
    float filt_RC_x = 0;
    float filt_RC_y = 0;

    // Aut Variables
    int i;
    short int degree;
    short int drive;

    // Extra variables
    float act_estop = 0; // E-stop activation feature (DON'T TOUCH THIS!!!)
    char rec = 'p'; // Variable that stores the character UART receives, default is stop
}
```

```

int reci;

// Initializations
init_osc();
init_IC();
init_rdm_pwm();
init_ldm_pwm();
init_wm_pwm();
init_LEDs();

init_UART4();

while(1) {
    // Set power LED high to show PIC is functioning
    LED_power(1);

    // Read from RC Controller
    RC_x = x_out();
    RC_y = y_out();
    RC_w = w_out();
    RC_aut = aut_out();
    RC_estop = estop_out();

    // Read UART
    if (IFS5bits.U4RXIF == 1) {
        reci = rx4();

        if ((reci > 179) && (reci != 183)) {
            drive = reci;
        }
        else {
            degree = reci;
        }

        IFS5CLR = _IFS5_U4RXIF_MASK;
    }

    // Read UART
    if (IFS5bits.U4RXIF == 1) {
        reci = rx4();
    }
}

```

```

    if ((reci > 179) && (reci != 183)) {
        drive = reci;
    }
    else {
        degree = reci;
    }

    IFS5CLR = _IFS5_U4RXIF_MASK;
}

/*
// Search mode
if (rec == 's') {
    degree = 90;
    drive = 181;
}
// Upside down
else if (rec == 'u') {
    degree = -90;
    drive = 181;
}
// Stop
else if (rec == 'p') {
    degree = 0;
    drive = 181;
}
// Full speed forward
else if (rec == 'f') {
    degree = 0;
    drive = 182;
}
// Full speed backwards (reverse)
else if (rec == 'b') {
    degree = 0;
    drive = 180;
}
// 45 degrees to the right, forward
else if (rec == 'r') {
    degree = 45;
    drive = 182;
}
// 45 degrees to the left, forward
else if (rec == 'l') {

```

```

degree = -45;
drive = 182;
}
*/

// If e-stop functionality off
if (act_estop == 0) {
    // Set e-stop functionality to on.
    if (RC_estop == 2) {
        act_estop = 1;
    }
    // No e-stop, continue with code
    else {
        // If upside down, manual mode
        if (upside_down == 1) {
            filt_aut = 1;
        }
        // If right-side up, continue
        else { // upside_down == 0
            // Set autonomy or manual
            if(RC_aut == 1 || RC_aut == 2) {
                filt_aut = RC_aut;
            }
        }
    }

    // Set weapon on or off
    if ((RC_w == 1 || RC_w == 2)) {
        dc_wm_pwm(RC_w);
    }

    // Set soft start increments
    if (filt_aut == 1) { // Manual mode
        wm_inc_val = 0.03;
        rdm_inc_val = 0.03;
        ldm_inc_val = 0.03;
    }
    else { // Autonomous mode
        wm_inc_val = 0.03;
        rdm_inc_val = 0.03;
        ldm_inc_val = 0.03;
    }
}

```

```

// Manual mode
if (filt_aut == 1) {
    // Turn autonomy LED off
    LED_autonomy(0);

    // Filter x & y components
    if (RC_x != 0.000000) {
        filt_RC_x = RC_x;
    }

    if (RC_y != 0.000000) {
        filt_RC_y = RC_y;
    }

    // Set drive motor speeds
    if (filt_RC_x != 0 && filt_RC_y != 0) {
        manual_RC_control(filt_RC_x, filt_RC_y);

        // Reset filtering variables
        filt_RC_x = 0;
        filt_RC_y = 0;
    }
}

// Autonomous mode
else { // filt_aut == 2
    // Turn autonomy LED on
    LED_autonomy(1);

    // Robot is upside down
    //if (degree == -90 && drive == 181) {
    if (degree == 183 && drive == 181) {
        spec = 1;
        upside_down = 1;
        dc_wm_pwm(1);
    }

    // Stop
    //else if (degree == 183 && drive == 181) {
    else if (degree == 90 && drive == 181) {
        spec = 1;
        dc_rdm_pwm(0);
        dc_ldm_pwm(1);
        //dc_wm_pwm(7700);
    }
}

```


9.1.2.12 OC1_RDM_PWM.c

```
// Include Header Files
#include <stdlib.h>
#include <math.h>    //for fabs()
#include <p32xxxx.h>
#include "defines.h"
#include "functions.h"

//Global Variables
extern float rdm_current_speed;
extern float rdm_inc_val;
extern float filt_aut;
extern float spec;

void init_rdm_pwm(void) {
    // Initialize digital output
    ANSELEbits.ANSE4 = 0;    // No analog
    TRISEbits.TRISE4 = 0;    // Output (P64)

    /*
    ANSELBbits.ANSB13 = 0;    // No analog
    TRISBbits.TRISB13 = 0;    // Output (P28)
    */

    // Initialize RPB14 for OC1
    ANSELBbits.ANSB14 = 0;    // No analog
    TRISBbits.TRISB14 = 0;    // Output
    RPB14Rbits.RPB14R = 0b1100; // P29

    // Initialize OC1
    OC1CONbits.SIDL = 0;
    OC1CONbits.OC32 = 0;
    OC1CONbits.OCTSEL = 0;
    OC1CONbits.OCM = 0b110;
    OC1CONbits.ON = 1;

    // Set motor speed off
    PR2 = (SYS_CLK / (2 * PWM_FREQ)) - 1;
    OC1RS = 0;
}
```

```

// Set weapon motor duty cycle
void dc_rdm_pwm(float duty) {
    if (duty > 0.15) {
        duty = 0.15;
    }
    else if (duty < -0.15) {
        duty = -0.15;
    }

    //added by tristin for soft start
    if (duty < (rdm_current_speed - rdm_inc_val)) {
        rdm_current_speed = rdm_current_speed - rdm_inc_val;
    }
    else if (duty > (rdm_current_speed + rdm_inc_val)){
        rdm_current_speed = rdm_current_speed + rdm_inc_val;
    }
    else {
        rdm_current_speed = duty;
    }

    if (filt_aut == 1 || (filt_aut == 2 && spec == 0)) {
        //Determine direction of motors "0" for reverse "1" for forward - CAN BE ALTERED IF
        //NEEDED
        if (rdm_current_speed >= 0) {
            dig_rdm_pwm(1);
        }
        else {
            dig_rdm_pwm(0);
        }
    }

    OC1RS = (PR2 + 1) * (fabs(rdm_current_speed));
}

// Set direction of weapon (1 = forward, 0 = backward)
void dig_rdm_pwm(int forward) {
    PORTEbits.RE4 = forward;
}

```


9.1.2.13 OC2_LDM_PWM.c

```
// Include Header Files
#include <stdlib.h>
#include <math.h>    //for fabs()
#include <p32xxx.h>
#include "defines.h"
#include "functions.h"

extern float ldm_current_speed;
extern float ldm_inc_val;
extern float filt_aut;
extern float spec;

void init_ldm_pwm(void) {
    // Initialize digital output
    ANSELBbits.ANSB9 = 0;    // No analog
    TRISBbits.TRISB9 = 0;    // Output (P28)

    // Initialize RPD5 for OC2 (THIS SHOULD BE RPB6 ON CONTROL BOARD)
    //ANSELDbits.ANSD5 = 0;    // No analog
    TRISBbits.TRISB6 = 0;    // Output
    RPB6Rbits.RPB6R = 0b1011; // P17

    // Initialize OC1
    OC2CONbits.SIDL = 0;
    OC2CONbits.OC32 = 0;
    OC2CONbits.OCTSEL = 0;
    OC2CONbits.OCM = 0b110;
    OC2CONbits.ON = 1;

    // Set motor speed off
    PR2 = (SYS_CLK / (2 * PWM_FREQ)) - 1;
    OC2RS = 0;
}

// Set weapon motor duty cycle
void dc_ldm_pwm(float duty) {
    if (duty > 0.15) {
        duty = 0.15;
    }
    else if (duty < -0.15) {
        duty = -0.15;
    }
}
```

```

//added by tristin for soft start
if (duty < (ldm_current_speed - ldm_inc_val)) {
    ldm_current_speed = ldm_current_speed - ldm_inc_val;
}
else if (duty > (ldm_current_speed + ldm_inc_val)){
    ldm_current_speed = ldm_current_speed + ldm_inc_val;
}
else {
    ldm_current_speed = duty;
}

if (filt_aut == 1 || (filt_aut == 2 && spec == 0)) {
    //Determine direction of motors "0" for reverse "1" for forward - CAN BE ALTERED IF
    NEEDED
    if (ldm_current_speed >= 0) {
        dig_ldm_pwm(0);
    }
    else {
        dig_ldm_pwm(1);
    }
}

OC2RS = (PR2 + 1) * (fabs(ldm_current_speed));
}

// Set direction of left drive motor (1 = forward, 0 = backward)
void dig_ldm_pwm(int forward) {
    PORTBbits.RB9 = forward;
}

```

9.1.2.14 OC3_WM_PWM.c

```
// Include Header Files
#include <stdlib.h>
#include <math.h>    //for fabs()
#include <p32xxx.h>
#include "defines.h"
#include "functions.h"

//Global Variables
extern float wm_current_speed;
extern float wm_inc_val;
extern float filt_aut;

// Initialize weapon motor PWM module
void init_wm_pwm(void) {
    // Initialize PWM pins
    TRISDbits.TRISD5 = 0;    // Output
    RPD5Rbits.RPD5R = 0b1101; // P23

    // Initialize oscillator
    OSCCONbits.COSC = 0b111; // 8 MHz clock, maybe?
    OSCTUN = 0;             // Keep it at 8 MHz, don't tune it up or down

    // Initialize Timer2
    // Input to timer 2 is clock frequency / 4 (8 MHz/4 = 2 MHz)
    T2CONbits.SIDL = 0;
    T2CONbits.TGATE = 0;
    T2CONbits.TCKPS = 0b000; // 1:1 prescale value (2 MHz / 1 = timer ticks at 2 MHz)
    T2CONbits.T32 = 0;
    T2CONbits.ON = 1;

    // Initialize OC9
    OC9CONbits.SIDL = 0;
    OC9CONbits.OC32 = 0;
    OC9CONbits.OCTSEL = 0;
    OC9CONbits.OCM = 0b110;
    OC9CONbits.ON = 1;

    // Set motor speed off
    PR2 = (SYS_CLK / (2 * PWM_FREQ)) - 1;
    OC9RS = 0;
}
```

```

// Set weapon motor duty cycle
void dc_wm_pwm(float on) {

    float duty;

    if (on == 1) {
        duty = 0;
    }
    // 7700
    // 2
    else if ((on != 1) && (on != 0)) {
        duty = 0.10;
    }

    /*
    //added by tristin for soft start
    if (duty < (wm_current_speed - wm_inc_val)) {
        wm_current_speed = wm_current_speed - wm_inc_val;
    }
    else if (duty > (wm_current_speed + wm_inc_val)){
        wm_current_speed = wm_current_speed + wm_inc_val;
    }
    else {
        wm_current_speed = duty;
    }

    OC9RS = (PR2 + 1) * (fabs(wm_current_speed));
    */

    OC9RS = (PR2 + 1) * duty;

}

```

9.1.2.15 UART4.c

```
// Include Header Files
#include <p32xxxx.h>
#include "defines.h"

// Initialize UART1 module
void init_UART4(void) {
    // Initialize pin for RX
    U4RXRbits.U4RXR = 0b1000; // Set UART1 RX to RPF3
    TRISFbits.TRISF3 = 1;    // Input

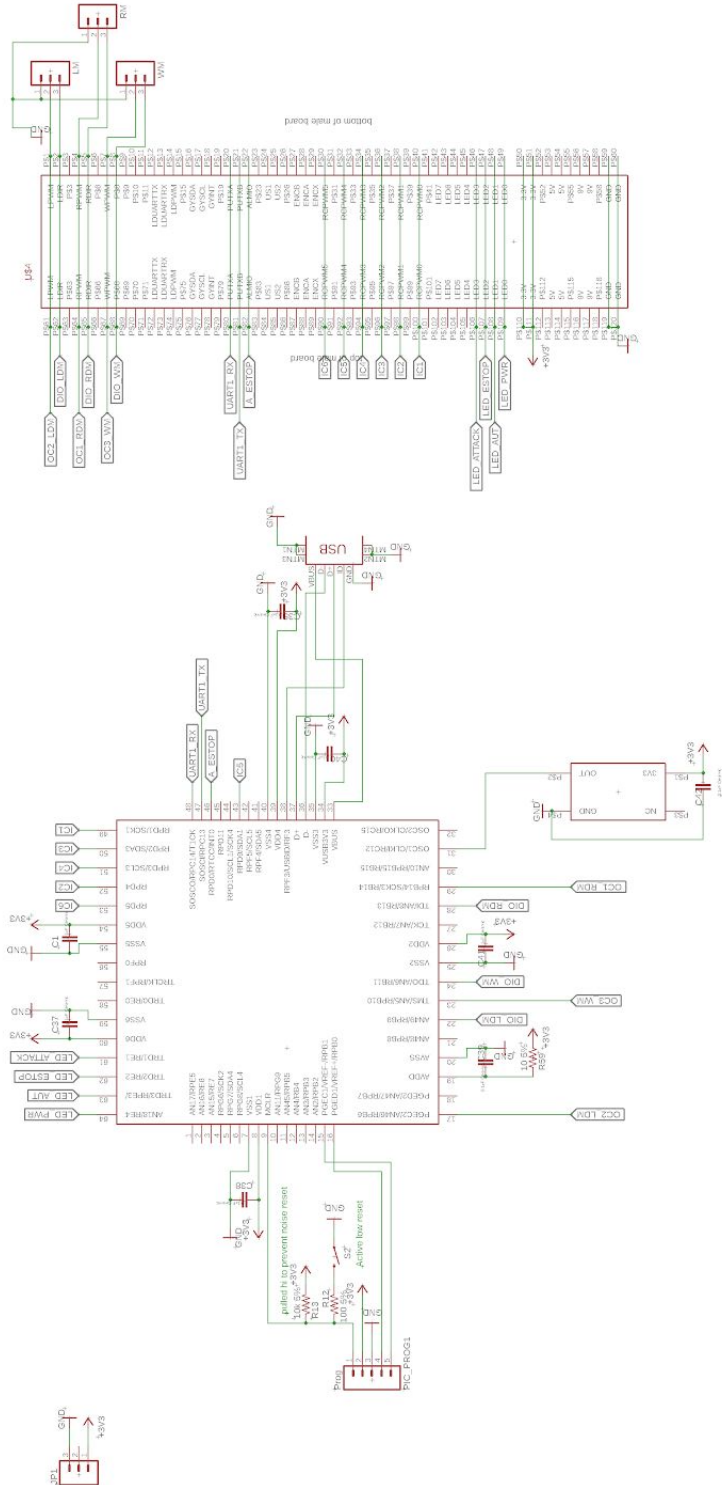
    //U1MODEbits.SIDL = 0;
    int long pbClk = SYS_CLK / 2;
    U4MODEbits.BRGH = 0;
    U4BRG = pbClk / (16 * 9600) - 1;
    U4STAbits.UTXEN = 1;    // Enable TX pin
    U4STAbits.URXEN = 1;    // Enable RX pin
    U4MODEbits.PDSEL = 0b00;
    U4MODEbits.STSEL = 0;
    U4MODEbits.ON = 1;     // Enable

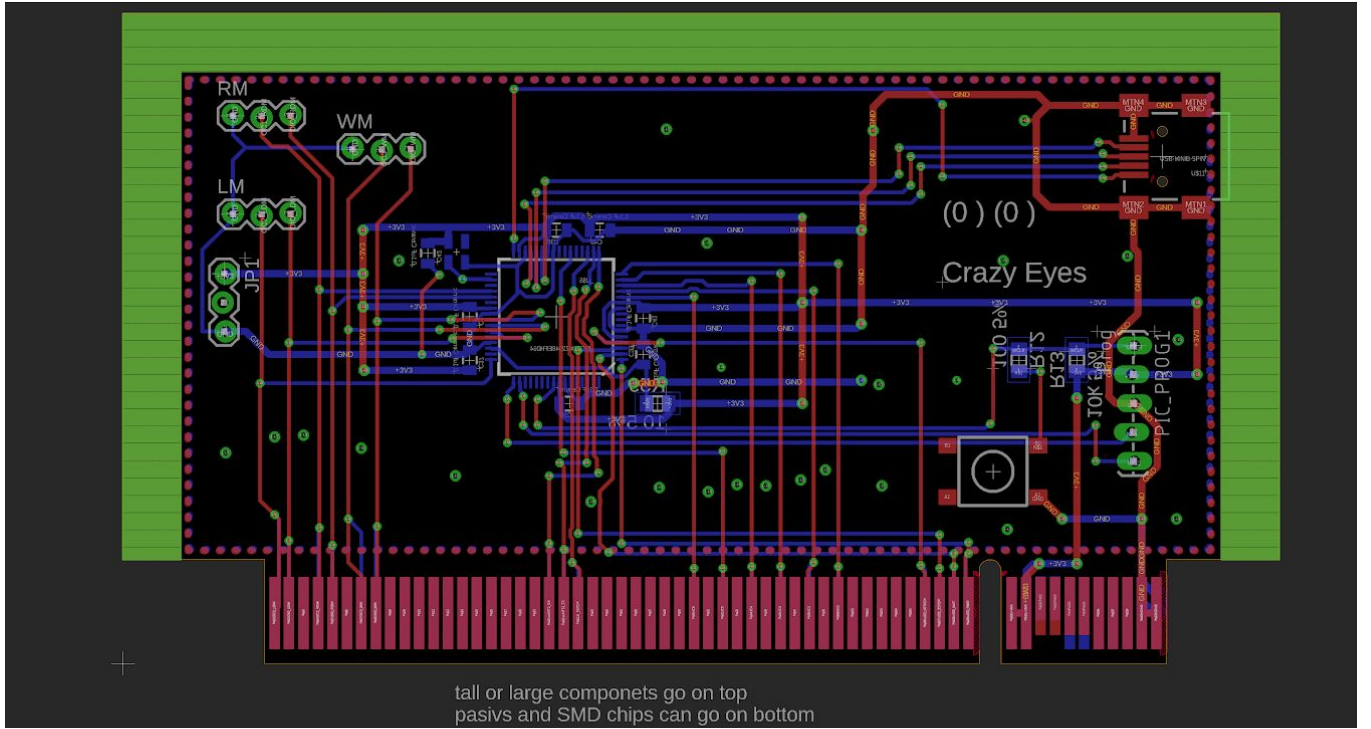
    // Enable interrupts
    IFS5CLR = _IFS5_U4RXIF_MASK;
    IPC42bits.U4RXIS = 1; // Priority (highest?)
    IPC42bits.U4RXIP = 0; // Sub priority?
    IEC5bits.U4RXIE = 1; // Enable IC1 interrupts
}

int rx4(){
    while(!U4STAbits.URXDA);
    return U4RXREG;
}
```

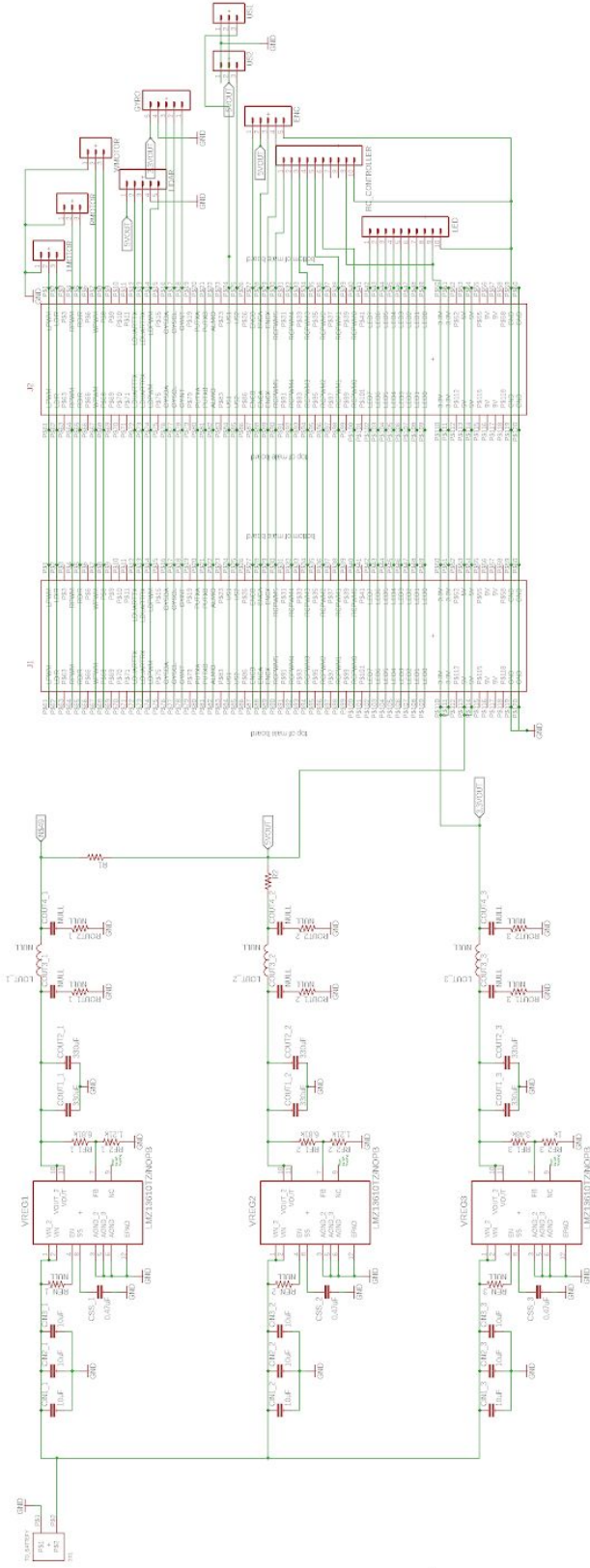
9.2 Board schematics

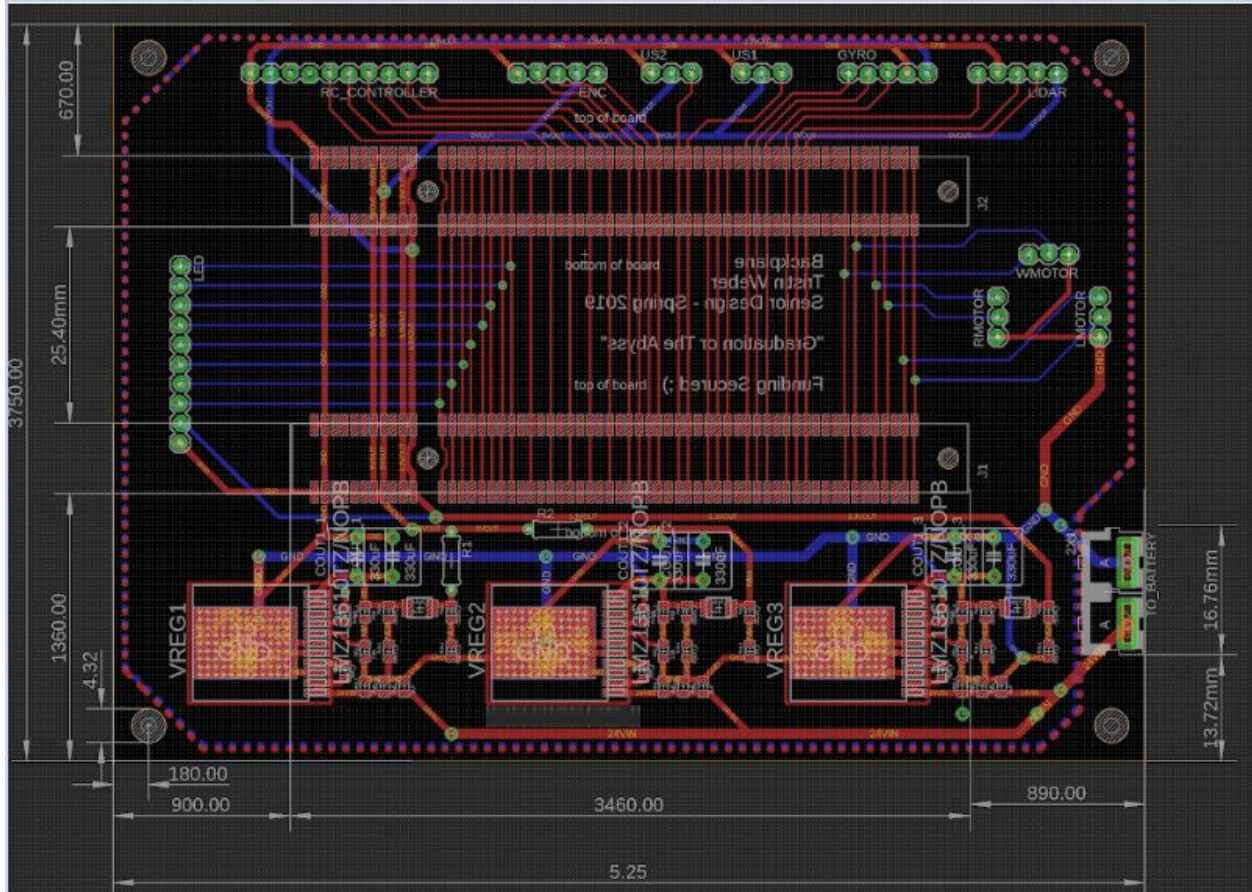
9.2.1 Control Board schematics





9.2.2 Power Board schematics





9.3 Useful links

Link for Harmony tutorials:

<http://microchipdeveloper.com/harmony:new-harmony-project-details>

Link for the PIC32MZ datasheet:

<http://ww1.microchip.com/downloads/en/DeviceDoc/60001320E.pdf>

Link for Drive Motor Controller Datasheet:

https://www.robotshop.com/media/files/images3/md30cusersmanual_1_.pdf

Link for Opto-Isolator Datasheet:

<https://www.mouser.com/datasheet/2/427/tcmt1100-103040.pdf>

9.4 Pictures

