2021

Simplification of Robotics Through Autonomous Navigation

Grant Turner
Western Kentucky University, grant.turner184@topper.wku.edu

SIMPLIFICATION OF ROBOTICS THROUGH AUTONOMOUS NAVIGATION

A Capstone Experience/Thesis Project Presented in Partial Fulfillment

of the Requirements for the Degree Bachelor of Science

with Mahurin Honors College Graduate Distinction

at Western Kentucky University

By

Grant Turner

May 2021

*****

CE/T Committee:

Dr. Mark Cambron, Chair

Dr. Robert Choate

Prof. Joel Lenoir

ABSTRACT

With self-driving vehicles, college campus food delivery, or even automated home vacuuming systems, robotics is undoubtedly becoming more prevelant in everyday society and it can be expected to continue with time. While many people are owners, users, or even just spectators of theses robotic products or services, there seems to be a negative perception of robotics that poses an intimidation factor regarding the attempt to understand the ideas driving technology. This perception tends to view robotics as machines that require rich education to understand the complexity and interworkings of, thus attempts understand the field are neglected.

To combat this line of thinking, I have set out to break down concepts of robotics to satisfy the basic understanding of an individual from an untrained background. To do this, I have developed a lesson plan that teaches fundamental principles behind robotics and I have developed a beginner-level autonomous navigation project that participants can do to prove their newfound understanding. From the lesson plan aspect, participants are introduced to electronics, mechanical design, and various programming techniques. When the participant attempts the autonomous navigation project, the individual interacts with a pre-built robot and the focus is on developing ideas of how to program the robot to navigate a unidirectional hallway system in which the robot is able to autonomously travel through system of irregular turns. Participants actively test their understanding through application of their programming ideas to the robot.

The inspiration for this project stems from my personal experience with secondary education and my experience as I transitioned into further education, but more

specifically, the lack of direction individuals similar to me had through these experiences. I come from a part of my city that is known for having a much smaller base of financial resources and is also often perceived to be lesser in terms of educational quality. While my place of secondary education partnered with the local technical school to provide an opportunity to take a robotics course, few were able to take advantage of this opportunity. Other than this singular opportunity off campus, there were few other known opportunities within our school to help individuals find interests in STEM based fields and few opportunities that pointed in alternative directions to STEM fields. Unless one had a relative in a field, most individuals were left directionless as to what they may want to do as a future career or what they may want to study if college was an option.

As an individual who attended college as a guess as to what to do next with life, selecting a major was also a blind throw at a dart board. The decision on my major was between creative writing and engineering, two very different subjects and if it were not for the simple ideas that I had already advanced through a couple of the beginning engineering math courses and my liking of the idea of "building things," I would have chosen writing. Even so, the handful of my high school class graduates who also chose engineering had little idea of any differentiation between the disciplines and still barely knew what engineering as a whole was, thus we all chose different disciplines and hoped for the best. Simplication of Robotics Through Autonomous Navigation was created to give learning opportunities to individuals who lack such opportunity and have interest in fields related to robotics, yet may also lack comfort to associate with the field.

ACKNOWLEDGEMENTS

I would like to thank all of the individuals who worked with me along the way of developing the autonomous navigation project. Thank you to the Mahurin Honors College donor who sponsored my project through an Honors Development Grant, making this project possible. Thank you to Dr. Mark Cambron, Dr. Robert Choate, and Prof. Joel Lenoir for serving as my advisors and readers on the project. Thank you to Ron Rizzo for assistance in development of my circuit boards and advising throughout the project. Finally, thank you to my family for the various instances of support whether it be assisting in assembling a robot or overhearing crash course curriculum.

VITA

EDUCATION

Western Kentucky University, Bowling Green, KY — May 2021
B.S. in Electrical Engineering – Mahurin Honors College Graduate
Honors CE/T: *Simplification of Robotics through Autonomous Navigation*

Warren East High School, Bowling Green, KY — May 2017

PROFESSIONAL EXPERIENCE

Engineering, HeathCo LLC — May 2019-
Intern — Dec. 2020

Engineering, Sumitomo Electric Wiring Systems — Apr. 2018-
Intern — May 2019

AWARDS & HONORS

Summa Cum Laude, WKU, May 2021

PROFESSIONAL MEMBERSHIPS

Tau Beta Pi – The Engineering Society (TBP)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION ONE

Background

Tell someone you study engineering and you will receive a slight chuckle followed with, “Sounds too tough for me.” Show someone a project you are working on and in a perplexed tone you will hear, “Now that’s just over my head!” Individually, I have completed only a handful of projects based around electronics and robotics, but each time I show someone outside of my field, I receive a response similar to these. Although I realize that as an individual studying electrical engineering, I am more familiar with components and robotic processes and I also may find my work to be quite simple relative to what I have seen, I can not help to be curious of what causes the untrained individual to have the perception that even the most basic understanding of how a robot functions is unrealistic for said untrained individual. Do individuals need a four-year education to begin to generate ideas of what could possibly cause their Roomba to adjust when it hits an object or make an educated guess to why a Starship food delivery robot always stops or moves around surrounding people or objects? If people had basic surface knowledge of different types of sensors or electronics, could they then formulate a hypothesis on why the robots operate as they do? Does the perception of robotics immediately drift to a form of complexity because society often only sees robots as final products and rather than machines functioning step by step?

While I believe formal education has its place in cultivating minds in the process of creating robotics, I also believe that even individuals without formal education are often capable of figuring out fundamental functions of how some robots operate. By

providing surface level knowledge of various necessary segments of robotics and an opportunity to apply ideas, I aim to cultivate minds and show that when learning to break down the subject into manageable portions, almost anyone can understand general concepts of robotics that overarch a system. By doing this, I hope to influence and alter the perspective believing robotics is only for trained minds and also enourage participation in STEM fields related to robotics.

SECTION TWO

Curriculum Plan

Since the target is to tackle key understanding or fundamental ideas, curriculum should cover enough background knowledge on a surface level to allow participants to get by, but the focal point of the project is to give the opportunity for the participants to apply their concept ideas to show their understanding. The best way of doing this is by allowing participants to operate their own programmable device and directly see how their ideas in code carries out to the functionality of the device. How this is done exactly will be discussed in detail later. First, the talked about superficial knowledge given through a robotics crash course will be explained.

Robotics Crash Course

Earlier it was noted that a possible disconnect of individuals and basic understanding may occur from an unfamiliarity or inability to recognize electronic components and/or principles. While it is not possible to force years of understanding and practice into the brains of another, the crash course is designed to brush many topics on a very high level and plant seeds of ideas to hopefully provide a familiarity or spark ideas later in the session when it comes time to develop the autonomous navigation project.

At this point it would be best to understand the overarching idea of what robotics even is. From a broad perspective concerning disciplines, robotics is the interdisciplinary field that integrates computer science and engineering (German National Library). I prefer to break it down into three main categories: skeleton design, electrical design, and

programming. The three portions may be a bit simpler to understand if thought of in terms of a human body.

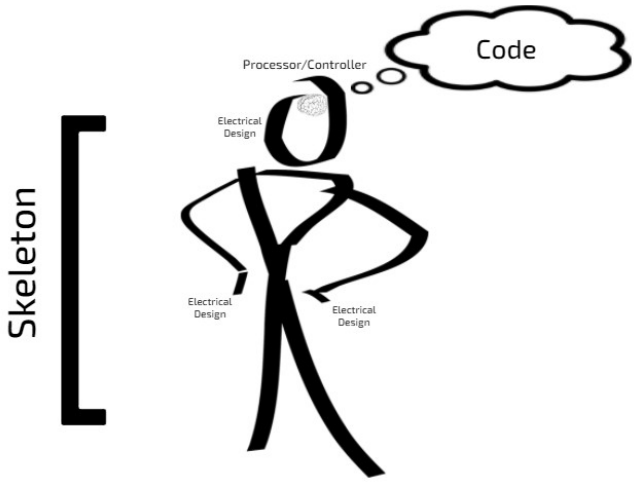

Figure 1. **Relationship of Components of Robotics to the Human Body**

The skeleton of a robot consists of internal and external framework designed to hold everything together. This part often accounts for all additional components of the system and joints them together to hold a certain shape or move as needed. When a skeleton begins to move or change shape to meet needs of the robot, the principle begins to explore a branch of mechanical design, which refers to how energy or forces affect a system (Merriam-Webster, 2001). Relating back to the human body, the skeleton serves as people typically know it to, framework that connects and holds together multiple body parts. Below in Figure 2 is a portion of the skeleton related to the autonomous navigation robot that will be used for the follow-up project. It contains simple brackets used to secure down servos and sensors and there are also many holes in the skeleton that are used to bolt on additional parts of the robot. In addition to the brackets and holes, two pipes and a basket can be seen in Figure 3 which are used to hide or organize wires from being tampered with.

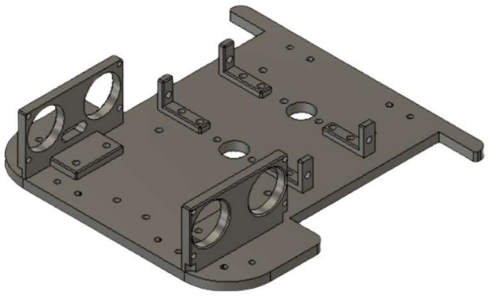

Figure 2. Partial Skeleton of Autonomous Navigation Robot

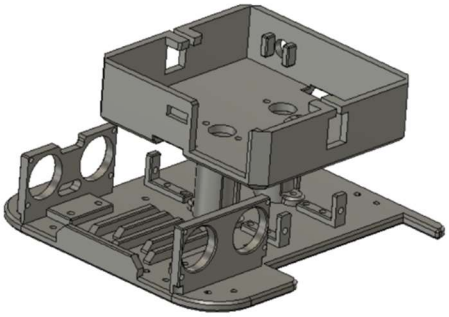

Figure 3. Complete Skeleton of Autonomous Navigation Robot

Following the skeleton comes the electrical or electronic design. In the most fundamental terms, electrical design can refer to anything that completes a circuit to allow electricity to flow. Regarding electronic design and robotics and in relation to the human body, electronics often relate to information receiving or sending in terms of the general five human senses: taste, touch, smell, sight, and hearing.

Most electronic components can be classified as either an input device (it receives information using senses) or an output device (it activates senses through information sent out), often grouped altogether and referred to as I/O components. Examples discussed in the crash course include ultrasonic sensors, infared (IR) sensors, LEDs, piezo buzzers, limit switches, and joysticks. Ultrasonic sensors and IR sensors are two components focused on in the curriculum, both of which serve as input devices which

relate to the human body analogy by being comparable to eyes and sight. The difference between the two is that the ultrasonic sensors can tell the user how far away an object is, but an IR can only give the user a yes/no answer of if an object is physically present within a given range. Another electrical component that was focused on was the contact switch or limit switch. By allowing or preventing electricity to flow when a button is pushed, the component can be thought of as satisfying the human sense of touch and can be used to directly feel if something if something is pushing against it. As far as talk about output devices, the curriculum uses LEDs as the main component in later programming examples. LEDs are used as visible indicators for participants to learn how programs work and monitor their understanding of sample programs.

Although the skeleton and electronic components of a robot can make it appear quite fancy, without an uploaded program, the robot will remain lifeless, or at least inanimate. Robots need some sort of brain with thoughts or set of instructions to actually tell it what to do. Microcontrollers are generally the electronic components used to serve as the brain of a robot. Technically defined, Robert Kein defines a microcontroller as an "integrated circuit (IC) device used for controlling other portions of an electronic system, usually via a microprocessor unit (MPU), memory, and some peripherals," but it should be understood that even though a microcontroller has the power to control all I/Os of a robot, without proper instructions though uploaded code, the controller does nothing (Keim, 2019). This concept suggests that for a robot to function, the programming portion must contain both a controller and a program code.

If it is still unclear of what code is exactly, it may be easier to think of it as a recipe from a cookbook. The processor or controller follows each instruction to a T and

starts with step one and only proceeds to the next step when the instruction allow it to. The speed in which each step or instruction is executed is almost instantaneous unless otherwise specified. One must remember that the device that reads the code often does not interpret to do what it thinks the user wants it to do, but rather does exactly what it is told to do. This is why it is important that code is written in a clear and methodical way. To give an example to show how clear a code needs to be, a simple recipe for a fried egg instructs the user to place an egg into a skillet on the burner. If the instructions do not deliberately instruct the cook to crack the egg open and place only the yolk and egg whites in the skillet, the new cook may end up placing a full unbroken egg into the skillet only for the shell to get charred. While this is just an example of the nature of coding, there is still much more to know before turning ideas into a program code.

Robot code is much like common world languages, there are a variety of options, but each one has its own time and place for when it may be best to use it. Similar to how world languages have their own perception of complexity, coding languages are also often perceived with their own idea of what is complex or simple. For example, Python is generally considered an easier coding language to learn than C or C++. While some programmers may prefer one language over another, depending on what hardware or type of controller used, the options of language choice may be limited. For the robots developed for the later project, the code language is locked in with the developed Arduino language, which is a variant of C++.

Returning to the main purpose of the developed curriculum – which is to stimulate ideas to assist in the understanding of robotics – it would be unreasonable to expect any participant to become proficient in the designated coding language, especially

if programming the robot is going to be done on the same occasion that the participant is first learning that different coding languages even exist. Fortunately, there is a common practice that promotes code transferability from language to language and this concept is called “pseudo-code”. Pseudo code can be described as a false code or code representation that is designed to be understood by one with little to no programming knowledge due to usage of layman’s terms and simple structure (How to write a pseudo code, 2021). Unlike most technical languages, this technique follows few rules in terms of syntax. It still promotes clear use of controls structures and groups, such as if-then statements, for loops, while loops, and basic case structures. If one uses control structures such as that just stated, proper indentation should be used to show which instructions belong to what structure group. Many also suggest that all variable names for pseudo code be kept in lowercase letters for consistency reasons (How to write a pseudo code, 2021). Figure 4 shows a shift in an Arduino based code syntax to a pseudo code syntax. The program that toggles an LED for one second, two seconds, and then instantaneously should appear understandable when placed in the pseudo code format. This example does not show any use of control structures, but rather a flat non-looping approach that executes once straight from top to bottom.

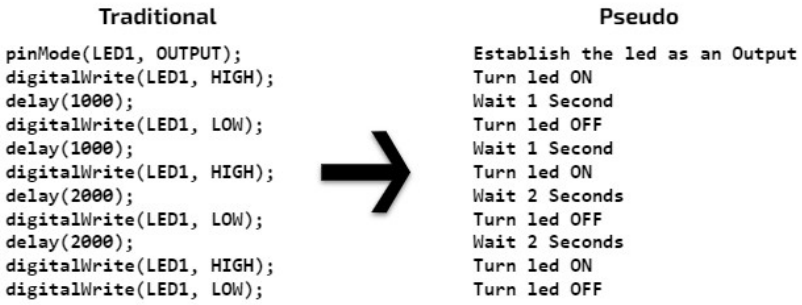

Figure 4. Pseudo Code Example 1

Although pseudo code is incredibly useful in helping individuals understand complex code because it puts code in a general language, code editors (computer applications used to upload code to a robot's controller) generally do not understand this "false code" or informal programming style. Since code editors sound their alarms with errors when a user makes even the smallest syntax or punctuation mistake, the developed curriculum embedded the next best thing to pseudo code, an abundant source of functions. A "function" in the programming world is a segment of reusable code that is used to perform specific defined actions (Computer Programming – Functions, 2021). By creating functions, participants would have to catch on to only minimal rules of syntax, such as how when using a function, the user must type the name of function followed directly by a set of parentheses and a semicolon. Figure 5 shows an example appearance of both application of an instruction in the Arduino C++ language application of a self-built function resembling a pseudo code similar format. Functions may simplify the coding experience of a new user and closely resemble pseudo code by allowing the user to express an action rather than forcing the user to know a full list of technical instructions to properly operate an electronic component. For example, an ultrasonic sensor measures distance through a calculation initiated by sending out a sound and tracking the amount of time it takes for the sound to bounce off of the object being detected and return to the sensor. The technical program code for this process can be seen by the left side of Figure 6. By applying this function as seen on the right side of the

figure, the user can easily calculate the detected distance and display this distance on an internal monitor which the user can then use to view the detected distance.

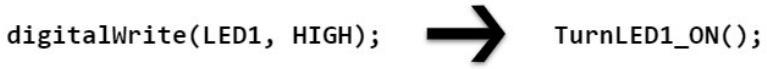

Figure 5. General Function Example

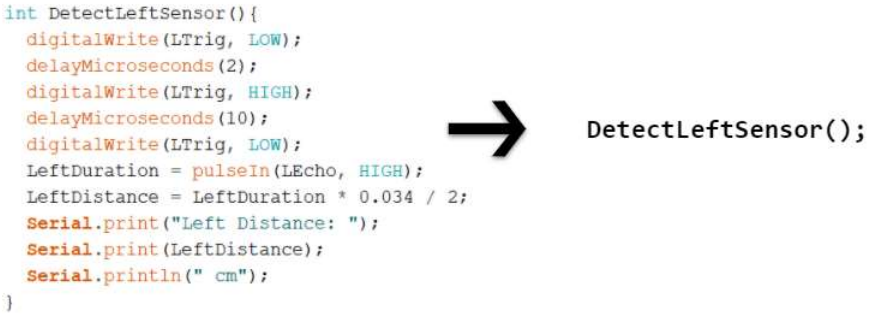

Figure 6. Ultrasonic Sensor Function Example

Using a multitude of functions, the curriculum allows participants to express programming ideas and sequence of events without needing much technical programming knowledge. To assist with recalling and arranging common coding actions for coherrent idea organization, physical blocks of paper containing function and control structure syntax is provided to participants. These blocks or cards of common coding statements and self-created functions are color-coded by usage functionality and designed to help an unexperienced programmer easily arrange program structure or program ideas without hassle of inputting it into the computer. The list of different blocks of code commands can be found in Table 1. Figure 7 shows how these blocks can be arranged to form a program that continuously flashes an LED at 1 second intervals.

Category	Syntax	Fill-In Examples
Movement	TurnRightWheel____();	Clockwise (CW) CounterClockwise (CCW)
	TurnLeftWheel____();	Still
	SwitchRead();	
	MoveForward();	
	TurnRight();	
	TurnLeft();	
	TurnAround();	
IO Control	DetectLeftSensor();	
	DetectRightSensor();	
	TurnLED1____();	ON
	TurnLED2____();	OFF
Conditionals or Control Structures	while(____) {	Button is Pushed
	if(____) {	Button is NOT Pushed
	Wait(____);	1 Second, 2 Seconds, etc.
	else {	
	loop {	
	}	

Table 1. Command List

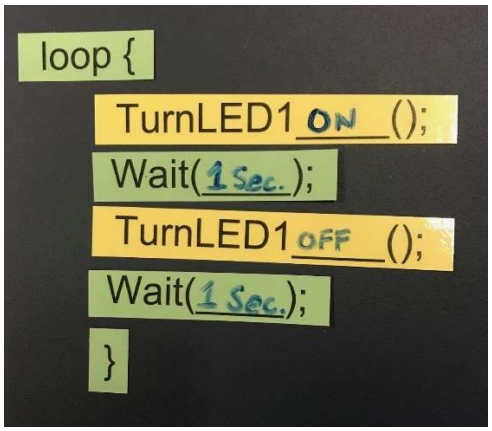

Figure 7. LED Block Example

With the sample code command blocks, small challenges are then given to participants to help the individuals get a practical grasp on how certain logic operators or control structures function. The curriculum refers to these challenges as “Logic Puzzles” and each puzzle is designed to teach a new concept or spark a new way of thinking, however each challenge is not limited to one correct solution. While some of the puzzles

give the participants a fixed set of components or model to work with, other puzzles challenge individuals to determine on their own what components will be needed and what to do with them. Each of the four concept challenges will now be discussed to explain lesson material.

The first puzzle begins by asking what components may be needed to drive a robot forward. Once it is determined that servos or alternative motors can provide the necessary wheel rotation, the model shown in Figure 8 is given in which the participant can then use the command blocks listed in Table 1 to piece together a simple code to move the robot forward.

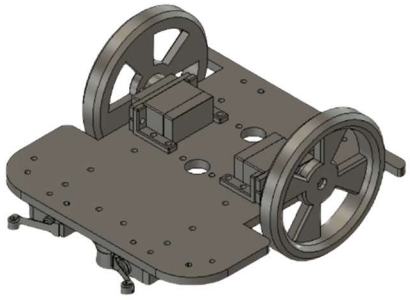

Figure 8. Puzzle 1 Robot

This question is specifically challenging for individuals because it requires an internal concept visualization to understand and get right on the first attempt. The solution to this question is to turn the left wheel counterclockwise while turning the right one clockwise, easily understood through Figure 9.

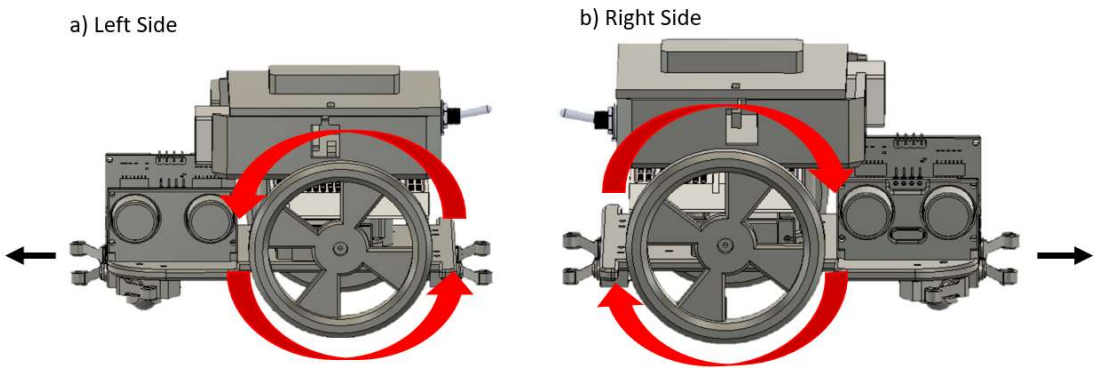

Figure 9. Puzzle 1 Solution Visualization

The next few puzzles focus more on the programming side and begin to introduce usage of control structures. While the term "control structure" sounds quite fancy, the usage of them is quite normal in everyday life and people begin understanding how the logic behind them work beginning at a decently early age. For example, most people understand that using the word "if" generally means that there is a condition or event that may or may not happen, but there are multiple options to the situation, thus *if* the event does not occur, something "else" will occur. In programming, "if-else" structures can be created to perform certain actions on the occasion that some event takes place, however alternative actions can be programmed if the event does not occur. It is also common knowledge that something stuck within a loop will continue repeating forever or until something stops it, thus in programming, code can be written to repeat actions continuously by placing them within a loop. In addition to what is commonly understood, many people automatically have a basic understanding of what a "while" loop does. In general, when one gives a command using the word *while*, one knows to follow a set of instructions until a condition is no longer met. Overall, the *if-else* conditional can be

referred to be a control structure used for selection or decision making, and *while* control structures to be repetitive techniques used to conditionally loop actions (Myers).

Puzzle two asks the participant to select components and write a program that can indicate when a button is pushed. This challenge is included to teach individuals to evaluate the status of input devices and how frequently to do so, and also one should see how an *if-else* structure can be applied. Without the *else* portion of the code, the LED is likely to remain ON until the device is powered off. Puzzle three asks the participant to complete the same task from puzzle 2, but now using a *while* control loop. This is included to challenge the individual to figure out how to escape this conditional loop, which can only be done if inputs information is re-evaluated within the loop.

The final puzzle does not add much more complexity, but simply asks the participant's program to only indicate when two push buttons are pushed. This simple request introduces the concepts of Boolean logic, which again is common understanding for most. Boolean logic requires the understanding of words such as AND, OR, and NOT. The full situation is often represented in what is called a "truth table" which uses 1's and 0's to represent yes's and no's respectively to show satisfaction of a situation. Table 2a shows the truth table of the *AND* logic, 2b for the *OR* logic, and 2c for the *NOT* logic. The curriculum or workshop addresses this topic in terms of when one is satisfied with items received from a food order. Using *AND*, if a customer wants a burger *AND* fries, the customer will only be satisfied when he or she receives both, but will not be satisfied if only one item is received. Looking at *OR*, a customer wants a burger or fries and so if the customer receives either one or both, then satisfaction is met. Satisfaction is not met in the event that neither is received. Finally, the *NOT* logic simply inverts

whatever the input is. For example, if a burger is ordered without pickles, in the *NOT* situation the cook would deliberated put pickles on the burger.

a)

Burger AND Fries Ordered

Was the item received?		Is there Satisfaction?
Burger	**Fries**	
NO	NO	NO
NO	YES	NO
YES	NO	NO
YES	YES	YES

b)

Burger OR Fries Ordered

Was the item received?		Is there Satisfaction?
Burger	**Fries**	
NO	NO	NO
NO	YES	YES
YES	NO	YES
YES	YES	YES

c)

Burger with/without Pickles (NOT)

Were pickles asked for?	Were pickles received?
YES	NO
NO	YES

Table 2. Boolean Truth Tables

Autonomous Navigation Project

While the crash course does provide a very basic information dump to content that is useful to the understanding of robotics, true understanding can be observed through an individual's application of the information. In this case, a robot for small-scale autonomous navigation serves as the application to observe understanding. Participants will have the opportunity to interact with and program a prebuilt robot (shown in Figure 10) that is designed to navigate through a one-way hallway system. This means that while the robot will have to make many unpredictable turns, there is no way for the robot to get lost or run into a dead end. A visual showing the concept of operation is respresented by Figure 11. To program the robot the participant will revisit the command blocks used before and will use these prewritten segments of code to structure the program for the robot. The individual will also develop his or her own functions to be used for common movements of the robot. Since asking an individual to begin by developing the final product is unreasonable for any project, the project is

deliberately broken into small segments which serve as building blocks leading to the final product. Each segment is designed to verify that fundamentals are understood and that components are functioning properly.

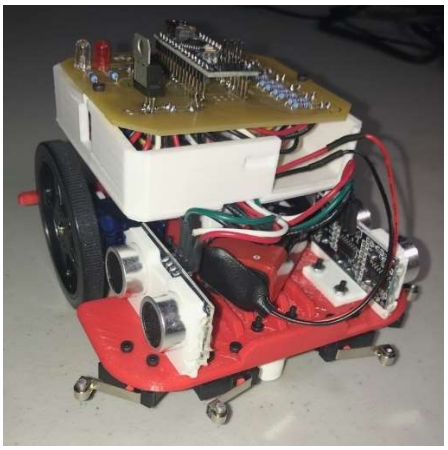

Figure 10. Final Robot Assembly

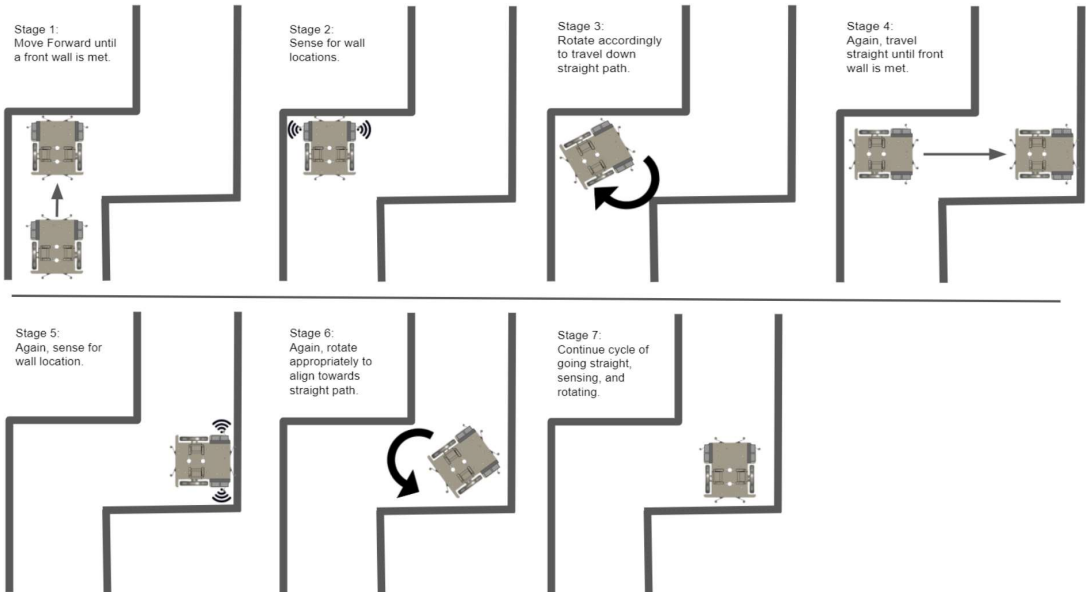

Figure 11. Concept of Operation

To form repeatable movements that can be reused and rearranged in a program, the participant creates his or her own functions such as *MoveForward*, *TurnLeft*, and

TurnRight. The first objective of the project is to develop the *MoveForward* function, which has the following requirements:

1) The robot shall move forward until the robot meets the wall on the front side.
2) The robot shall adjust to direct the robot towards the center of the hallway if the robot bumps into either of the side walls.

It is suggested to participants that each requirement is attempted in block format first and attempted in order and building on the previous requirement. Through developing this function, the participant should practice developing conditionals within control structures. By doing this the participant shows understanding of obtaining input information and using this information to develop a desired output.

Following the *MoveForward* function, participants then develop a function to turn the robot left and a function to turn the robot right. Participants must recognize that the *TurnLeft* and *TurnRight* functions directly follow the *MoveForward* function and so the robot will be located up against a wall on the front side of the robot when the system must decide whether to turn left or turn right. This may be clearer by looking at the position of the robot during Stage 2 shown in Figure 11. Using the robot playing space or navigation space shown in Figure 12, the participants are able to test each of his or her created functions.

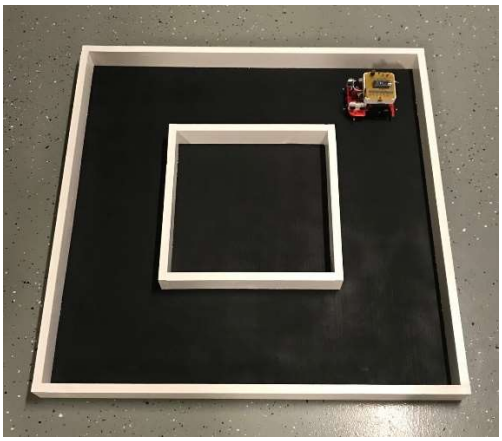

Figure 12. Square Navigation Space

Physically testing newly created programs allows the user to immediately evaluate the level of success along with potential areas that need troubleshooting. Once these three general movement functions are created and working properly, the participant should then see that using the movement functions and only these functions appropriately within a loop, the robot should not only be able to navigate the square navigation space in Figure 12, but also the changeable or adaptable hallway system shown in Figure 13.

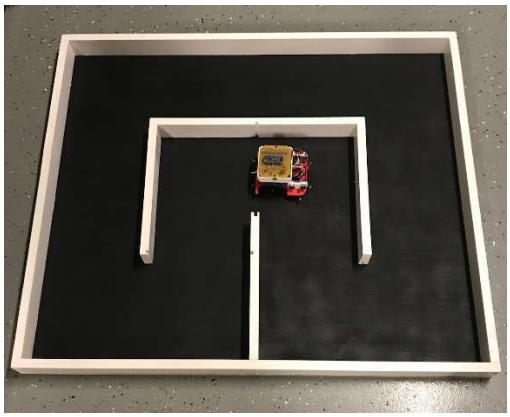

Figure 13. Adaptable Navigation Board

While it is possible to use only the general movement functions so that the robot can continuously navigate around the hallway system in Figure 13, the overarching issue

that such a program has is that this program is most likely not adaptive. This means that this program works on one situation only and the robot must be placed to start in a specific location with the robot facing a specific direction. Starting the robot in any other location or direction or if changing the course in any way will cause the program to fail in terms of navigating the robot through the course.

The final task for the project is for the participant to figure out how to use the components on the prebuilt robot to develop a program that allows the robot to navigate the board no matter the starting position or if an additional wall is inserted to block a path, such as shown in Figure 14. To assist with solving this issue, ultrasonic sensors were added on the robot to allow the participant additional input information in terms of side distance sensing and depth perception. Using a single one of these sensors or using both, there are many different solutions to developing a program so that the robot is able to decide when to turn left or when to turn right in order to adapt to the changing navigation space or navigation board. One solution may be simply comparing each side off the robot to a wall and whichever side has the fartherst wall is the direction the robot should turn. Other solutions may pick a specific numeric value and compare the left, right, or both sensors to that value to determine what the robot should do. For example, the program may state that the robot should turn left if the left wall is greater than 18 centimeters away and the right wall is less than 18 centimenters away.

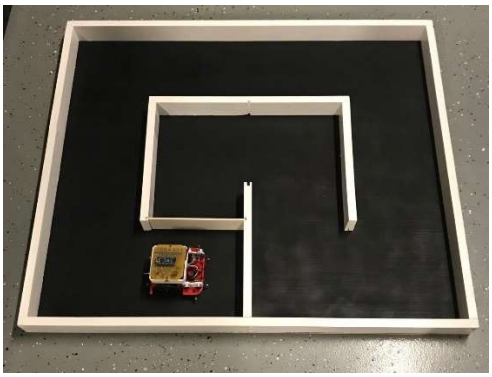

Figure 14. Altered Adaptive Navigation Board

SECTION THREE

Curriculum Reflection

Following the development of the curriculum, around a handful of individuals were able to participate in parts of or the full curriculum plan. From each session there were takeaways for content within the crash course that could be improved to be taught better to a non-technical participant, tactics to make the curriculum flow better in the desired direction, suggestions for how to create more clear instruction for participants, and much more. Since no official data was collected during any of these robotics workshops, general observations of struggles and successes guided these takeaways.

From the first workshop it was obvious that certain precautions needed to be taken care of regarding the tangible code blocks. Since the code blocks are simply pieces of paper or cardstock and not directly linked to anything digital, it was apparent that these blocks needed to be verified according to the Arduino code template each participant started the workshop with. Multiple instances occurred where typing in the code block functions (which were supposed to be verbatim) led to errors within the Arduino software. While these issues may be something simple such as adding a semicolon here or there, or maybe changing the spelling of a function name at the top of the template, receiving errors while attempting to prove concepts to participants or receiving errors while the participant is attempting the project often leads to the loss of interest by the participant and confuses the participant to as if what they had done was correct. For example, the *TurnLED2_ON* function originally gave a program error when attempting to verify participant program developed by a participant with the blocks. While the

troubleshooting only took less than five minutes to figure out that a copy-paste error had been made and that the function said *LED1* somewhere it should have said *LED2*, the participant had already forgotten what the designed program was supposed to do and so a review was necessary. While the template was used to verify that the final robot project was possible, this issue could have been prevented by doing a run through using the code block for all examples and not just the final project.

While it is commendable if others can solve the same problem in alternative ways, in terms of teaching specific content, I found that it is generally a good idea to guide participants down the path that teaches the desired material first and then open the floor for alternative ideas later. When going through the logic puzzles in the initial crash course during the first session with participants, I found that allowing participants to attempt alternative methods than what the puzzle was designed to teach resulted in counterproductive measures. This issue was quite apparent when all blocks commands shown in Table 1 were given to participants at the same time. In the first robotics workshop all cards were distributed at the same time and it seemed more frequent that participants would lean towards using commands and control structures prior to the logic puzzle that was implemented to teach or learn about that type of command or structure. For example, *if-else* conditions within a standard loop are often interchangeable with *while* loop structures also within a larger loop. In puzzle 2 the participant is asked to develop a program to light an LED if or when a switch is pushed. Since the status of the switch must continuously be evaluated to determine when the switch is actually pushed and the instructions did not specifically tell the participant to use an *if-else* statement, it was common for participants to jump straight to looping the program with a *while* loop

within, which solves the problem, but also jumps to the solution of puzzle 3 while neglecting to practice of the *if-else* structure expected in puzzle 2. While giving participants additional command blocks that were not meant to be used at the moment may have caused part of the issue here, it would also have been useful to deliberately state the concept goal within each puzzle.

Keeping the overall goal of the workshop in mind – which is to increase understanding of the field of robotics and incite interest in the field – the robotics crash course seemed to slightly bore participants during the small topical lecture, but increase the understanding of the individuals. Throughout the crash course most participants were able to understand the majority of included content and perform the developed logic puzzles using the provided command blocks. For concepts such as control structures and Boolean logic, most participants found that this material was familiar with a technical relation to what is thought to be common sense, but had not thought of the material in numeric or instructual terms. When it came to the logic puzzles, the participants needed to be guided to which control structures or logical ideas to begin with, but were then able to create their own program using the blocks and achieve a functional program within two to three attempts. Since the autonomous robot project was a more hands-on approach to learning than the topical crash course, particpants seemed to show a greater interest and activity level once beginning on the project and they also showed deeper understanding of subjects.

Results of the autonomous navigation project proved positive, but also revealed its own challenges. One of the obstacles that hindered flow of the project related to the chosen solution of using functions to compensate for the inability to program with

pseudocode. While it was convenient to expose participants to a true programming technique by allowing participants to build his or her own functions for *MoveForward, TurnRight,* or *TurnLeft,* the functions were originally viewed as the tangible code blocks, by not initially removing the function blocks that were to be made by participants and by not teaching about functions within the crash course, there was confusion. By not removing function blocks such as *MoveForward, TurnRight,* or *TurnLeft* from the start, participants in the first session had trouble understanding that basic movements sometimes require specific commands to various parts of a system, such as turning the left and right wheel separately. Instead it was initially believed that one could simply just tell the robot to move forward or to move backwards, without specific instruction to any specific component. Moving into the second session, the command blocks or function blocks mentioned before were removed from the start of the project and were given to the participants as he or she completed the function for each of the basic movements. At that point it was explained the purpose of a function in programming and how the technique can be used. Functions were explained in relation to cooking tutorials. As one goes through a recipe, he or she may come to a step that calls for one to use a fancy piece of equiptment such as an industrial mixer, thus he or she may refer to and follow the instructions in the user's manual to operate the mixer and then return to the steps of the recipe. Using this example, functions were described as alternative sets instructions or steps that could be inserted at any point of an overarching group or set of instructions. By presenting command blocks appropriately and explaining usage of a function within the second session, participants showed a much greater understanding of the subject matter.

Although project performance for each step of the project pointed in a successful direction, it was quickly noticed from the first session with participants that breaking the project into 3-4 steps still left the project too vague for the entrants to robotics. For example, during the first stage where participants were asked to create a program that moved the robot forward until a wall was met, participants frequently failed to realize correlation between using input conditionals to control outputs and they also often failed to understand what was asked. The original display of instructions was given in the format of the crash course slides along with specific stage project requirements. The concept of operations shown in Figure 11 was also given as visual representation of what was needed to be done. These written requirements and visual representations did not appear effective and so they were scrapped in the presentation of the project. A possible loss of effectiveness for the concept of operations diagram may have resulted from the necessity to quickly digest the operation as a whole and internally break the diagram into the desired stage, all to keep the workshop on a reasonable pace. Instead, participants appeared more comfortable and adapted better to the project requirements when they were physically able to hold the robot and navigate the robot by hand on the navigation board. The tangible and learn by physical touch and play technique seemed much more effective in this situation than the learn by visuals and reading.

Another approach that assisted in participant comprehension of robotic design needs was additional breakdown of each subsystem or feature. The original project consisted of three overall steps with one sub-step. The plan was to allow participants to attempt to figure out the majority of the project on his or her own, consisting of the following portions: creation of the *MoveForward* function, programming the robot to

make turns, and programming the robot to detect the open path. While this does sum up the autonomous navigation project from a wholistic perspective, the size of each step was much to large to be taken on at once by a beginner. Realizing this, the workshops adapted to break down each step into may substeps that each needed to be verified before moving on. For example, a group should begin the project by proving the robot would in fact go straight with developed instructions. If the results were not as expected, participants could analyze what needed to change, such as if one wheel needed to be sped up to solve a swerving robot caused by difference in wheel speeds. From here the participant could add to the program to make the robot stop when it reached the end wall. The robot was then verified and programmed to adjust if it veered off into one of the side walls, verified and programmed again to adjust for veering in the other direction. By taking much more minute steps, participants seemed to get on the right track quicker and experienced less confusion regarding how to begin the program with the command blocks. Even with smaller, more direct steps, participants needed close guidance and often took around 3 or more attempts to complete segments of the project adequately.

Reflecting on the crash course and robotic project as a whole, participants were familiar to some logical concepts used in basic robotics and showed a greater understanding of robotics as the sessions progressed. Individuals expressed that the topical crash course content was less interesting than a hands-on project, but was crucial information for understanding how technology operates and processes instructions. The autonomous navigation robot project showed that individuals can understand robotics on basic levels with minimal exposure to the field. Completion of the robotic project took

near twice the amount of time as originally expected for participants, however the small project was able to be completed within a couple sessions of around three hours each.

SECTION FOUR

Robotics Project in the Making

The following section is dedicated to the developmental stages of the autonomous navigation project and a will discuss steps taken to reach the final product used in the workshops. More specifically the background for what the project was based on will be discussed, along with theoretical changes to prior designs, how the project was modeled or planned with engineering software, and how the tangible robot was created.

Background of the Project

The autonomous navigation robot discussed throughout this paper is the result of two alternative navigational robots. The bulk of the robot resembles a robot created by the Western Kentucky University (WKU) 2021 Institute of Electrical and Electronics Engineering (IEEE) robot team and small concept design alternatives were decisions made in light of a common line following robot that is often used as a beginner's project for students interfacing with electronics. Both sources of inspiration will be explained briefly.

Each year, WKU enters a southeastern regional IEEE robotics competition and the basis of the 2021 competition was developing a robot that can autonomously navigate a playing space to locate and pick up various blocks hidden behind wooden barricades. This navigation portion of this project is tricky because although the playing space is symmetric, the given space does not show consistent patterns (seen in Figure 15) and design parameters required by the competition rules required a robot within a 7x7x7 inch

cube, which limits many design options because of its small size (Hardware Competition Rules, 2021). The competition navigation space also features multiple locations where obstacles may be placed at random to obstruct the robot's path.

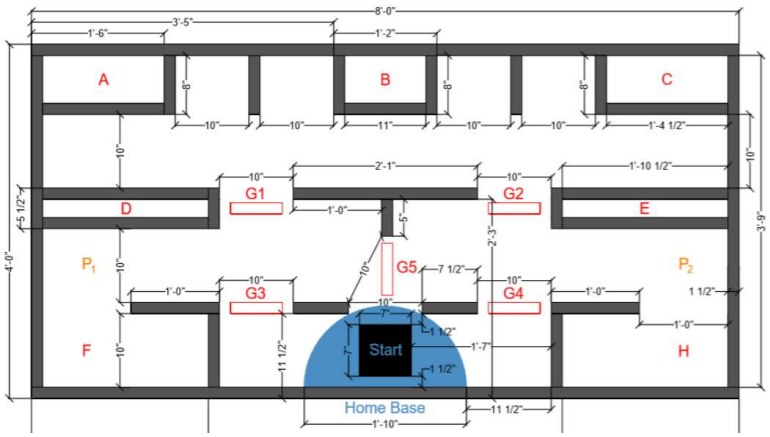

Figure 15. IEEE Competition Navigation Space

To develop a robot that could navigate around the given space, our team developed a robot that uses limit switches in at least six locations around the edges of the robot to feel its way around the course and align itself when needed. The robot also features a single ultrasonic sensor used to detect wall distance straight ahead. This robot can be seen in Figure 16. Despite having multiple input devices and attempting to use various other sensors, the navigation portion of this robot struggled to consistently make it around the board due to various aspects such as irregular board layout, inconsistent servo motors, limitations of sensing technology, and other small inconveniences. Due to limitations of usable components, much of the robot's navigation needed to be hardcoded, which in other words means forced by very specific code that is not transferrable if any relevant factor were to change. Knowing that hardcoding is often not a suggested technique to regularly use, I decided to alter the factors and goals of the overall project to develop a new project that was simpler and could allow the robot to

make decisions on its own. To do so I related the limit switch navigation method of the IEEE robot, which was used for feeling location and robot alignment, to a commonly used line follower robot.

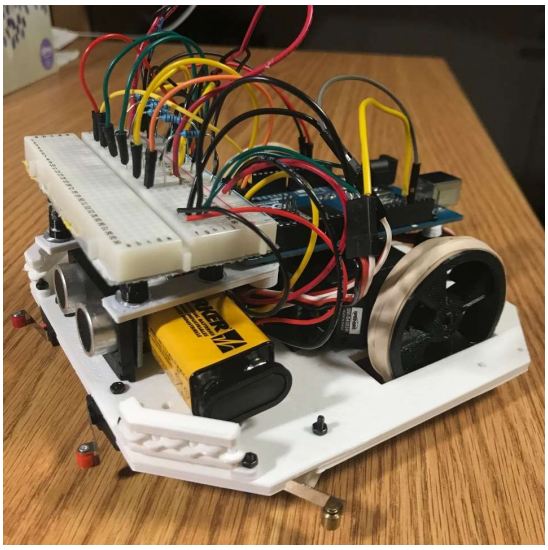

Figure 16. IEEE Robot

The typical line follower robot can be described as a beginner level robot that is able to follow a white line on the black ground (or vice versa) wherever the line twists or turns. The robot often uses two IR sensors to differentiate between a black and white surface. Placing a sensor on both sides of the line, when either sensor detects the designated color of line, depending on which sensor detects it, the robot knows which way to adjust while moving forward.

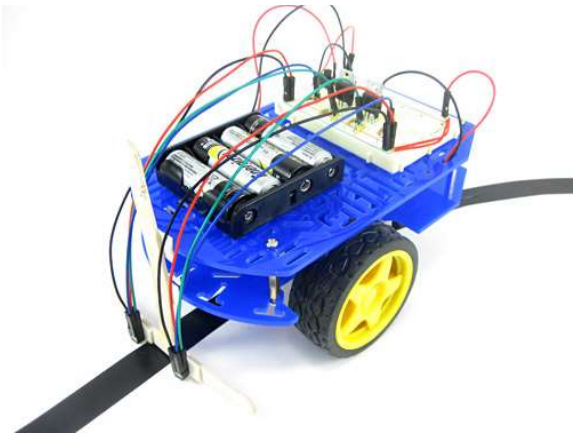

Figure 17. Line Following Robot Example (Science Buddies, 2020)

Similar to the line follower robot, I wanted to create a system that the robot could travel autonomously, but while also using navigational techniques used on the IEEE robot. Following this idea, the robot used for my autonomous navigation robot would at least use multiple limit switches around the edges of the robot, but a new method would need to be arrived at to allow the robot to make its own decisions. Rather than having a continuous line to follow that could vary in how gradual it turned, the new design would simply use perpendicular walls that could be detected by some other device.

Theoretical Design

While it was decided that the autonomous robot project for my developed curriculum would adopt the limit switch design from the IEEE robot and would navigate a playing space with perpendicular walls, there were still many design decisions that needed to be made such has how to automate the changing directions, how to power the device to maneuver freely, how to design a device to be handed off to individuals unfamiliar with almost all electronics, how should the playing surface be arranged to prove project completion. Many of these design decisions are highly related and so

making a quick and definite choice for one area was often not an option. For example, the power source for a robotic project is dependent on what components are needed for the robot to function and how long these components may be operating at any given time, but which components are used may be dependent on the arrangement of the playing surface, which may change based on the size of the robot, again relating to which components are used. All of the relations can quickly get confusing if not carefully organized.

To organize the project's relations, a large part of creating the project followed the Vee Model often seen when using a systems engineering approach. The basic idea of how the Vee Model operates can be seen in Figure 18. While the overall development stages flow from left to right of the V shape, validation of the either side of the model can be checked by performing the step directly across from it in the model. For example, by testing various components, I was able to nail down a detailed design of the robot.

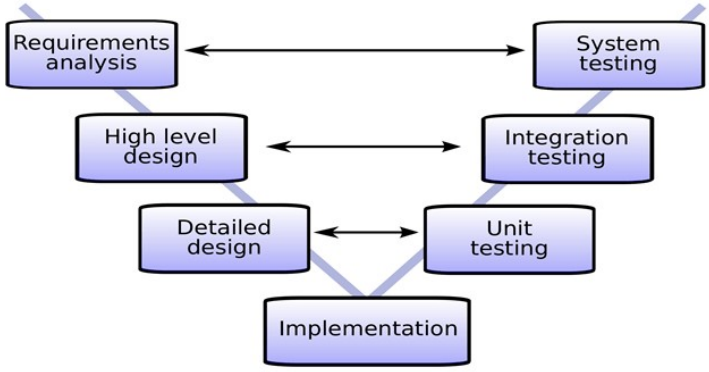

Figure 18. Simplified Vee Model Approach (Admin, 2019)

Starting out the robotic project, I set my own system requirements to define the desired functionality and physical constraints. A functional requirement may have appeared as, "the robot shall detect wall locations from both sides." A physical constraint

may have appeared as, “the robot shall fit within a 6x6 inch square.” By creating these requirements, a high-level design begins to emerge. While it is not clear what specific components will work best, it is understood that components on the sides will need to be able to detect presence of a wall. The requirements also help understand that spacing of the playing space needs to accommodate for a 6x6 inch robot to navigate around, thus having walls nine inches apart seems to be a reasonable start.

There were a few design alternatives for electronic components on the robot, however most components were decently straightforward choices. The most important decision was choosing what components would be used to detect wall positions when the robot needs to turn a new direction. The two sensor options were between IR sensors (detect objects based on light reflection) and ultrasonic sensors (detect distance of objects based on time for sent sound waves to return). Both types of components were tested, however it was found that the given IR sensors could only detect walls within 5 centimeters (almost 2 inches) away, despite being guaranteed by the manufacturer to detect up to 30 cm (almost 12 inchs). While this could potentially work as needed, there is a possibility that the robot arrives at the end of the hallway off-centered and 5 cm or more from both side walls, thus not detecting presense of either wall and not able to determine which direction is the open path. The solution to this issue was to use something that could detect a farther distance, however all IR sensor alternatives were reviewed to perform much under the manufacturer’s specification, thus the ultrasonic sensor was evaluated.

Since ultrasonic sensors measure using calculations on the speed of sound waves, the original concern was that using multiple of these sensors would result in issues with

the sound waves interfering with each other – in other words a sensor receiving the other's sound wave – thus receiving incorrect results. After testing these two sensors pointing in opposite directions as they would be on the robot, it was noted that the interference did not occur. The advantage of using these sensors was that they could reach up to around 60 cm (just under 2 feet) and give the user a numeric distance reading rather than a binary (yes/no) answer regarding the presense of a wall. This solution supported the design option of using two ultrasonic sensors for wall detection.

One design aspect that was overthought was the options for a power supply. To decide the requirements for this segment, a table (shown in Table 3) was constructed to show the amount of power draw from the robot as the worst-case scenario. The table shows the necessary voltage to for each component along with the current draw of each component when it is consuming the most electricity. By doing this I was able to determine the smallest amount of voltage of battery needed to operate the whole robot adequately and then I was also able to determine an estimate of how long a battery may last if the robot is consuming energy in an inefficient way. From this table it was concluded that a 9-volt battery was sufficient, however a rechargeable battery is ideal because a typical 9V battery has a capacity of around 650 mA*hours, thus would only last about 30 minutes. Using a battery with a capacity of 800 mA*hrs, there were no issues with power draw or operation time not lasting long enough to complete a workshop. Since the robot was continuously turned on and off during the workshops, the total amount of operation time was realisticly the matter of minutes per session. Original assumptions were that the robot sacrifice consistency in functionality as the battery

drained and so larger batteries with higher capacities were first explored, but not necessary.

Part	Quantity	Volts	mA	Current Consumption
FS90R Continuous Servo	2	4.8 - 6	650	1300
HC-SR04 Ultrasonic	2	5	15	30
Arduino Nano	1	7-12	20	20
Arduino Pins	2	5	5	10
			Total	1360 mA

Table 3. Robot Power Consumption Estimates

With the components and battery selected, the overall size of the robot could be estimated and the navigational board could then be created. Due to the usage of ultrasonic sensors, it was reasonable to require all turns to be full right angles as stated before. This is necessary because sound waves need to hit the wall perpendicularly to bounce straight back to the sensor. The other large consideration of the board arrangement was the functionality of the space. To allow participants to best see results of robot programming at various stages, the board needed to specifically allow the participant to test functions for moving forward and making turns. This was best done through a square system of straightaways so the participant could test one function at a time by starting the robot in different directions. This navigation space is shown in Figure 12 from earlier. To adequately show that the autonomous feature in the final product, a second board was developed as shown in the earlier Figure 13.

Computer Aided Design (CAD)

To develop permanency and meet desired design specifications, the robot body, circuitry, and navigation boards were created and calculated using different computer aided design softwares.

The robot itself took a large sum of computer modeling to ensure a precise fit for components and the assembly of. Fusion 360 was the 3-D modeling software used to create and virtually assemble most all parts of the robot. Figures 2, 3, 8, and 9 are all examples of the robot on the software while only showing desired components. While the robot was created from scratch, the fitting of specific purchased components was often supported through shared models made by either the component manufacturer or another individual who has taken time to recreate the component in the software. Utilizing the file sharing through databases such as GrabCAD, I was able to create my own skeleton as shown in Figure 3 to tightly hold the robot together. The base of the robot alone contains just under 40 locations that holes must be placed to secure components and brackets down with bolts. Knowing that each hole is only two millimeters in diameter and the shifting of any hole by a simple millimeter or two could fault the integrity or fitting of the entire design, the precision of each part and alignment is of upmost importance and needed to be modeled on a computer to ensure proper calculations. Resulting in an incredibly compact design, the final product of the CAD model can be viewed in Figure 19.

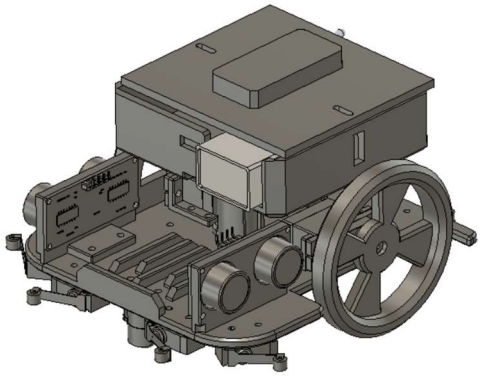

Figure 19. Full CAD Assembly

Development of the two navigation boards followed similar steps to develop as the robot. To achieve a flexible design with automatic calculations to assist with the later fabrication process, these boards were also first designed in Fusion 360. Since the boards were to be made of wood, the virtual designs gave accurate measurements to allow me to total the exact amount of materials necessary to create the project. The size of both boards were modeled by placing designing the boards on top of a 4x8 foot rectangle, which is a standard size of plywood. From here the exact end dimensions – such as a 2x3 inch – of standard wooden studs were used to frame in the boards. The requirements for these boards are quite simple. All walls shall be nine inches apart where the robot may travel. Each turn shall be a right angle. The first board shall allow the robot to travel in a square pattern. The second board shall utilize both left and right turns in an unpatterned manner.

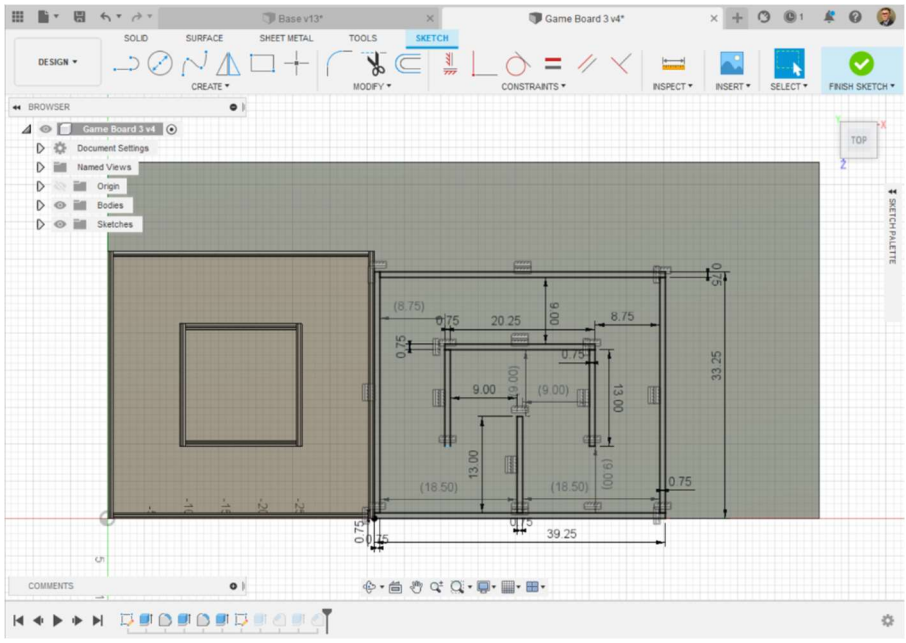

Figure 20. Navigation Boards in CAD

Moving away from the physical modeling, it became evident that the robot maintain some sort of permanency when it came to its electrical wiring and the thought of handing the robot to an untrained individual to test with. While jumper wires may work effectively in prototyping, these wires appear messy even when they are strategically placed. The simple loss of a single wire could result in complete system failure and so relying on this method of wiring is quite risky. To solve this issue, Eagle circuit board design software was used to recreate the seemingly messy wires found on the prototyping board. The creation of this computer design leads to a physical circuit board similar to what one may see if their household electronics were taken apart.

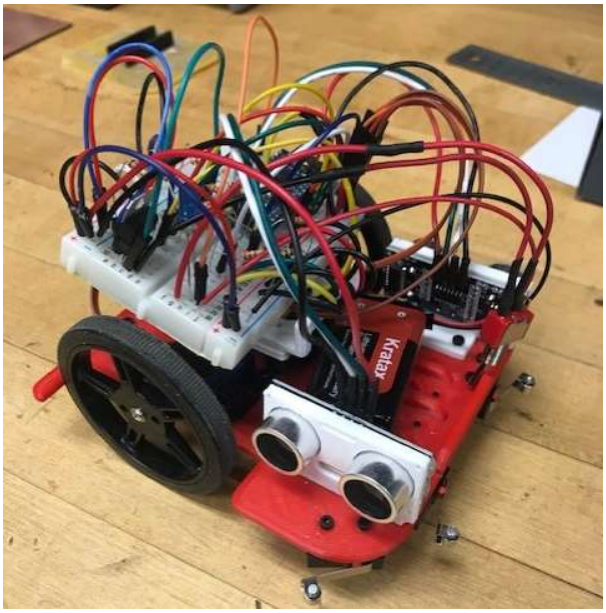

Figure 21. Prototype Wiring

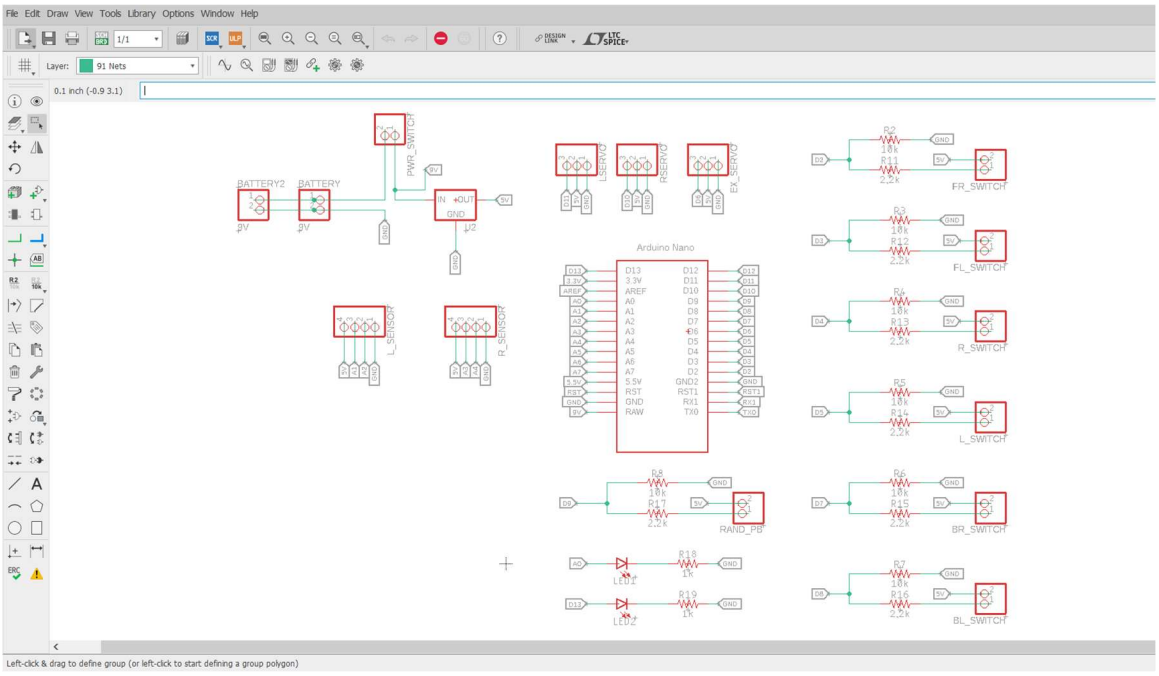

Figure 22. Eagle Circuit Board Schematic

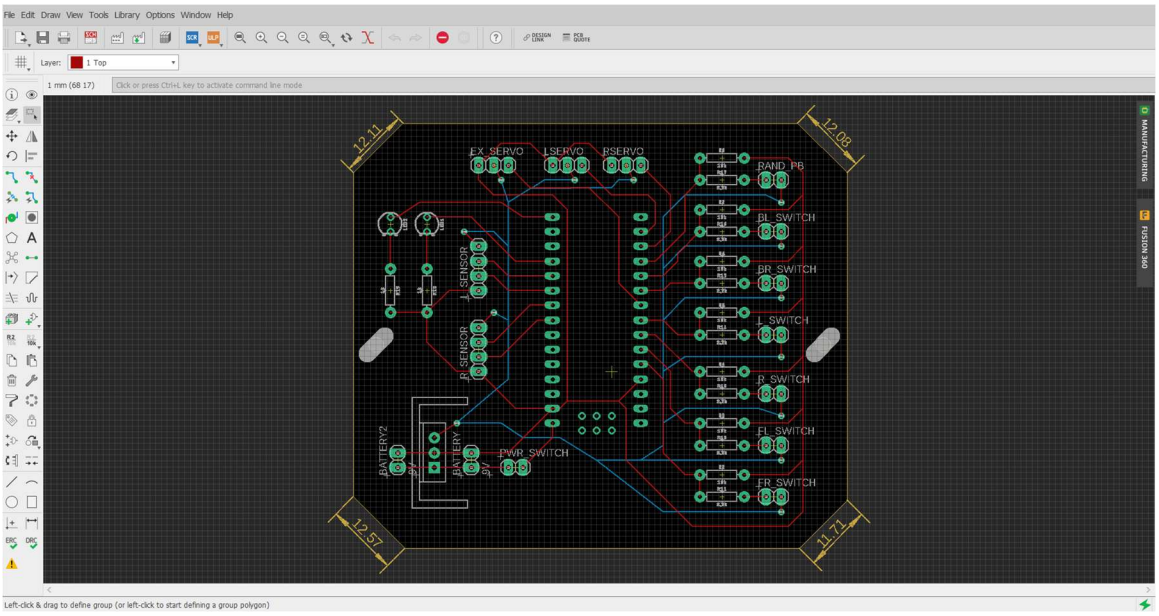

Figure 23. Eagle Circuit Board Layout

Final Implementation

CAD designs served as a bridge from the theoretical design that made way for the robot to be physically possible. With the CAD models, the physical models were made could be physically produced with the right machinery. With the models made for the robot body in the Fusion 360, the files could be translated to into code to make the skeleton components printable with a 3-D printer. With the navigation board models, proper measurements were provided to cut wood needed to assemble the robot's space of operation. The Eagle board layout files made it possible to print out a circuit board that could be soldered for permanent used without worry of a loose wire.

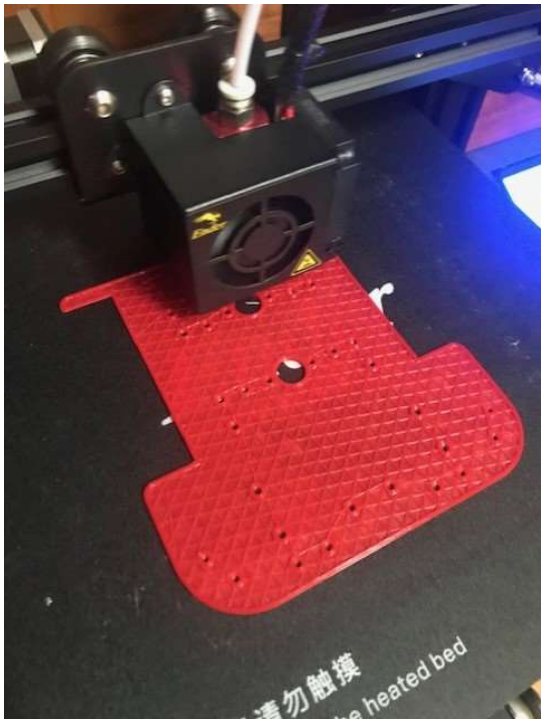

Figure 24. 3-D Printer Producing Robot Base

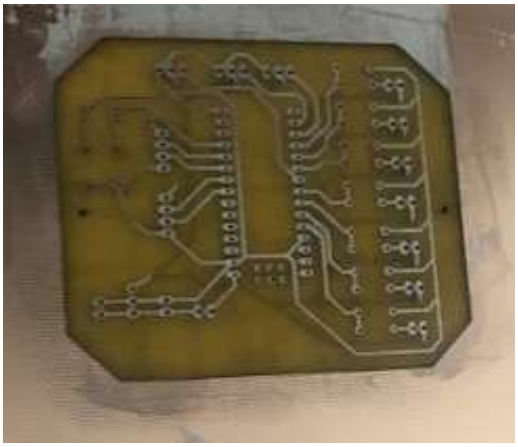

Figure 25. Circuit Board in Production

Through production of 3-D printed skeleton parts of the robot and the circuit board, the components were able to be wired properly to function just as the original prototype. Although not discussed in the prior pages, the original prototype was a consisted of a cardboard skeleton with all components super-glued in place. While this

served as a proof of concept, this cardboard robot met all functional requirements implemented in the CAD models. Comparing the two shows just how far the development of the project came to make the autonomous navigation robot usable for the robotic workshop participants.

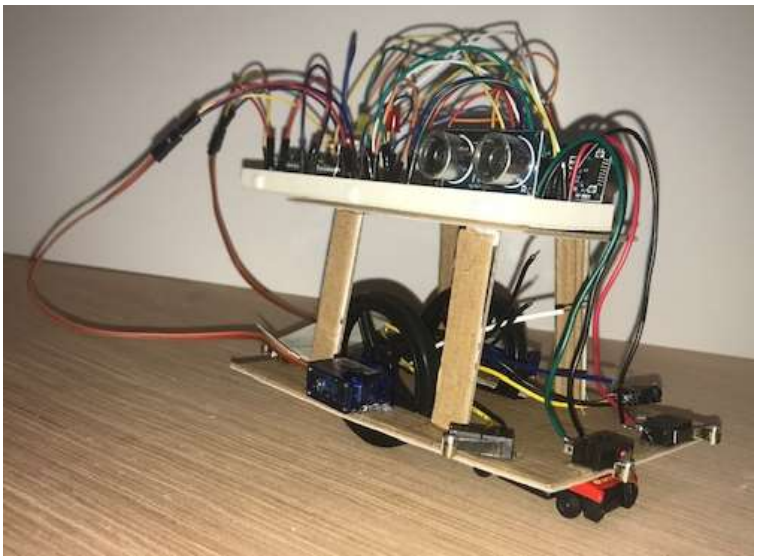

Figure 26. Original Autonomous Navigation Robot Prototype

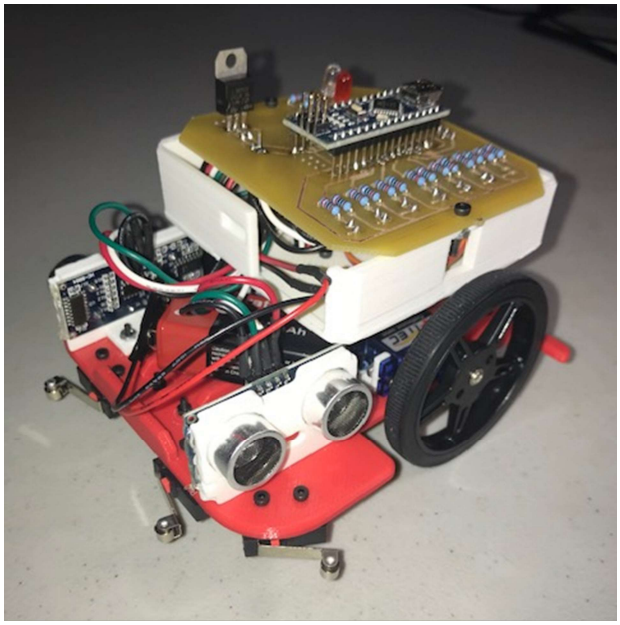

Figure 27. Final Autonomous Navigation Robot

REFERENCES

Admin. (2019, March 03). Pros and Cons of the V Model. Retrieved April 02, 2021, from https://prosancons.com/education/pros-and-cons-of-the-v-model/

Computer Programming - Functions. (n.d.). Retrieved April 02, 2021, from https://www.tutorialspoint.com/computer_programming/computer_programming_functions.htm#:~:text=A%20function%20is%20a%20block,perform%20a%20single%2C%20related%20action.&text=Different%20programming%20languages%20name%20them,%2Droutines%2C%20procedures%2C%20etc

German National Library. (n.d.). Robotics. Retrieved April 01, 2021, from "German National Library". International classification system of the German National Library (GND).

Hardware Competition Rules. (2020, December 15). Retrieved April 01, 2021, from https://attend.ieee.org/southeastcon-2021/wp-content/uploads/sites/213/Hardware-Rules-v4.1.pdf

How to write a pseudo code? (2021, February 02). Retrieved April 02, 2021, from https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

Keim, R. (2019, March 25). What is a microcontroller? The defining characteristics and architecture of a common component - technical articles. Retrieved April 02, 2021, from https://www.allaboutcircuits.com/technical-articles/what-is-a-microcontroller-introduction-component-characteristics-component/

Merriam-Webster. (n.d.). Mechanics. In Merriam-Webster.com dictionary. Retrieved April 1, 2021, from https://www.merriam-webster.com/dictionary/mechanics

Myers, B. (n.d.). Control Structures - Intro, Selection. Retrieved April 02, 2021, from https://www.cs.fsu.edu/~myers/c++/notes/control1.html

Science Buddies. (2020, May 02). Line-Following Robot: Lesson Plan. Retrieved April 02, 2021, from https://www.sciencebuddies.org/teacher-resources/lesson-plans/line-following-robot