SAMAR

SPARRING AUTONOMOUS MARTIAL ARTS ROBOT

PRODUCT DESIGN DOCUMENT

Version 1.0

09/19/2021

Author

James Garrett

Project Contributors

Brian Ah Koi

Daniel Campas

James Garrett

Ken Ramirez

Carlo Schumaker

TABLE OF CONTENTS

1 INTRODUCTION 3
1.1 Purpose of the Design Document 3
1.2 Problem Statement 3
2 DESIGN OVERVIEW 3
2.1 Synopsis 3
2.2 Constraints 3
2.3 Trade-Offs 4
2.3.1 Weight Capacity 4
2.3.2 Sensor Location 4
2.3.3 Data Collection 5
3 HARDWARE ARCHITECTURE 5
3.1 Mechanical Components 5
3.2 Electrical Components 6
3.3 User Interaction 7
3.4 Mechanical Drawbacks 7
3.5 Electrical Drawbacks 7
3.6 Future Improvements 8
4 SOFTWARE ARCHITECTURE 8
4.1 Sensor Analysis 8
4.2 Maneuver Calculations 10
4.3 Robot Repositioning 12
4.3.1 Obtaining Accurate RPM Values 12
4.3.2 Angular Direction Repositioning 13
4.3.3 Stopping Robot Movement 15
4.4 User Interaction 16
4.5 Drawbacks 16
4.6 Future Improvements 16

1 INTRODUCTION

1.1 PURPOSE OF THE DESIGN DOCUMENT

This design spec has been developed to elaborate on the hardware and software implementation choices for the Sparring Autonomous Martial Arts Robot (SAMAR) system. Readers will have an understanding of what specific actions can be performed by the machine, along with an in-depth look into why those actions are taken and how they are performed. The document also evaluates potential areas of improvement for future installments of the robot. Individuals interested in a comprehensive analysis of the SAMAR system are encouraged to read this document. Knowledge of algebra and trigonometry is recommended for Section 4.

1.2 PROBLEM STATEMENT

Sparring training is a critical aspect to the development of an individual's martial arts technique and is the most suitable method for simulating a potential self-defense encounter. However, because sparring is an exercise that requires another person, many do not have the ability to practice this skill outside of a class environment. SAMAR looks to resolve this issue, interacting with its user and surroundings in a real-time manner to best replicate a sparring experience.

2 DESIGN OVERVIEW

2.1 SYNOPSIS

SAMAR is an autonomous three-wheel drive system that supports the weight of a standing punching bag[1] and uses lidar tracking to sense its environment. Preset angular and distance ranges establish regions that, once entered by the user or other environmental obstacle, inform the robot to relocate. There are three distinct maneuvers that SAMAR can perform to reestablish its desired positioning: moving toward the user when they are too far, turning to the face the user when they are not centered, and moving away from the user or any obstruction when it is deemed too close. Auditory cues communicate the current state of the machine to the user. An onboard Raspberry Pi controls robot functionality and a 12-volt battery provides power to the system.

2.2 CONSTRAINTS

The current iteration of SAMAR is our first attempt at developing a solution for simulating martial arts sparring and is appropriately considered a prototype design. As such, materials used for this installment were to remain inexpensive. Most notably, the support frame of the machine is constructed entirely using wood. The tracking sensor used is the Slamtec RPLidar A1; there are two other versions of this sensor with better performance specs, but they are beyond the budget of this current design. Some electrical components are reused from past robotics endeavors and may be less known or obsolete compared with current standards. This includes the use of FIRST Robotics Competition (FRC) specific components.

[1] The punching bag being used is a Century Bob XL – Body Opponent Bag.

2.3 TRADE-OFFS

As this project continues development, conflicting issues arise that have to be addressed when designing both the hardware and software for this machine. The following trade-offs have been the most prevalent of these problems.

2.3.1 Weight Capacity

In order for the punching bag to sustain forceful contact from the user, its base can be filled with water or sand. This protects the robot frame from taking high impact and keeps the machine from being lifted off the ground. However, the extra weight means more wood beams will be needed as support, blocking the view of the sensor located beneath the bag. Thus, we make an effort to strategically place supports primarily in the rear of the robot as to keep the user in its sights more effectively. Approximately 13 pounds of water has been added to the 43-pound[2] punching bag with 4 supports in the back of the machine and 2 in the front. Figure 1 shows the base of the bag and the constructed wooden frame. Figure 2 shows the frame without the bag and better captures the 6 support beams and the sensor. Additional weight will affect the acceleration of the robot and could also require more supports to be added to the frame.

Figure 1

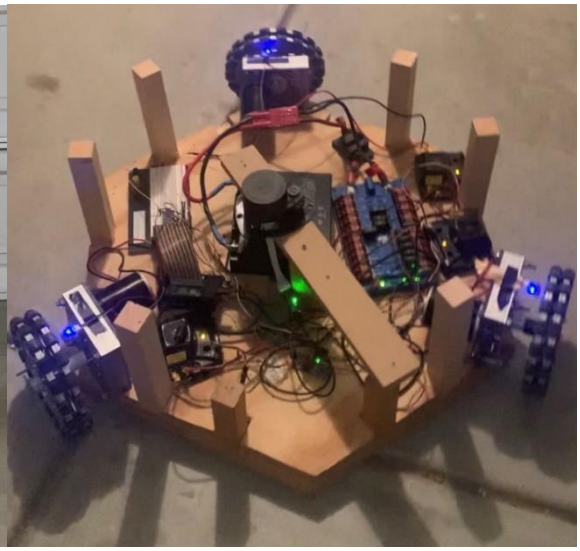


Figure 2

2.3.2 Sensor Location

Along with being centrally located within the robot perimeter, it is preferred that the sensor be placed as close to the ground as possible so it will see as many potential obstacles as it can. In order to prevent the sensor from being blocked by the wheels and their associated gearboxes – in addition to the support beams mentioned above – the sensor has been placed at a height of about 10 inches. This means that shorter obstacles, in particular the feet of the user, are not visible by the robot.

[2] This weight includes the base plus the bag, but note that the bag can be swapped with other Century products; the base by itself is 19 pounds.

2.3.3 Data Collection

The RPLidar can gather distance measurements for a larger number of angles the more times that it rotates. With the current software design, the sensor must cease recording data before the robot begins analyzing any values. Thus, with more rotations, SAMAR will react slower to its surroundings and may not move as quickly as expected by the user. A test was ran to see how we can keep the system working in real-time while also gathering enough data from the sensor. The results are shown below in the Figure 3 table. It has been decided that 4 sensor rotations is the best compromise for minimizing this issue.

RPLidar Scan Test (750 scans for each trial)		
Rotations/Scan	**Avg. # of Measured Distances/Scan**	**Avg. Time/Scan (s)**
1	52.53	0.3981
2	266.14	0.5228
3	345.65	0.6459
4	350.39	0.7702
5	351.37	0.8948
6	353.94	1.0212
7	354.50	1.1470

Figure 3

3 HARDWARE ARCHITECTURE

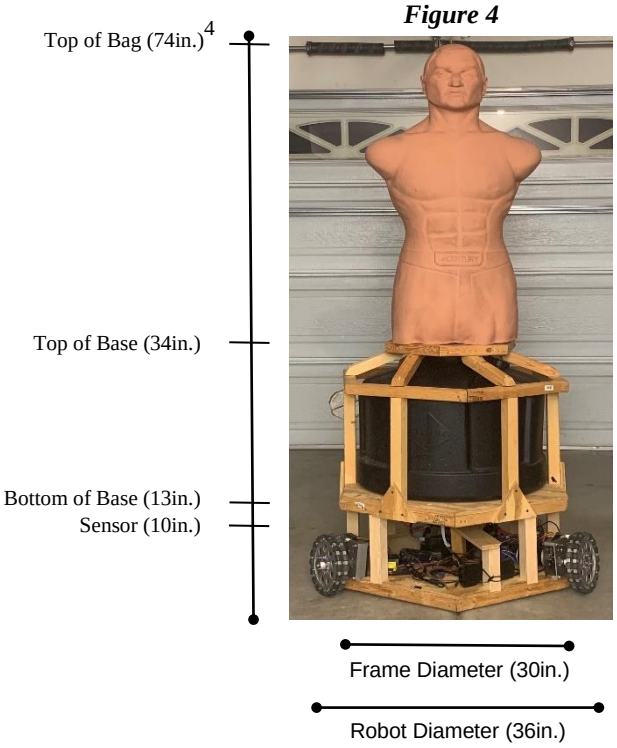
The specific physical construction of the robot itself is what we consider to be the hardware architecture. There are two main components that contribute to the tangible, hardware aspects of the SAMAR design: mechanical components, which are used to build the apparatus, and electrical components, which are powered devices that interact with software to control the entire machine.

3.1 MECHANICAL COMPONENTS

The frame, wheels, gearboxes, and punching bag are what make up the body of the robot. Figure 3 on the next page shows the machine in its entirety along with some of its key measurements. The frame is constructed using 1.5 x 1.5-inch wood blocks. These pieces are screwed together to form hexagonal layers. The top-most hexagon surrounds the narrow part of the punching bag to keep it from shifting due to robot maneuvering or from physical contact by the user. The middle and bottom layers are fitted with flat, wooden sheets. The middle layer carries the punching bag, while the bottom layer stores the electrical components. All three layers are separated and supported by additional 1.5 x 1.5-inch blocks. The outside of the bottom layer is where three pairs of 8-inch omnidirectional wheels[3] attached to AndyMark Toughbox gearboxes are evenly spaced around the frame. These wheels form the outer perimeter of the robot and allow SAMAR to move in all angular directions without the need to reorient itself. In addition, this setup also allows for easy rotations of the robot when applicable.

[3] Both wheels within each pair are connected and act as one wheel, and thus will only be referred to as a single wheel in the future to avoid confusion.

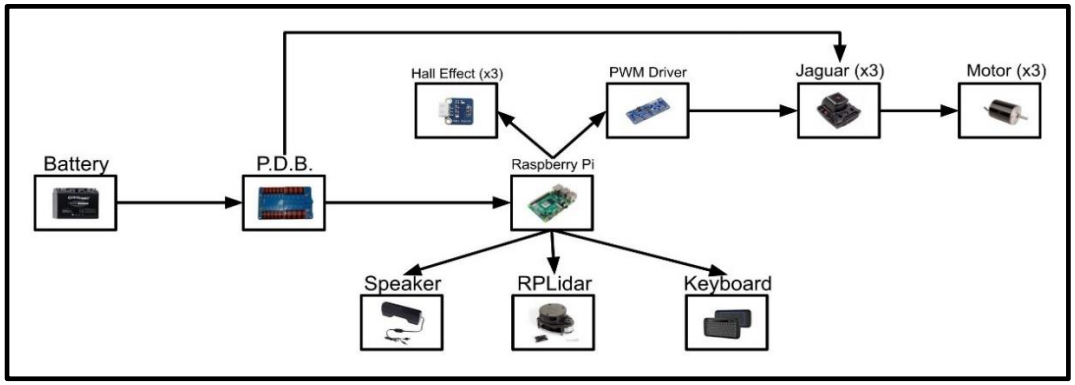
Figure 4



3.2 ELECTRICAL COMPONENTS

The robot's wheels are the only mechanisms it has that need to be controlled by the onboard electronics. The majority of the electrical components are used for this purpose, while some are used for enhancing user experience. A diagram showing the relationship between the electronics is pictured below in Figure 5. A rechargeable 12-volt battery gives power to the system using a FRC Power Distribution Board (PDB). Three Texas Instruments Jaguar motor controllers connect from the PDB to each of the three wheels. The Jaguars use pulse width modulation (PWM) to control specific electrical current to the wheels, which is needed in order for the robot to move in precise angular directions. Each wheel is powered by a Vex 12-volt. DC Mini CIM motor.

Figure 5



[4] The height of the bag is adjustable on the base and can increase in 6-inch increments up to an additional 18 inches.

The PDB also has a 5-volt connector that we use for powering a single Raspberry Pi 4.0. The remaining electrical devices are connected to this Pi. This includes an Adafruit 16-Channel 12-bit PWM/Servo Driver that is used to feed PWM values to the Jaguar motor controllers. To ensure that these values correlate to the desired revolutions per minute (RPM) for each wheel, we also installed SunFounder hall effect sensors. More details on how these sensors are used can be found in Section 4.3.1. The PWM Driver – which interfaces over I^2C – and the hall effect sensors are connected to the Raspberry Pi using its general-purpose input/output (GPIO) ports. All other electronics are connected via USB, including: the RPLidar sensor for distance measuring, an audio speaker for communicating with the user, and a mini wireless keyboard for ease of access in terminating and restarting the robot code. The lidar sensor has a USB to Serial adapter and thus uses serial communication.

3.3 USER INTERACTION

The user will need to have an understanding of how to provide power to the system, as well as how to charge the 12-volt battery that supplies this power. The user's interactions with other electrical components will be discussed in Section 4.4, as involvement with these devices is primarily influenced by software design. Our focus here turns to the robot's mechanical components. The user can perform light sparring on this machine without introducing any significant harm. However, full-strength hits or any contact that results in a wheel rising from the ground can cause damage over time. Due to the punching bag's distance off the ground, shorter individuals may find it difficult to perform leg techniques on the robot. It is recommended for all individuals to use protective footwear because the sensor cannot detect feet or any other object less than 10 inches off the ground, as discussed in Section 2.3.2. The machine doesn't move at a high rate of speed, so any incidental contact will not cause harm.

3.4 MECHANICAL DRAWBACKS

Many of the mechanical shortcomings for this machine have already been covered in Sections 2.2 and 2.3 where we discussed the constraints and trade-offs of its design. The previous section also reiterates some of these points. Additional drawbacks have come to light during testing as well. The robot will often drift when repositioning, changing the orientation of the sensor and thus when the robot stops maneuvering. The initial acceleration of the wheels can cause the frame to contact the floor, worsening the drifting problem. The weight of the machine is making the gearboxes slightly buckle, contributing to these issues as well.

3.5 ELECTRICAL DRAWBACKS

As covered in Sections 2.2 and 2.3, the lidar sensor has limitations and can be upgraded to newer versions. The Figure 3 table in Section 2.3.3 shows that the current sensor has about a 1/8-seconds increase in scan time for each additional rotation, with a minimum of almost 0.4 seconds for a single rotation. Testing has shown that the robot will not react as quickly as desired, taking away from the user's real-time experience. The vision sensing of the robot must be prioritized in later installments as it is the most crucial aspect to its performance.

3.6 FUTURE IMPROVEMENTS

In terms of mechanical components, adding springs to the support beams of the robot frame will let the machine absorb harder contact from the user. A steel or carbon fiber frame will allow for thinner and more robust supports. This could also mean requiring fewer support beams altogether, which blocks less of the sensor's view. Turning to the electronics, adding a gyroscope would assist with drifting issues while repositioning. Robot vision can also be improved just by using more sensors. Evenly spacing multiple sensors around the edge of the frame instead of a single, centrally-located one will prevent them from being blocked and instead permits them to be placed or angled lower to the ground. The addition of a camera will also allow for specific tracking of the user, rather than inferring based on the locations of objects within specific regions as is the current implementation.

4 SOFTWARE ARCHITECTURE

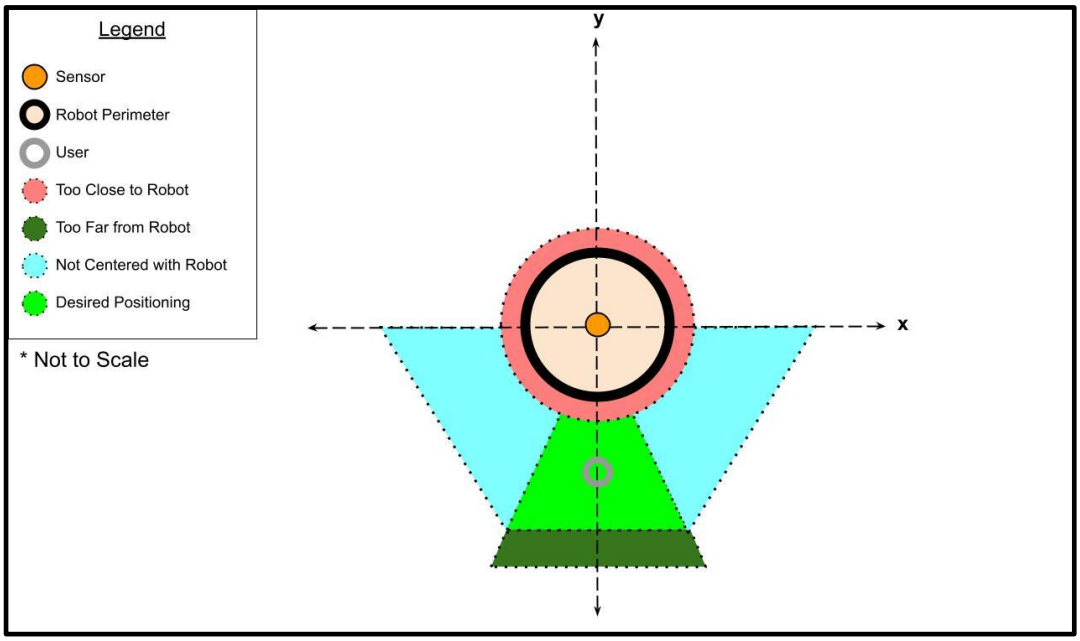
The onboard Raspberry Pi is in command of the software architecture for the SAMAR system. A single Python project, which is ran immediately on startup of the Pi, contains the code to control the robot and interact with its electrical components. This project is split into seven modules that contribute to three primary tasks for the robot: recording and analyzing sensor data to see if repositioning is appropriate, determining the most suitable method for maneuvering the robot if it is deemed necessary, and maneuvering the machine to its desired position when required.

4.1 SENSOR ANALYSIS

Data collection is performed indefinitely until a keyboard interrupt is inputted by the user on the mini wireless keyboard. Distance values are retrieved and stored by for all 360 degree-measured angles (converted from millimeters to inches). Values of 0 are those that could not be measured by the robot because they are greater than 236.22 inches, the max distance range of the sensor. This max range value is used for data storage purposes for angles that return a 0 value. Once sensor data is collected, the system analyzes all angles to see if and how the robot should reposition itself.

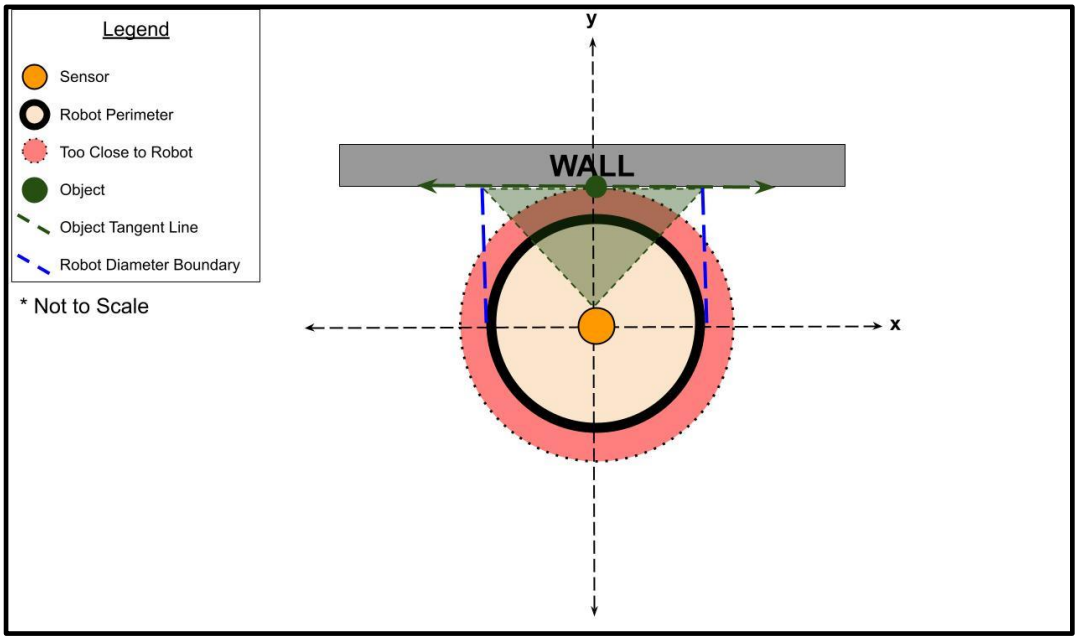
Figure 6 on the next page shows a diagram of how the program divides the robot's surroundings into distinct regions, giving a birds-eye view from the perspective of the user facing it. There are four unique regions split into five sections that let the robot know if and how to maneuver itself. The sensor gathers distance data following a standard Cartesian xy-coordinate system, where 0° is located on the positive x-axis. The preset angular and distance values that form these regions are declared in a separate module with all other global constants used throughout the project. Anything that is considered too close to the machine takes precedence and is maneuvered away from. Beyond that, if an object is detected within the desirable region, the robot will not move. Once the user is found to be too far or not centered, the robot will reposition accordingly. If more than one of the sections that form these regions detect an object, the section that detects an object most recently – implying that the user has just moved there – will have priority. These particular regions are not circular as to allow for lateral movement along the edge.

Figure 6



The Figure 7 diagram below shows how the program approaches objects when they are detected by the lidar sensor as too close at just one singular point. Flat objects – most notably walls – are likely to cause this issue to arise. To account for this, once an object is found to be too close at a given point, the robot will also look at additional points along the tangent line centered at this location. If an object is detected on or within this line, SAMAR will consider these angles to have distance values that are also too close to the robot, even if they are technically beyond the region. This tangent line will only be analyzed within the boundary formed by the diameter of the machine.

Figure 7



4.2 MANEUVER CALCULATIONS

Calculations are warranted only when objects are detected as too close to the machine. When the user is detected as off-center or too far away, the machine simply turns or moves directly forward until they are back within the desired region. However, when there are objects detected as too close, the program must determine the angle and distance to move at in order to create adequate space between the robot and said objects. The angle the robot will reposition at is based on the location of these objects. If only one object is detected as too close, the angle 180° opposite of this object will be the angle the machine maneuvers at. For two or more objects, the midpoint of the largest gap between consecutive angles will become the maneuver angle. If no gap between consecutive angles is larger than 90°, the robot is unable to move. A visualization is pictured below in Figure 8.

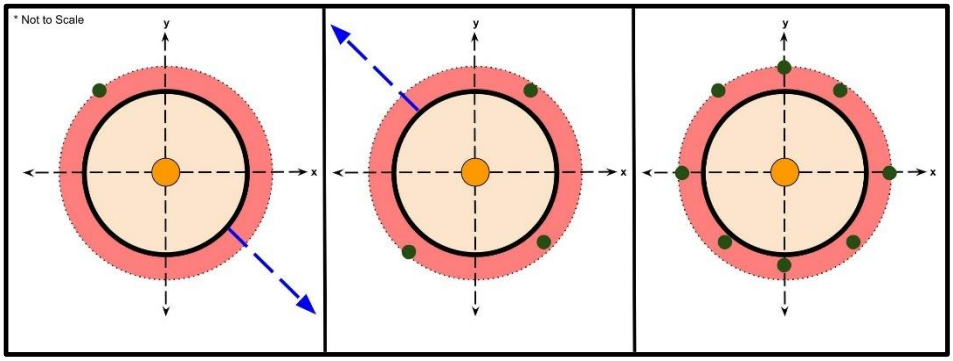
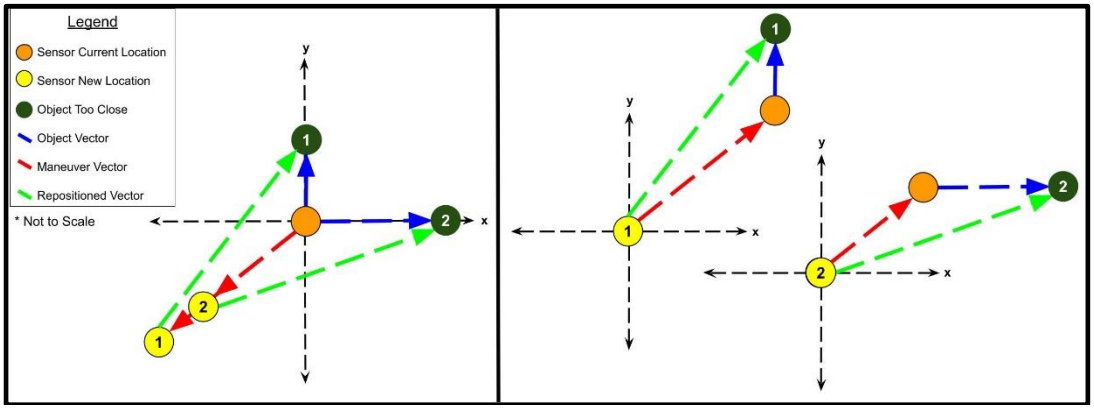


Figure 8

There is a predetermined value that is used as the desired distance that all objects must be from the robot once they are detected as too close. If this optimal distance value is not reachable because it would place the robot too close to what it's moving toward, then the maneuver angle and its distance value are now also considered too close to the robot and a new maneuver angle is found. Vector addition is used to determine how far SAMAR must travel in the direction of the maneuver angle so that it can be the desired distance away from all objects that are too close. The largest calculated distance that is needed to correctly reposition from a given object becomes the maneuver distance to be traversed. Figure 9 gives a visual of the vectors used for calculating the maneuver distance value.

Figure 9



The diagram on the left in Figure 9 shows three vectors for each object from the perspective of the robot before repositioning. The magnitude and direction of the object vector are already known: the direction is the angle of the object being read by the sensor, and the magnitude is the distance recorded at that angle. The direction of the maneuver vector is also known, equal to the aforementioned maneuver angle value. Its magnitude is the distance value we are looking to calculate for each object. In order to find this value using vector addition, we need the magnitude of the repositioned vector, which is equal to the predetermined desired distance value. Vectors can only be added when they are tip to tail, which means we must negate the maneuver vector, changing its direction but not its magnitude. This can be seen on the right-side diagram of Figure 10. This process is repeated for each object and the greatest distance calculated will become the maneuver distance. Proof 1 shows how to calculate this distance for each object.

Proof 1

Let $\boldsymbol{R}$ be the repositioned vector for a given object, where $|\boldsymbol{R}|$ is the desired distance value.

Let $\boldsymbol{O}$ be the object vector, where $\angle\boldsymbol{O}$ is the sensor angle degree value for said object and $|\boldsymbol{O}|$ is the distance value read by the sensor at this angle.

Let $\boldsymbol{M}$ be the opposite to the maneuver vector, where $\angle\boldsymbol{M}$ is the angle 180° offset from the precalculated maneuver angle value We look to show that we can solve for $|\boldsymbol{M}|$.

$$|\boldsymbol{R}| = \sqrt{\boldsymbol{R_x^2} + \boldsymbol{R_y^2}} \qquad \textbf{by the vector magnitude formula}$$

$$|\boldsymbol{R}| = \sqrt{(\boldsymbol{O_x} + \boldsymbol{M_x})^2 + \left(\boldsymbol{O_y} + \boldsymbol{M_y}\right)^2} \qquad \textbf{by vector addition}$$

$$|\boldsymbol{R}| = \sqrt{\begin{matrix}(\cos(\angle\boldsymbol{O}) * |\boldsymbol{O}| + cos(\angle\boldsymbol{M}) * |\boldsymbol{M}|)^2 + \\ (\sin(\angle\boldsymbol{O}) * |\boldsymbol{O}| + sin(\angle\boldsymbol{M}) * |\boldsymbol{M}|)^2\end{matrix}} \qquad \textbf{by properties of trigonometry}$$

Let $\mathbf{V_1}$, $\mathbf{V_2}$, $\mathbf{V_3}$, and $\mathbf{V_4}$ represent the calculatable values based on what has already been given, where:

$$\mathbf{V_1} = \cos(\angle\boldsymbol{O}) * |\boldsymbol{O}|$$
$$\mathbf{V_2} = \cos(\angle\boldsymbol{M})$$
$$\mathbf{V_3} = \sin(\angle\boldsymbol{O}) * |\boldsymbol{O}|$$
$$\mathbf{V_4} = \sin(\angle\boldsymbol{M})$$

Therefore,

$$|\boldsymbol{R}| = \sqrt{(\mathbf{V_1} + \mathbf{V_2} * |\boldsymbol{M}|)^2 + (\mathbf{V_3} + \mathbf{V_4} * |\boldsymbol{M}|)^2} \qquad \textbf{by substitution}$$

$$|\boldsymbol{R}| = \sqrt{\mathrm{V}_1^2 + (V_2 * |\boldsymbol{M}|)^2 + 2 * |\boldsymbol{M}|\boldsymbol{V_1}\boldsymbol{V_2} + \mathrm{V}_3^2 + (V_4 * |\boldsymbol{M}|)^2 + 2 * |\boldsymbol{M}|\boldsymbol{V_3}\boldsymbol{V_4}} \qquad \textbf{by FOIL}$$

$|\boldsymbol{R}|^2 = (\mathbf{V_2^2} * |\boldsymbol{M}|^2 + (2\mathbf{V_1}\mathbf{V_2}) * |\boldsymbol{M}| + \mathbf{V_4^2} * |\boldsymbol{M}|^2 + (2\mathbf{V_3}\mathbf{V_4}) * |\boldsymbol{M}| + \mathbf{V_1^2} + \mathbf{V_3^2}$ **simplified**

$0 = (\mathbf{V_2^2} + \mathbf{V_4^2})|\boldsymbol{M}|^2 + (2\mathbf{V_1}\mathbf{V_2} + 2\mathbf{V_3}\mathbf{V_4}) * |\boldsymbol{M}| + (-|\boldsymbol{R}|^2 + \mathbf{V_1^2} + \boldsymbol{V_3^2})$ **simplified**

Let *a*, b, and c be the coefficients for the quadratic formula, where:

$a = \mathbf{V_2^2} + \mathbf{V_4^2} = 1$ **by trigonometric identities**
$b = 2\mathbf{V_1}\mathbf{V_2} + 2\mathbf{V_3}\mathbf{V_4}$
$c = -|\boldsymbol{R}|^2 + \mathbf{V_1^2} + \mathbf{V_3^2}$

Therefore,

$0 = a * |\boldsymbol{M}|^2 + b * |\boldsymbol{M}| + c$

$|\boldsymbol{M}| = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ **by quadratic formula**

$|\boldsymbol{M}| = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$ **by substitution**

Using back substitution for a, b, c, and subsequently $\mathbf{V_1}$, $\mathbf{V_2}$, $\mathbf{V_3}$, and $\mathbf{V_4}$, we can solve this equation for $|\boldsymbol{M}|$. It must be equal to the positive outcome as magnitude cannot be negative.

QED

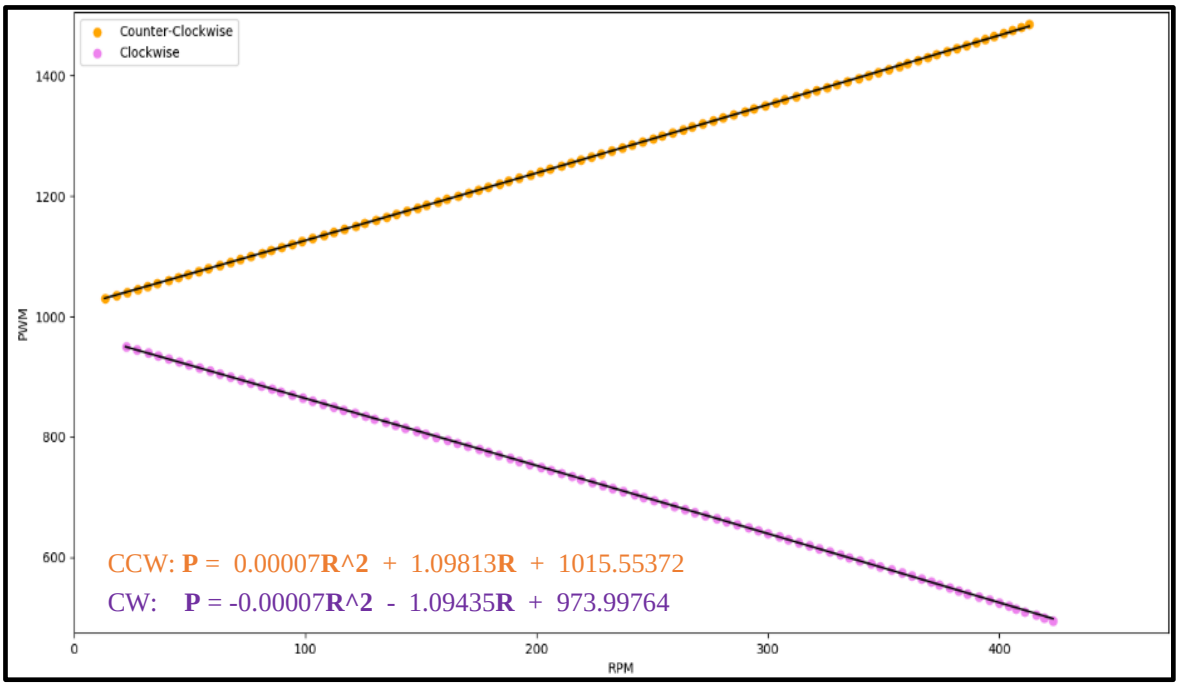
4.3 ROBOT REPOSITIONING

As discussed in Section 2.1, SAMAR can reposition itself in three different ways: turning in place to face the user when they aren't centered, moving forward to the user when they are too far away, and moving away from the user or any obstruction considered too close. Figure 2 in Section 2.4 shows where the three omni-wheels are placed on the robot from the perspective of the user. They are evenly spaced around the frame, located at 90°, 210°, and 330°, using the standard Cartesian xy-coordinate system described in Section 4.1.

4.3.1 Obtaining Accurate RPM Values

Due to factors such as wear and tear on the motors, it is possible for identical PWM values for two or more wheels to result in nonequal RPM values. To ensure that RPM values are accurate across all wheels, we implement the installed hall effect sensors. They are currently only used for testing purposes in which we create a function of RPM to PWM. The test records RPM values as it slowly increases the PWM of a free-spinning wheel. We can then plot the data recorded and estimate a function of RPM to PWM using those points. This process is done for all three wheels going both clockwise and counterclockwise. With this testing, when a specified RPM value is given, the program module for repositioning can now accurately find the correct PWM input that will yield that RPM value. Figure 10 on the next page shows the graph for one of these wheels.

Figure 10



4.3.2 Angular Direction Repositioning

When turning the machine in place, the speeds of all three wheels are simply set to the same predetermined RPM value and are also set to rotate in the same orientation. However, the RPM values will vary for each wheel when the machine is moving at a particular angle to or from an object or individual. We use vectors once again to determine what percent of a predefined maximum RPM value must be used for each wheel in order to move in the required direction. Figure 10 above shows that as RPMs increase, PWM values will also increase if the wheel is spinning counterclockwise, while they decrease when the wheel is rotating clockwise.[5] To keep this continuity, each wheel movement vector will have its direction component associated with the angle that it spins at when rotating counterclockwise. If the calculated magnitude for a given wheel is positive, the direction remains the same, however if negative, the wheel must spin clockwise and the value is negated as magnitude must always be positive. Based on the wheel locations around the frame of 90°, 210°, and 330°, when the wheels are rotating counterclockwise, they are moving at the angles of 0°, 120°, and 240° respectively. The magnitude of each vector represents the decimal percentage of the max RPM value to be applied to that wheel. Thus, the magnitude of the resultant vector will be equal to 1. In order to guarantee that no singular value is greater than 1 or less than -1, the sum of the wheel vector magnitudes before taking their absolute value must be equal to 0. Proof 2 on the next page will demonstrate how to calculate the RPM value for each wheel given a particular angle to maneuver in the direction of.

[5] These rotation directions are determined by how each individual wheel is spinning from the perspective of an onlooker directly facing the wheel.

Proof 2

Let $\boldsymbol{M}$ be the resultant, maneuver vector, a unit vector where $\angle\boldsymbol{M}$ is the precalculated maneuver angle value and $|\boldsymbol{M}| = 1$ represents the percentage of the vectors being summed.

Let $\boldsymbol{W_1}$, $\boldsymbol{W_2}$, and $\boldsymbol{W_3}$ be the wheel vectors, where $\angle\boldsymbol{W_1}$, $\angle\boldsymbol{W_2}$, and $\angle\boldsymbol{W_3}$ are the angles that each wheel rotates in the direction of when spinning counterclockwise: 0°, 120°, and 240° respectively. We look to prove that we can solve for $|\boldsymbol{W_1}|$, $|\boldsymbol{W_2}|$, and $|\boldsymbol{W_3}|$.

Let $|\boldsymbol{W_1}| + |\boldsymbol{W_2}| + |\boldsymbol{W_3}| = 0$.

$\boldsymbol{M_x} = \boldsymbol{W_{1x}} + \boldsymbol{W_{2x}} + \boldsymbol{W_{3x}}$ **by vector addition**

$\cos(\angle\boldsymbol{M}) * |\boldsymbol{M}| = \cos(\angle\boldsymbol{W_1}) * |\boldsymbol{W_1}| + \cos(\angle \mathrm{W}_2) * |\boldsymbol{W_2}| + cos(\angle \mathrm{W}_3) * |\boldsymbol{W_3}|$ **by properties of trigonometry**

$\cos(\angle\boldsymbol{M}) = 1|\boldsymbol{W_1}| - \frac{\boldsymbol{1}}{\boldsymbol{2}}|\boldsymbol{W_2}| - \frac{\boldsymbol{1}}{\boldsymbol{2}}|\boldsymbol{W_3}|$ **by substitution**

$\boldsymbol{M_y} = \boldsymbol{W_{1y}} + \boldsymbol{W_{2y}} + \boldsymbol{W_{3y}}$ **by vector addition**

$\sin(\angle\boldsymbol{M}) * |\boldsymbol{M}| = \sin(\angle\boldsymbol{W_1}) * |\boldsymbol{W_1}| + \sin(\angle \mathrm{W}_2) * |\boldsymbol{W_2}| + sin(\angle \mathrm{W}_3) * |\boldsymbol{W_3}|$ **by properties of trigonometry**

$\sin(\angle\boldsymbol{M}) = 0|\boldsymbol{W_1}| + \frac{\sqrt{\boldsymbol{3}}}{\boldsymbol{2}}|\boldsymbol{W_2}| - \frac{\sqrt{\boldsymbol{3}}}{\boldsymbol{2}}|\boldsymbol{W_3}|$ **by substitution**

We now have a system of linear equations represented with this augmented matrix:

$$\left[\begin{array}{ccc|c} \boldsymbol{1} & -\frac{\boldsymbol{1}}{\boldsymbol{2}} & -\frac{\boldsymbol{1}}{\boldsymbol{2}} & \cos(\angle\boldsymbol{M}) \\ \boldsymbol{0} & \frac{\sqrt{\boldsymbol{3}}}{\boldsymbol{2}} & -\frac{\sqrt{\boldsymbol{3}}}{\boldsymbol{2}} & \sin(\angle\boldsymbol{M}) \\ \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{0} \end{array}\right]$$

We now have a system of linear equations represented with this augmented matrix:

$$\left[\begin{array}{ccc|c} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \frac{2 * \cos(\angle\boldsymbol{M})}{3} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & -\frac{\cos(\angle\boldsymbol{M}) - \sqrt{3} * sin(\angle\boldsymbol{M})}{3} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & -\frac{\cos(\angle\boldsymbol{M}) + \sqrt{3} * sin(\angle\boldsymbol{M})}{3} \end{array}\right]$$

Thus $\boldsymbol{W_1} = \frac{2*\cos(\angle\boldsymbol{M})}{3}$, $\boldsymbol{W_2} = -\frac{\cos(\angle\boldsymbol{M}) - \sqrt{3}*sin(\angle\boldsymbol{M})}{3}$, and $\boldsymbol{W_3} = -\frac{\cos(\angle\boldsymbol{M}) + \sqrt{3}*sin(\angle\boldsymbol{M})}{3}$.

QED

4.3.3 Stopping Robot Movement

Regardless of the manner in which it is repositioning, once the robot begins maneuvering, there are just two situations in which it will stop: it has detected an object as too close in the direction it is traveling or it has reached its new, expected location. For the first scenario, when turning, a sensor reading in any direction that is considered too close to the machine will stop its rotation. A breakdown of the different sensor regions is described in Section 4.1. When moving closer or further from the user or other object, the robot is looking out for obstructions within a range of 180 degrees centered at the angle that it's currently traveling in the direction of. In the case of moving to the user, this central angle is 270°; when moving away from an object, this angle is the calculated maneuver angle value described in Section 4.2.

When turning or when moving forward, the robot has reached its target location when it can again detect the user within the desired region. When an object is too close, the robot has reached its target when the object it is moving from is now the desired distance away. However, as can be seen in Figure 9 of Section 4.2, the angle of the object may change after repositioning. We can again use vector addition to find what the angle of this object will be; Proof 3 below will elaborate on this. For all three methods of maneuvering, a range of angles are used as a failsafe to account for drifting or missed sensor readings when attempting to stop maneuvering at a specific angle.

Proof 3

Let $\boldsymbol{O}$ be the object vector, where $\angle\boldsymbol{O}$ is the sensor angle degree value for said object and $|\boldsymbol{O}|$ is the distance value read by the sensor at this angle.

Let $\boldsymbol{M}$ be the opposite to the maneuver vector, where $\angle\boldsymbol{M}$ is the angle 180° offset from the precalculated maneuver angle value and $|\boldsymbol{M}|$ is the calculated maneuver distance value after using the equation from Proof 1 on each object.

Let $\boldsymbol{R}$ be the repositioned vector for the object that determined the maneuver distance value, where $|\boldsymbol{R}|$ is the desired distance value. We look to show that we can solve for $\angle\boldsymbol{R}$.

$$\boldsymbol{R_x} = \boldsymbol{O_x} + \boldsymbol{M_x} \qquad \textbf{by vector addition}$$

$$\boldsymbol{R_x} = \cos(\angle\boldsymbol{O}) * |\boldsymbol{O}| + cos(\angle\boldsymbol{M}) * |\boldsymbol{M}| \qquad \textbf{by properties of trigonometry}$$

$$\textbf{cos}(\angle\boldsymbol{R}) = \frac{\boldsymbol{R_x}}{|\boldsymbol{R}|} \qquad \textbf{by properties of trigonometry}$$

$$\angle\boldsymbol{R} = \text{acos}\,(\frac{\boldsymbol{R_x}}{|\boldsymbol{R}|}) \qquad \textbf{simplified}$$

Arccosine has a range limited to the first two quadrants, thus we must also solve for $\boldsymbol{R_y}$:

$$\boldsymbol{R_y} = \boldsymbol{O_y} + \boldsymbol{M_y} \qquad \textbf{by vector addition}$$

$\boldsymbol{R_y} = \sin(\angle \boldsymbol{O}) * |\boldsymbol{O}| + sin(\angle \boldsymbol{M}) * |\boldsymbol{M}|$ **by properties of trigonometry**

We may use the signs of $\boldsymbol{R_x}$ and $\boldsymbol{R_y}$ together in order to find the quadrant of $\boldsymbol{\angle R}$:

If $\boldsymbol{R_x}$ and $\boldsymbol{R_y}$ are both negative, $\boldsymbol{\angle R}$ is in the third quadrant: $\angle \boldsymbol{R} = 360 - \text{acos}\left(\frac{\boldsymbol{R_x}}{|\boldsymbol{R}|}\right))$

If $\boldsymbol{R_x}$ is positive and $\boldsymbol{R_y}$ is negative, $\boldsymbol{\angle R}$ is in the fourth quadrant: $\angle \boldsymbol{R} = 360 - \text{acos}\left(\frac{\boldsymbol{R_x}}{|\boldsymbol{R}|}\right)$

Therefore $\angle \boldsymbol{R}$ is solvable within all quadrants with the given values.

QED

4.4 USER INTERACTION

Once the electronics are powered, the Raspberry Pi will immediately boot up and start the sensor analysis program. An audio alert informs the user when this occurs. The robot will be in a standby state until the user interacts with the handheld keyboard. Pressing <ENTER> will allow robot sensing and repositioning to begin, sounding another audio cue. Currently there are four audio signals in total that the user will need to be made aware of. A basic understanding of Python keyboard interrupts for stopping the running code, as well as knowledge of Linux commands for properly shutting off the Pi when not in use, is also needed.

4.5 DRAWBACKS

The previous section brings attention to the fact that the user is expected to have some software knowledge in order to use the machine properly, which is not ideal if the product is mass produced. The user also has no flexibility in terms of different modes or default settings to choose from, such as how far or close they can be to the robot before it moves. This could be an issue as individuals' sparring strategies vary.

4.6 FUTURE IMPROVEMENTS

Adding multithreading to our project so that modules can run synchronously should improve runtime and allow the robot to respond quicker to its surroundings. This is especially needed if multiple sensors are being used for distance tracking. Learning and implementing OpenCV will improve accuracy of user tracking if a camera is incorporated into the SAMAR system. The user experience drawbacks discussed in the last section will be addressed, though this may not be a priority until a final product is ready to be released.