Autonomous Delivery Robot

EEE 193B Senior Product Design

Justin Dzida *Austin Hendrickson* *Michael O'Neill* *Damon Tregear*
California State University, Sacramento
EEE Department (193B)
Sacramento, California

***Abstract* - Guests and patients need items delivered to their rooms in a timely manner to increase the experience of their stay. Businesses routinely hire hourly employees to deliver items. A better solution is a system that can accomplish these same tasks at a lower cost. Our Autonomous Delivery Robot delivers items inside variety of businesses such as retirement homes, hotels, and hospitals.**

I. Introduction

A requested item is placed in the compartment located on the top of the system. The delivery process will begin when a user types and enters a room number on a Graphical User Interface, (GUI), touch screen. As the system is in its "delivery process" the GUI, communicates with a RFID Reader, and is prompted to begin searching for RFID tags that are preplaced on the walls of a hallway. During the delivery process, the system will turn to avoid objects or humans. If the M6E Nano RFID Reader detects the RFID tag that matches the inputted room number, the system will turn and face the door. The system will greet the user and ensure the recipient received the requested item. When the delivery process is completed, the Autonomous Delivery Robot will return to its starting position.

II. Product Proposal

The objective of our final project is to design an Autonomous Delivery Robot capable of identifying objects in its environment, and navigate through indoor surroundings to deliver a requested item. The robot will be based on a Differential Drive System, using two independently controlled motors and two caster wheels for support. Each motor will implement an attached encoder as feedback for a PID controller to move in a straight motion. To identify objects, an array of ultrasonic sensors will be used on the front, left, and right sides. The sensors will communicate feedback for various object avoidance and wall-following algorithms. Radio-frequency identification (RFID) tags will be assigned to individual rooms. These tags will provide location feedback to the robot in reference to the desired room of delivery. To implement user interaction, a Graphical User Interface (GUI) will act as the main communication system of the robot. The GUI will notify the robot of what action to perform as well as provide basic security features; sending SMS text messages to the user necessary information to properly receive the requested item.

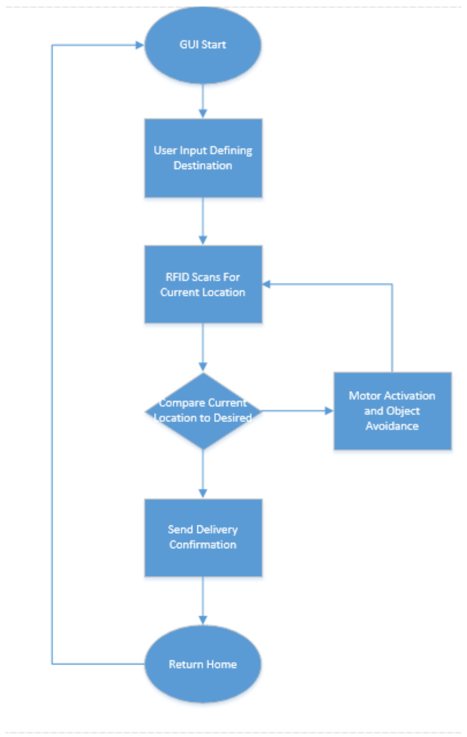
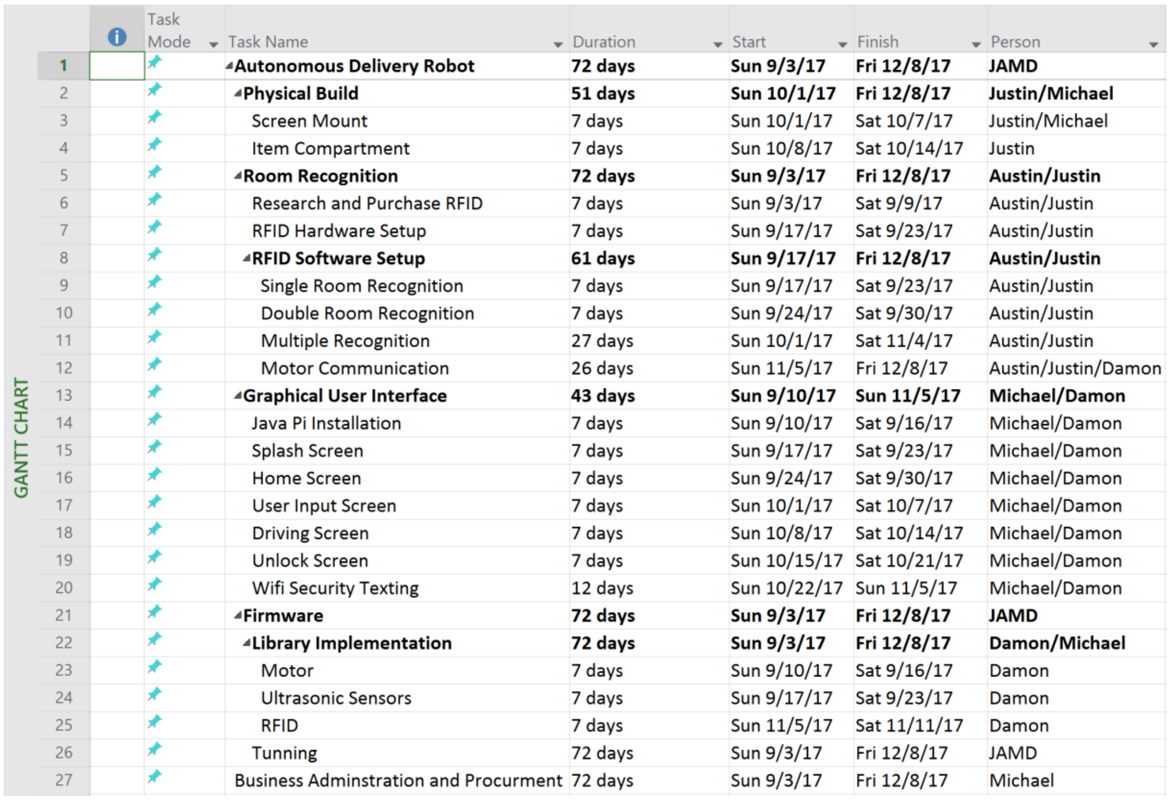

GUI Start
User Input Defining Destination
RFID Scans For Current Location
Compare Current Location to Desired
Motor Activation and Object Avoidance
Send Delivery Confirmation
Return Home

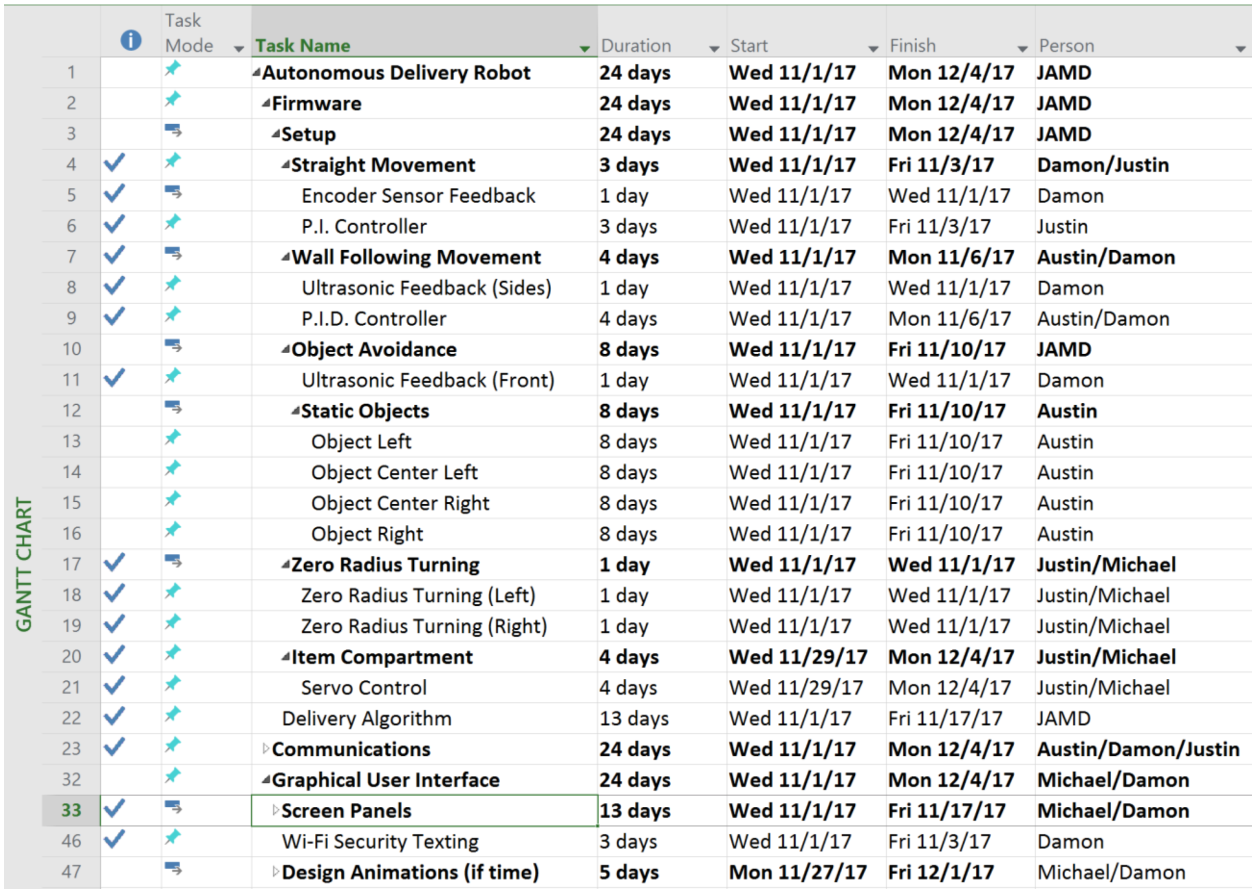
Figure 1 — Complete Delivery Block Diagram

III. Work Breakdown Schedule and Scheduling


GANTT CHART

		Task Mode	Task Name	Duration	Start	Finish	Person
1		📌	**◢Autonomous Delivery Robot**	**72 days**	**Sun 9/3/17**	**Fri 12/8/17**	**JAMD**
2		📌	**◢Physical Build**	**51 days**	**Sun 10/1/17**	**Fri 12/8/17**	**Justin/Michael**
3		📌	Screen Mount	7 days	Sun 10/1/17	Sat 10/7/17	Justin/Michael
4		📌	Item Compartment	7 days	Sun 10/8/17	Sat 10/14/17	Justin
5		📌	**◢Room Recognition**	**72 days**	**Sun 9/3/17**	**Fri 12/8/17**	**Austin/Justin**
6		📌	Research and Purchase RFID	7 days	Sun 9/3/17	Sat 9/9/17	Austin/Justin
7		📌	RFID Hardware Setup	7 days	Sun 9/17/17	Sat 9/23/17	Austin/Justin
8		📌	**◢RFID Software Setup**	**61 days**	**Sun 9/17/17**	**Fri 12/8/17**	**Austin/Justin**
9		📌	Single Room Recognition	7 days	Sun 9/17/17	Sat 9/23/17	Austin/Justin
10		📌	Double Room Recognition	7 days	Sun 9/24/17	Sat 9/30/17	Austin/Justin
11		📌	Multiple Recognition	27 days	Sun 10/1/17	Sat 11/4/17	Austin/Justin
12		📌	Motor Communication	26 days	Sun 11/5/17	Fri 12/8/17	Austin/Justin/Damon
13		📌	**◢Graphical User Interface**	**43 days**	**Sun 9/10/17**	**Sun 11/5/17**	**Michael/Damon**
14		📌	Java Pi Installation	7 days	Sun 9/10/17	Sat 9/16/17	Michael/Damon
15		📌	Splash Screen	7 days	Sun 9/17/17	Sat 9/23/17	Michael/Damon
16		📌	Home Screen	7 days	Sun 9/24/17	Sat 9/30/17	Michael/Damon
17		📌	User Input Screen	7 days	Sun 10/1/17	Sat 10/7/17	Michael/Damon
18		📌	Driving Screen	7 days	Sun 10/8/17	Sat 10/14/17	Michael/Damon
19		📌	Unlock Screen	7 days	Sun 10/15/17	Sat 10/21/17	Michael/Damon
20		📌	Wifi Security Texting	12 days	Sun 10/22/17	Sun 11/5/17	Michael/Damon
21		📌	**◢Firmware**	**72 days**	**Sun 9/3/17**	**Fri 12/8/17**	**JAMD**
22		📌	**◢Library Implementation**	**72 days**	**Sun 9/3/17**	**Fri 12/8/17**	**Damon/Michael**
23		📌	Motor	7 days	Sun 9/10/17	Sat 9/16/17	Damon
24		📌	Ultrasonic Sensors	7 days	Sun 9/17/17	Sat 9/23/17	Damon
25		📌	RFID	7 days	Sun 11/5/17	Sat 11/11/17	Damon
26		📌	Tunning	72 days	Sun 9/3/17	Fri 12/8/17	JAMD
27		📌	Business Adminstration and Procurment	72 days	Sun 9/3/17	Fri 12/8/17	Michael

Table 1— Work Breakdown Structure and Scheduling



GANTT CHART

	ℹ	Task Mode	Task Name	Duration	Start	Finish	Person
1			**◢Autonomous Delivery Robot**	**24 days**	**Wed 11/1/17**	**Mon 12/4/17**	**JAMD**
2			**◢Firmware**	**24 days**	**Wed 11/1/17**	**Mon 12/4/17**	**JAMD**
3			**◢Setup**	**24 days**	**Wed 11/1/17**	**Mon 12/4/17**	**JAMD**
4	✓		**◢Straight Movement**	**3 days**	**Wed 11/1/17**	**Fri 11/3/17**	**Damon/Justin**
5	✓		Encoder Sensor Feedback	1 day	Wed 11/1/17	Wed 11/1/17	Damon
6	✓		P.I. Controller	3 days	Wed 11/1/17	Fri 11/3/17	Justin
7	✓		**◢Wall Following Movement**	**4 days**	**Wed 11/1/17**	**Mon 11/6/17**	**Austin/Damon**
8	✓		Ultrasonic Feedback (Sides)	1 day	Wed 11/1/17	Wed 11/1/17	Damon
9	✓		P.I.D. Controller	4 days	Wed 11/1/17	Mon 11/6/17	Austin/Damon
10			**◢Object Avoidance**	**8 days**	**Wed 11/1/17**	**Fri 11/10/17**	**JAMD**
11	✓		Ultrasonic Feedback (Front)	1 day	Wed 11/1/17	Wed 11/1/17	Damon
12			**◢Static Objects**	**8 days**	**Wed 11/1/17**	**Fri 11/10/17**	**Austin**
13			Object Left	8 days	Wed 11/1/17	Fri 11/10/17	Austin
14			Object Center Left	8 days	Wed 11/1/17	Fri 11/10/17	Austin
15			Object Center Right	8 days	Wed 11/1/17	Fri 11/10/17	Austin
16			Object Right	8 days	Wed 11/1/17	Fri 11/10/17	Austin
17	✓		**◢Zero Radius Turning**	**1 day**	**Wed 11/1/17**	**Wed 11/1/17**	**Justin/Michael**
18	✓		Zero Radius Turning (Left)	1 day	Wed 11/1/17	Wed 11/1/17	Justin/Michael
19	✓		Zero Radius Turning (Right)	1 day	Wed 11/1/17	Wed 11/1/17	Justin/Michael
20	✓		**◢Item Compartment**	**4 days**	**Wed 11/29/17**	**Mon 12/4/17**	**Justin/Michael**
21	✓		Servo Control	4 days	Wed 11/29/17	Mon 12/4/17	Justin/Michael
22	✓		Delivery Algorithm	13 days	Wed 11/1/17	Fri 11/17/17	JAMD
23	✓		**▷Communications**	**24 days**	**Wed 11/1/17**	**Mon 12/4/17**	**Austin/Damon/Justin**
32			**◢Graphical User Interface**	**24 days**	**Wed 11/1/17**	**Mon 12/4/17**	**Michael/Damon**
33	✓		**▷Screen Panels**	**13 days**	**Wed 11/1/17**	**Fri 11/17/17**	**Michael/Damon**
46	✓		Wi-Fi Security Texting	3 days	Wed 11/1/17	Fri 11/3/17	Damon
47			**▷Design Animations (if time)**	**5 days**	**Mon 11/27/17**	**Fri 12/1/17**	Michael/Damon

Table 2— Revised Work Breakdown Structure and Scheduling

IV. Funding Proposal

A. *Components in Use*

This section documents the necessary components for the final prototype of the system. The frame of this system was composed of angle iron fastened together with nuts and bolts with three 1ft square foot pieces of acrylic placed towards the base, middle, and top to hold components. The chassis is composed of steel sheets for the base and siding and aluminum L-brackets to maintain a solid shape with little give. The system is driven by two Pololu 131:1 gearmotors with encoders. Three controllers were used in this system, a Raspberry Pi 3 to control the GUI, two Arduino Megas, one to drive the motors and another to continuously search for RFID tags. The GUI is displayed on a 7-inch Raspberry Pi Touchscreen. The motor circuitry is run by a dual H-Bridge motor driver rather than the original hand built circuit. This system uses 5 ultrasonic sensors to maintain wall distance and run object avoidance. Additional costs include the acrylic shell, wheels, mounting brackets, mounting hubs, a touch screen case, and SD cards.

B. *Unused Components*

This section documents the extra and unnecessary components purchased for the system. Excess parts were bought in case parts were broken, this includes extra 131:1 gearmotors, two extra Arduino Megas, and extra dual H-Bridge motor drivers. There are also components that were replaced along the way such as the transistors used in the original H-Bridge circuit. There were also two sets of wheels that were not used, one had a bearing in it and the other was just for test purposes. Another excess component purchased was the Kinect adapter which was originally used to connect the Kinect camera for mapping.

RFID Materials	$266.94
SMS Messages	$7.60
Physical Build	$250.35
Dual H-Bridge Motor Drivers	$23.97
Pololu 131:1 Motors (2)	$89.85
Arduino Mega	$11.00
Raspberry Pi	$35.00
Raspberry Pi Case	$20.00
2 64gb micro sd cards	$24.62
Raspberry Pi Touchscreen	$77.66
TOTAL COST:	$806.99

Table 3 — Cost Analysis

V. Individual Task Assignment

A. Justin Dzida

The individual tasks assigned to Justin Dzida include physical build, compartment functionality, and assisting with RFID setup. The physical build for this semester was primarily the item compartment but also included mounting the GUI and other needed physical adjustments. Compartment functionality refers primarily to the servo control that opens and closes the lid of the box. The servo control is tied into the GUI so that it opens and closes when specific buttons on the screen are pressed. RFID is our robot's primary means of navigation. Passive RFID tags have been placed at each room replacing the previously used QR codes. RFID tags allow the robot to scan them as it passes and learn what room it is currently by, or its current position.

B. Austin Hendrickson

The individual tasks assigned to Austin Hendrickson included setting up the RFID reader and identifying tags for a multitude of rooms. The mapping system would allow the identification of rooms as well as navigation and correction to stay on target. The tags allow the system to monitor its location in reference to these RFID landmarks and adjust its course accordingly. This system is what allowed the project to navigate the halls and stop off to make its deliveries instead of aimless wandering. The integration of the system involved the use of bits sent over serial communication to the different systems as well as a set of loops that would alter actions based on what input was sensed. Once at the chosen room the RFID will wait for the signal from the GUI to navigate home.

C. Michael O'Neill

The individual tasks assigned to Michael O'Neill included Physical Build, and the Graphical User Interface (GUI) The Physical Build was designed to be three feet tall and 12 inches wide. Theses dimensions were designed to be a perfect fit most recipients to remove their delivered item out of the compartment box with ease. The Autonomous Delivery Robot is designed to be appear friendly, approachable, and just the right size. The GUI was performed on a NetBeans, a software development platform written in Java The GUI is seen on a 7-inch Raspberry Pi Touchscreen on the top of the system. Operators or front desk attendants and recipients can easily interact with the GUI using screens, push buttons that appear on the touchscreen.

D. Damon Tregear

The individual tasks assigned to Damon Tregear included Motor Control, Object Detection, Subsystem Communication, assistance with Object Avoidance, and assistance with the Graphical User Interface. To control the motors for straight line movement, zero radius turning, wall following, and avoiding objects, feedback was needed from the Motor Encoders and the Ultrasonic Sensors. With the respective feedback from each sensor, various P.I. or P.I.D. controllers were implemented to perform the desired actions needed for the system to be autonomous throughout the delivery process. These actions included forward movement upon location input, robot orientation correction for straight-line movement, object avoidance while the delivery is in progress, stopping movement and facing the desired room upon delivery arrival, and returning to the system's starting position. The first sub-process and the ladder two sub-processes involved a creation of a Universal Asynchronous Receive/Transmit (UART) data transmission protocol.

VI. User Manual

A. Graphical User Interface

The Autonomous Delivery Robot interacts with users using a Graphical User Interface (GUI). The GUI is incorporated into a 7-inch Raspberry Pi Touchscreen. The Graphical User Interface (GUI) application is intended to first interact with an operator or desk attendant and later interact with a recipient to ensure their item has arrived. An operator or desk attendant will enter the recipient's phone number and room number. A SMS text message including a passcode will be sent to the recipient. The recipient will enter the passcode and retrieve their item once the robot has arrived at their room. The recipient will also rate the process of the delivery and send the robot back to its home base.

B. Motor and Sensor Initialization

Once the Graphical User Interface is properly setup and seen functioning properly, the motors and sensors for autonomous navigation can be initialized. Since the Autonomous Delivery System comes wired and assembled, the only step needed for motor and sensor initialization is to provide power to the system via the three battery packs. Two battery packs require 4 1.5 V AA batteries, and one battery pack requires 8 1.5 V AA batteries. This is projected to be sufficient for a full day's use out of the system. If the batteries are placed properly in the battery packs but the Arduino Mega does not turn on or the motors do not move when they are supposed to, try placing the batteries in the packs again. More than likely a full connection was not established, so have a multimeter handy to check if a voltage above 7 V is read.

C. RFID Room Recognition Setup

Setting up this systems room detection is a relatively simple task. Initially the tag had to be read then we assigned this specific tag to a room number. However, the initial signal strength was too low, so an antenna had to be attached and a power redirect needed to be soldered. Along with this new antenna an additional 5 V battery pack was added to allow the system to run at 20 dBm. The antenna is placed in the middle of the robot flat and can sense anything to the side or above it up to about 8 feet away.

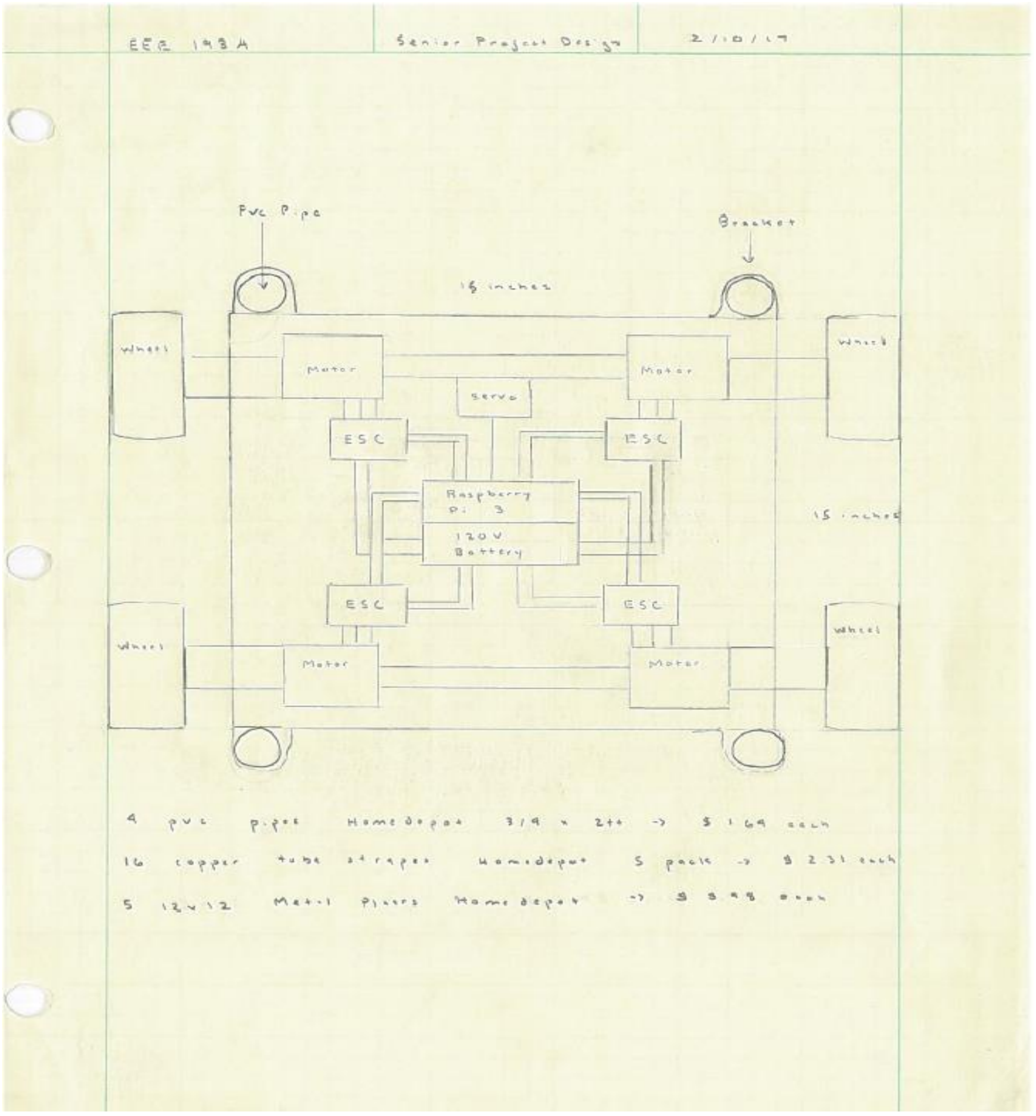
VII. Design Documentation

A. Physical Build

Early in the first semester, the physical build began with design drawings sketched to serve as blueprints to what the Autonomous Delivery System would look like. These drawings are pictured as Figures 8-10 below. As the semester progressed, some physical build features turned out like the drawings while other features did not. A trip to Home Depot was made to purchase six 1-1/4-inch x 48-inch zinc-plated punched angle iron rods with holes, and six 12-inch x 12-inch aluminum metal sheets. Four of the six rods were each cut to three feet using a Sawzall reciprocating saw with metal cutting blades. The freshly cut three-foot rods served as the corner posts of our system which met our desired system's height of three feet. The two other rods were each cut into one-foot rods. The freshly cut one-foot rods served as shelves of our system's chassis and levels which met our desired system's width of one foot. Four of the six sheets served as the chassis' base and levels to our system. The two other sheets were cut with a Plasma Cutter creating four metal sides to our chassis. A Metal Grinder Machine was used to smooth all the freshly cut edges. The chassis base was composed of two 8-inch Ribbed Wheels and two casters. Holes were cut in the metal sides of the chassis, so the motors could be secured to the chassis base and be connected to the wheels. Holes were also cut in the base of the chassis, so the casters could be drilled into the base.

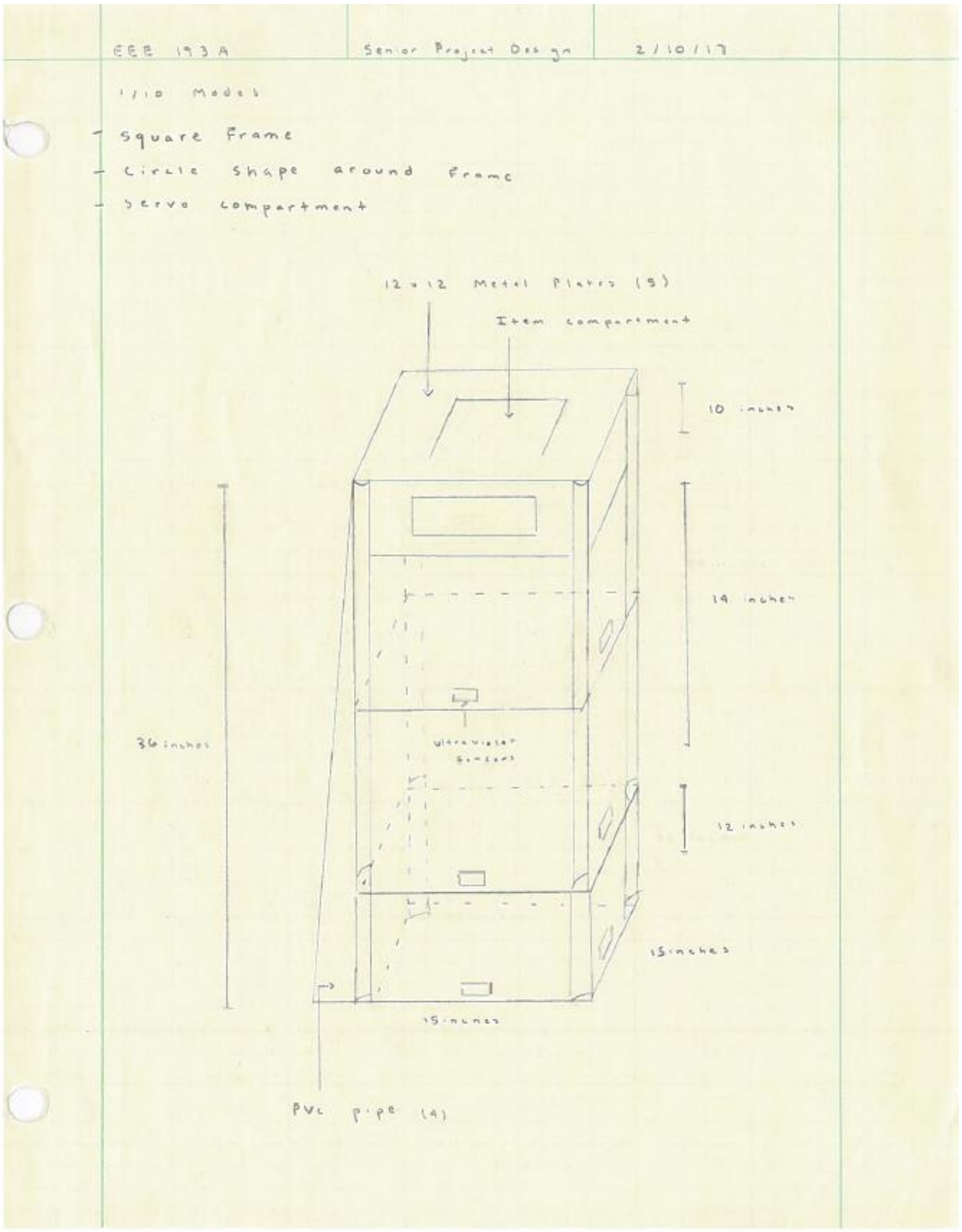
Midway through the first semester, nine 12-inch thin metal pieces to hold 15 ultrasonic sensors were added to the system. Three pieces were attached to the front plastic panel. These pieces required using a vice to implement a 30-degree bend on each side of the piece. These pieces held three ultrasonic sensors each. Additionally, three pieces were attached to left panel and three pieces were attached the right panel. These pieces held one ultrasonic sensor each. Throughout the system, diamond shaped drill bits were used to cut holes through plastic and metal. A Dremel rotary tool was used to fine tune the holes drilled into the plastic and metal.

Towards the end of the first semester, a shell was placed around the system. The shell was composed of four 36-inch x 12-inch durable plastic panels. Additionally, the chassis' base remained metal, but the three additional metal levels were replaced with 12-inch x 12-inch durable plastic sheets. The plastic replacements lightened the system's weight and allowed a viewer to see through the system. Nuts, washers, and bolts were also purchased and installed through the holes of the rods to connect the corner posts to the shelves, panels, and the ultrasonic sensor thin metal pieces. The final physical build of our Autonomous Delivery System is pictured in Figure 2 below.


EEE 193A Senior Project Design 2/10/17

4 pvc pipes Homedepot 3/4 x 2ft -> $ 1.64 each
16 copper tube strapes Homedepot 5 pack -> $ 2.31 each
5 12x12 Metal Plates Homedepot -> $ 5.98 each

Figure 2 — Initial Motor Chassis Layout


EEE 193 A | Senior Project Design | 2/10/17

1/10 Model

- square Frame
- circle shape around frame
- servo compartment

Figure 3 — Initial Autonomous Delivery System Mechanical Frame Drawing #1

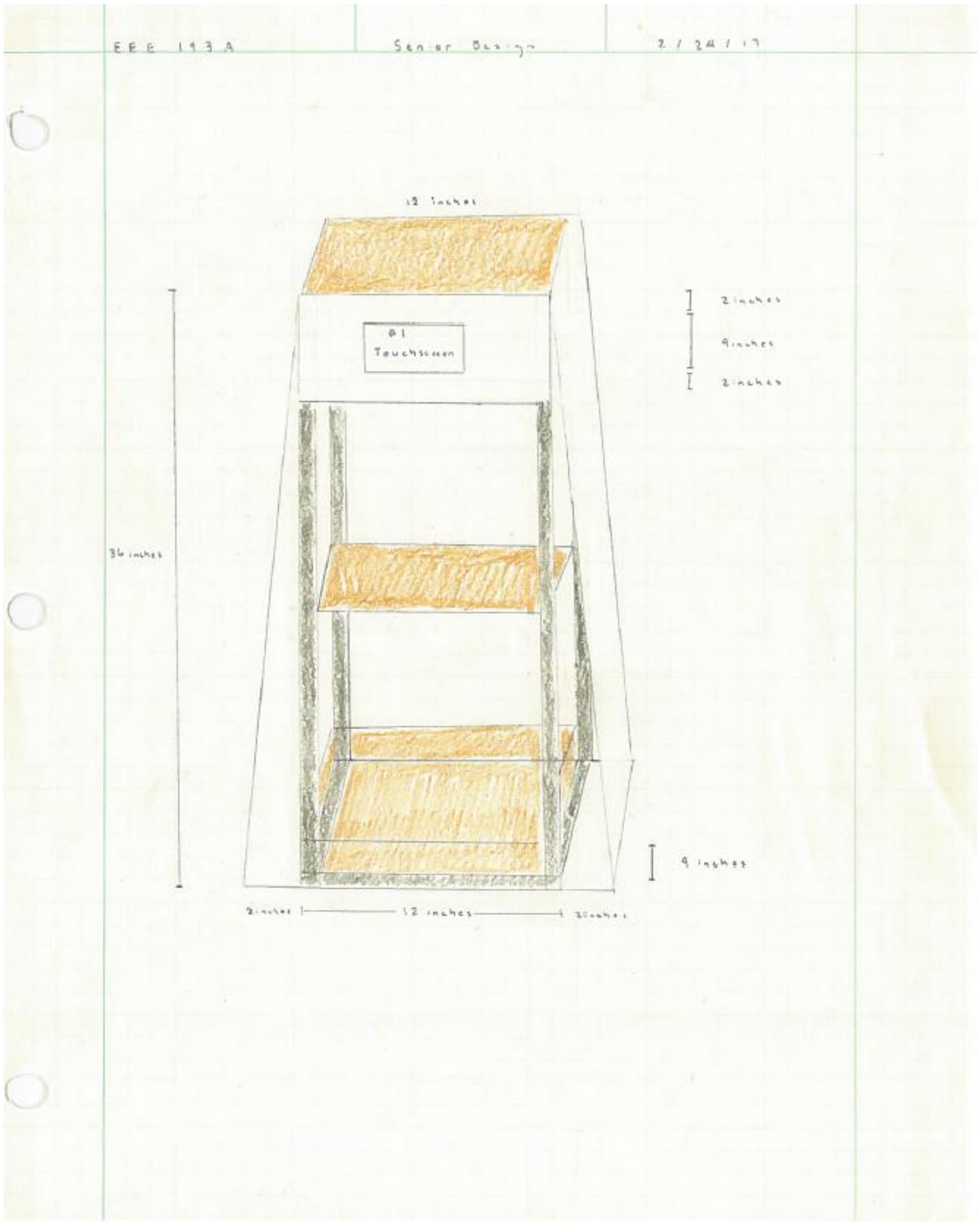


Figure 4 — Initial Autonomous Delivery System Mechanical Frame Drawing #2

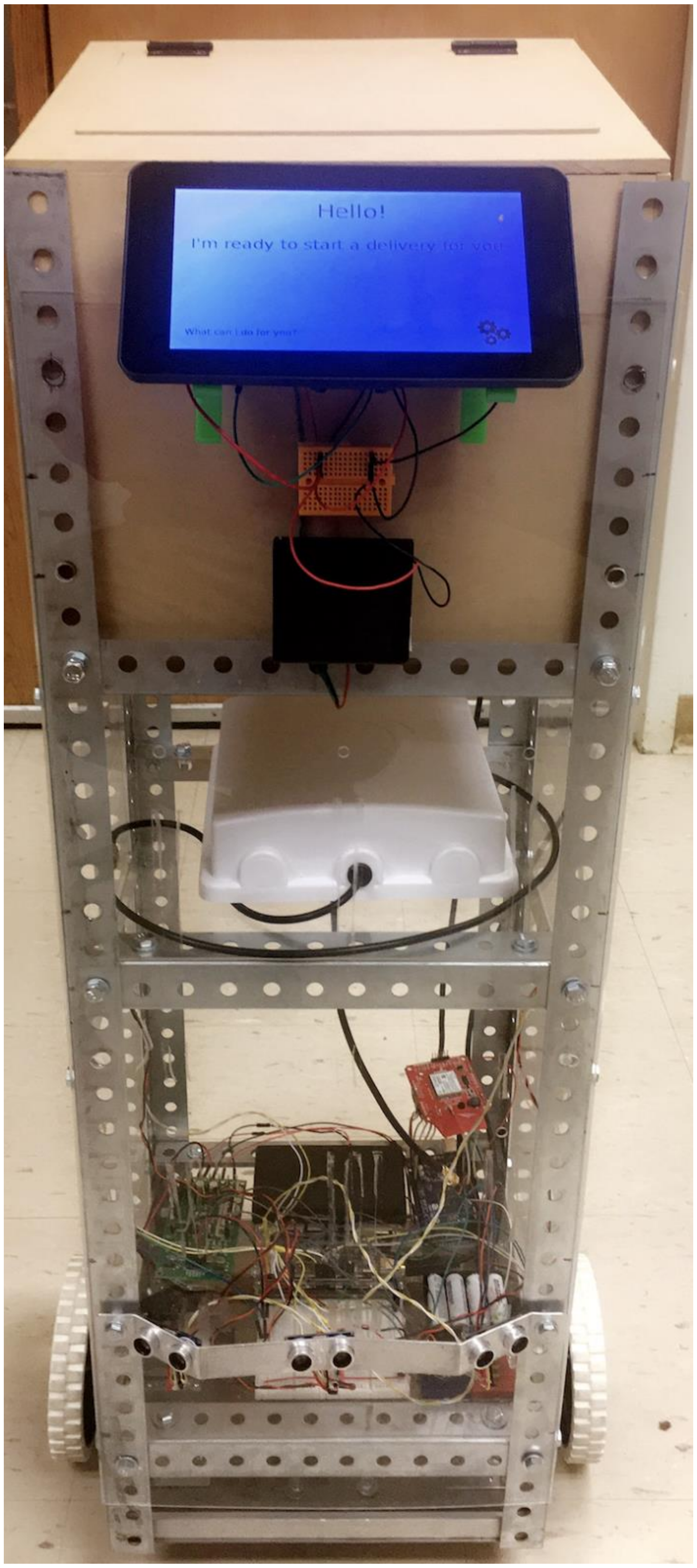


Figure 5 — Final Autonomous Delivery Robot Physical Build

Once the basic frame was finished the top compartment and GUI mounting became the primary goals of the physical build. The first step was building the top compartment. The top compartment is constructed of ¼ inch thick particle board with a pine veneer. The boards were cut to be about 1 square foot with a 10-degree angle from front to back on the top edge of the two side pieces so that the top face of the compartment would be angled towards the user. The box was held together with wood glue to ensure that nails would not poke through the box such as in Figure 6. The top of the box is built with a 9in by 10in hole for items to be placed inside, the actual size is about a half inch smaller in both length and width due to a lip placed around the inside for the lid to rest on. once the box was constructed and a hinge was added to the lid, a servo was placed towards the back of the box so that it could open the lid. The servo was placed towards the back so that the small range of motion that it had would be able to cause a profound change in the range of motion of the lid Figure 7. Once the box was finished it could be slotted into the top of the robot. The next step for the design was to mount the GUI onto the box and the front of the robot. The measurements of the GUI screen were taken using a caliper to ensure accuracy. once the position of the mounting holes was known a custom sketch was drawn up in SolidWorks to mount the screen to the box at a 45-degree angle Figure 8. Once the file was made it was 3D printed into mounts that could be used for the GUI. After the prints were finished it was simple to screw the brackets to the box and to the screen. Once the screen was attached to the box and the box was slotted into the top of the robot the physical build was finalized for the project.

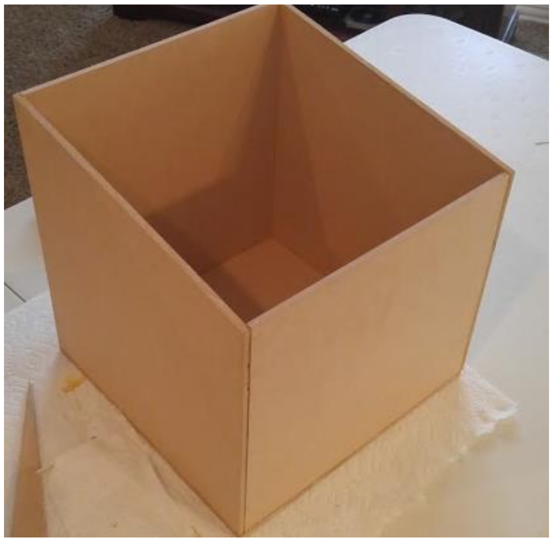


Figure 6 — Prototype Compartment Design

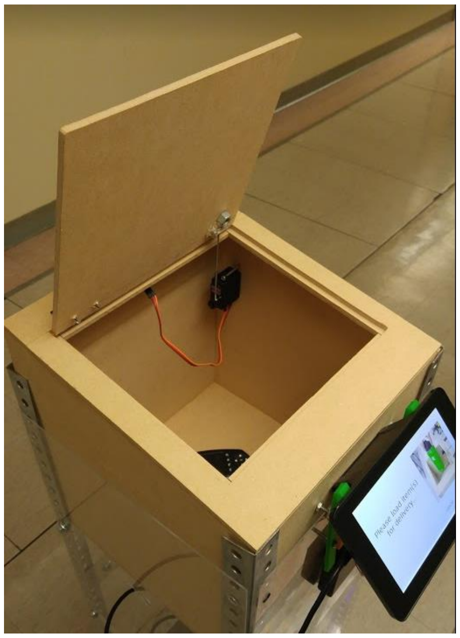


Figure 7 — Compartment Box and Mounted Screen

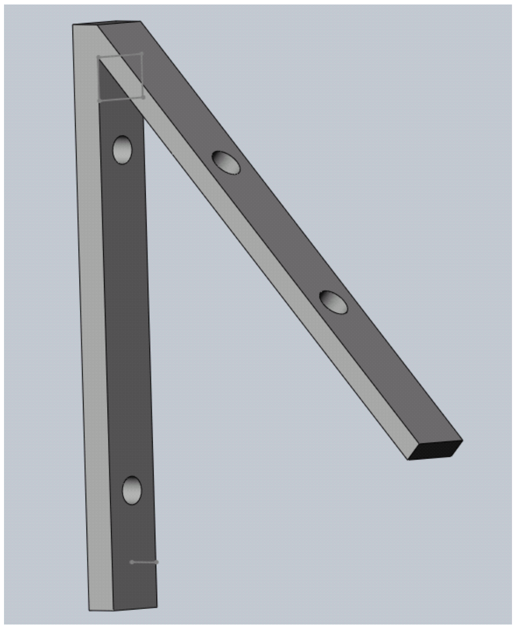


Figure 8 — Screen Mount Design

B. Motor Control

The two motors of the Autonomous Delivery Robot needed to be manipulated in a manner that would allow for smooth movement in a desired direction of travel and function efficiently under a maximum load of 50 pounds. Since the system is designed for delivery tasks within buildings, high speeds of travel were not desired. For this reason, two 131:1 Metal Gearmotor 37D x 73L mm with 64 CPR Encoders were chosen as the system’s motors. Operating at 12

V, these DC brushed motors have a stall torque rating of 250 oz-in and a stall current rating of 5 A. Under maximum load, it was found the motors can operate at speeds up to approximately 1.5 to 2 mph.

Figure 9 — Pololu 131:1 Metal Gearmotor

To manipulate the motors to travel in different directions and speeds, a dual H-Bridge circuit was needed. A simple H-Bridge circuit, such as the one seen below in Figure 10, consists of four transistors and acts as a current amplifier. The voltage sources V_{fwd} and V_{rev} are connected to pulse width modulation (PWM) pins on the Arduino Mega. An outputted signal from one of these pins while an outputted LOW (0 V) signal from the other pin determines the motor's direction of travel. The voltage source V_{CC} is used as the circuit's positive voltage rail and ground is used as the circuit's negative voltage rail. Referring to Figure 2, the desired direction of travel for this simulation is in the forward direction at a maximum PWM signal of 255, or 5 V. This signal flows through the transistor Q2, the motor (the center of the schematic where the 10.85 V bias reading is seen) and then to transistor Q3 where it will then flow to ground. If the desired direction of travel were switched to reverse, the signal would flow through transistor Q4, the motor, and then to transistor Q1. Though this schematic design does not account for current spikes or back emf voltages across the motor or PWM pins, it provides a basic understanding of how the motors can be controlled to travel in various directions and speeds. With the implementation of the UniqueGoods Dual H-Bridge DC Motor Driver PWM Module (Figure 11) connected to each motor, the system can be controlled via software to move forward, backwards, left, and right.

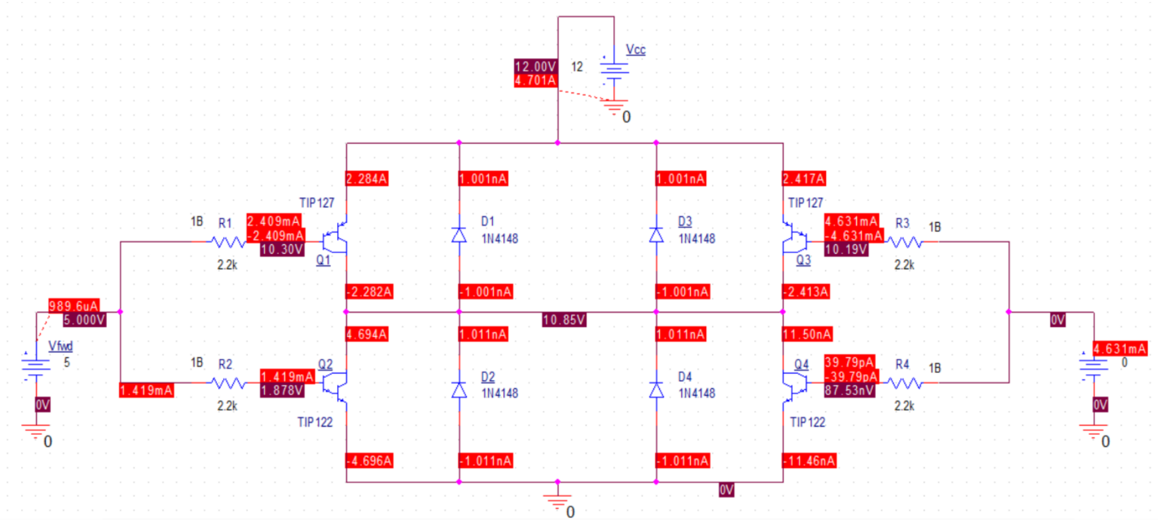


Figure 10 — Simple H-Bridge Circuit Schematic with Bias Readings

Figure 11 — UniqueGoods Dual H-Bridge DC Motor Driver

Attached to the Pololu 131:1 Metal Gearmotors are two-channel Hall-effect encoders. Hall-effect encoders can read the rotation of a magnetic disk attached to the motor's shaft, which can be decoded to find the distance travelled, speed of the motors, and robot orientation angle. To determine the distance travelled, the amount of degrees per count needed to be determined. For this system, a change in position only needed to be sensed across one of the encoder pins, which accounts for 16 counts/revolution. Taking the gear ratio of the motor into account, the total counts/revolution can be found using Equation (1). This can then be used to find the degrees per revolution using Equation (2).

$$\textit{Encoder counts/revolution} = (131.25)(16\ \textit{counts/revolution}) = 2{,}100\ \textit{counts/revolution} \qquad (1)$$

$$(360\ \textit{degrees}/1\ \textit{revolution})(1\ \textit{revolution}/2{,}100\ \textit{counts}) = 0.1714\ \textit{degrees/count} \qquad (2)$$

Once the total degrees/count was found the distance can be measured knowing the circumference of the wheel attached to the motor. For a wheel with a diameter of 6 inches, the distance can be found using Equation (3).

$$\textit{Distance [cm]} = ((\textit{Degrees turned})*(\textit{Wheel Circumference}))/360\ \textit{degrees} \qquad (3)$$

Using the found distance over a set time sample allows the speed of the system to be found, which is of utmost importance for control over the motors. Using a sampling time of a millisecond, the speed of each motor can be found.

$$\textit{Wheel Linear Velocity [cm/s]} = 1000*(P_{new} - P_{old})/(t_{new} - t_{old}) \qquad (4)$$

Using the measured linear speed of each motor allows for the robot orientation angle to be found. Once the robot's angular velocity is found, the robot's orientation angle can be found by integrating computationally and converting the value from radians to degrees. This can be seen in Equation 7 below, where Equation 5 and Equation 6 find the angular wheel velocity and robot angular velocity needed to determine the orientation angle.

$$\textit{Wheel Angular Velocity [rad/s]} = \textit{Speed/Wheel Radius} \qquad (5)$$

$$\textit{Robot Angular Velocity [rad/s]} = (\textit{Wheel Radius/Robot Diameter})*(U_{LEFT} - U_{RIGHT}) \qquad (6)$$

$$\textit{Robot Angle [deg/s]} = (\textit{Robot Angle} + \textit{Robot Angular Velocity})*T*(180/\pi) \qquad (7)$$

A P.I.D. controller to control the robot's orientation during straight-line movement was implemented. The purpose for P.I.D. implementation is to equalize the speed measured between the two motors and drive the system in a straight forward direction based on the robot's orientation angle. With the robot's initial angle at the start of forward movement at 0 degrees, and a reference angle value of 0 degrees, the encoders provide negative feedback to the controlled action. The past, present, and future errors in angle can be accounted for in each wheel to drive the system to a steady state despite physical disturbances, which can be seen in Figure 12.

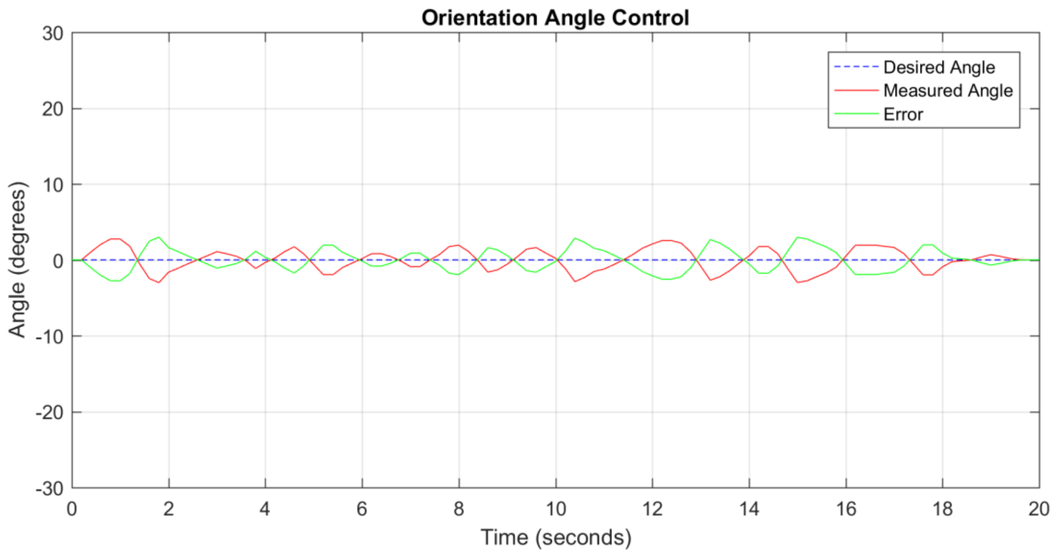


Figure 12 — Robot Orientation Angle Control Data Analysis

C. *Object Detection*

To detect objects in the system's immediate surroundings, HC-SR04 Ultrasonic Sensors were used (Figure 13). Each HC-SR04 sensor has a detection range of 2 – 400 cm and a viewing angle of 30 degrees, although measurements are most accurate within 15 degrees of the object's center point.

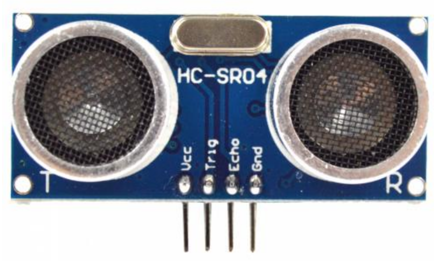


Figure 13 — HC-SR04 Ultrasonic Sensor

The ultrasonic sensors output an 8-cycle burst of a 40-kHz sound, which then bounces back towards the device if it hits an object. By determining half of the travel time of the sound in reference to the speed of sound, the distance can be determined in Equation (8).

$$\textit{Distance [cm]} = \textit{Sound}_{time} * 0.034/2 \qquad (8)$$

Since the viewing angle of each sensor is minimal and objects can only be detected along a single viewing plane, an array of ultrasonic sensors are needed (Figure 14). Including additional sensors along a single horizontal plane allows for a wider range of sight for the system. To account for forward-facing objects and wall detection on either side of the system, a total of five vertical planes of a single sensor on each are needed to obtain an accurate description of the system's immediate surroundings (Figure 16).

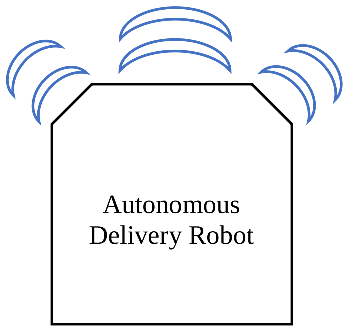


Figure 14 — Single Horizontal Plane Field of Sight

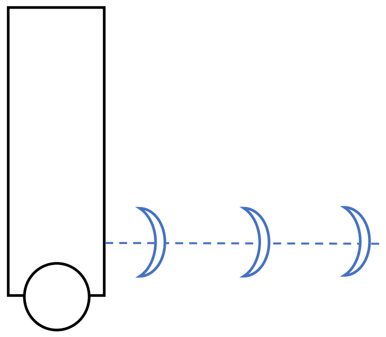


Figure 15 — Single Horizontal Plane of Sight, Side View

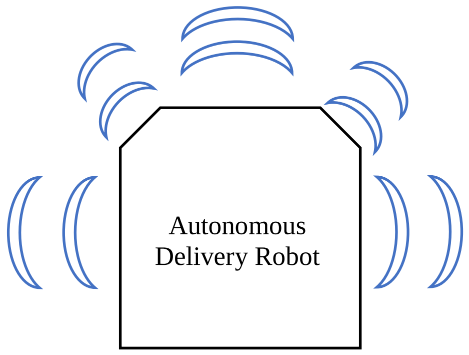


Figure 16 — Forward and Side Complete Plane of Sight

With all vertical planes of sight covered, distance measurements can be recorded. To account for any inaccurate readings produced from the ultrasonic sensors, three samples are collected from each sensor. From these three samples, the maximum distance is found. The reason for this is for object or wall avoidance implementation, where what matters is what is closest to the system, so it can avoid the object or wall accordingly. To obtain more accurate distance readings, a bigger sample can be taken from each sensor. However, this was not implemented into the system because it slowed down the overall functionality of the system due to the lack of multiple cores in the Arduino Mega and inability to easily access the processor's interrupt service routines.

D. Object Avoidance

Object avoidance is integrated into the Autonomous Delivery Robot in two different manners; wall following and forward object avoidance. Using the side-mounted ultrasonic sensors the robot can adjust its angular orientation to maintain a reasonable distance from the wall. For this system, the desired range of travel away from the wall was referenced to be 40 cm. Wall following only occurs briefly during the correction process after an object is avoided. It then returns angle orientation control.

Using the forward-mounted ultrasonic sensors, oncoming objects can be avoided as well. For this project, all objects are treated as static. The reason for this is the ultrasonic sensors are of too low of a quality to provide accurate data to detect object velocity and direction. Moving on, objects can be detected along all front-facing vertical planes of sight, two vertical planes of sight, or a single vertical plane of sight. For each of these varying situations a different protocol needs to be introduced. Each protocol is based off a thresholding P.I. controller that increases/decreases the speed of each motor based on the position of the detected object(s).

To avoid static objects, the object in question needs to be assumed that it will remain static for the entirety of the time the system detects it because it is not capable of knowing what the object truly is. For this reason, the object(s) must be avoided. To trigger the system to begin avoiding static objects, the object in question needs to be detected at distance smaller than the set threshold distance, which is 60 cm. Once a detected distance is less than the set threshold of one of the planes the object was detected in, the corresponding P.I. controller is activated. The P.I. controller adjusts the speeds of the motors to turn away from the detected object at a speed calculated based off the error between the threshold distance and measured distance of the object. As the Autonomous Delivery Robot approaches the object, the speed of rotation (either left or right) to avoid the object increases. Once the object is avoided, the system will begin correcting for its orientation angle. To return to the desired orientation angle of 0 degrees, the PWM values of the two motors are offset to turn along a curved path until the measured angle is once again 0 degrees. This can be seen below in Figure 17.

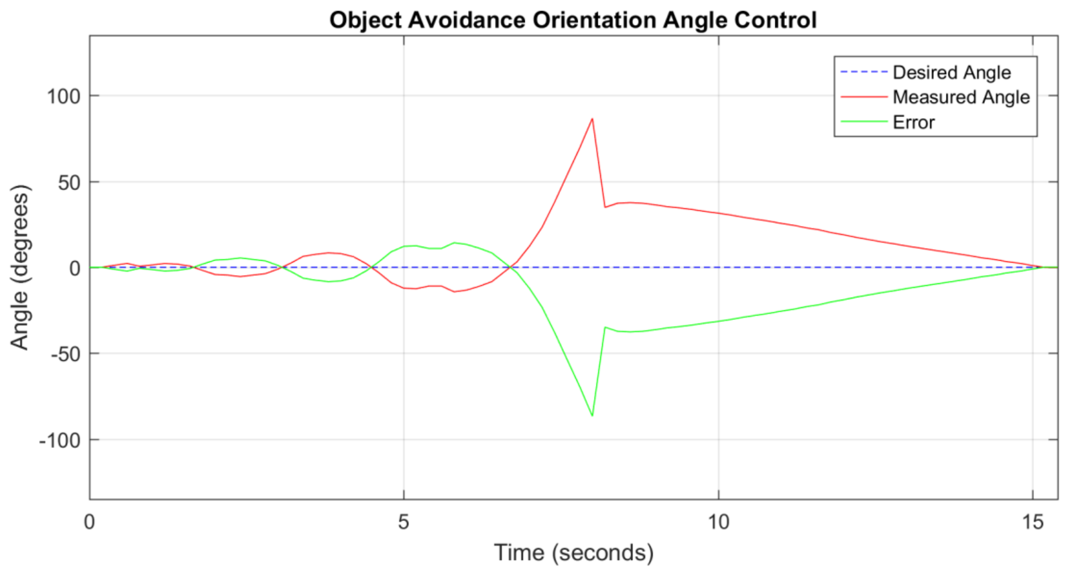


Figure 17 — Object Avoidance Orientation Angle Control Data Analysis

After the robot's orientation angle is corrected for, the robot is alongside the object. The side ultrasonic sensors are then activated, notifying the robot to move forward in a straight path until the object is cleared. Upon clearing the object, the robot will return to the desired distance to the wall, 40 cm, by following the hypotenuse of a 30-60-90 triangle. If an object is detected in the center vertical plane, the center-left and center-right vertical plane minimum distances are checked. Whichever distance is of greater value, the system will avoid the object in that direction.

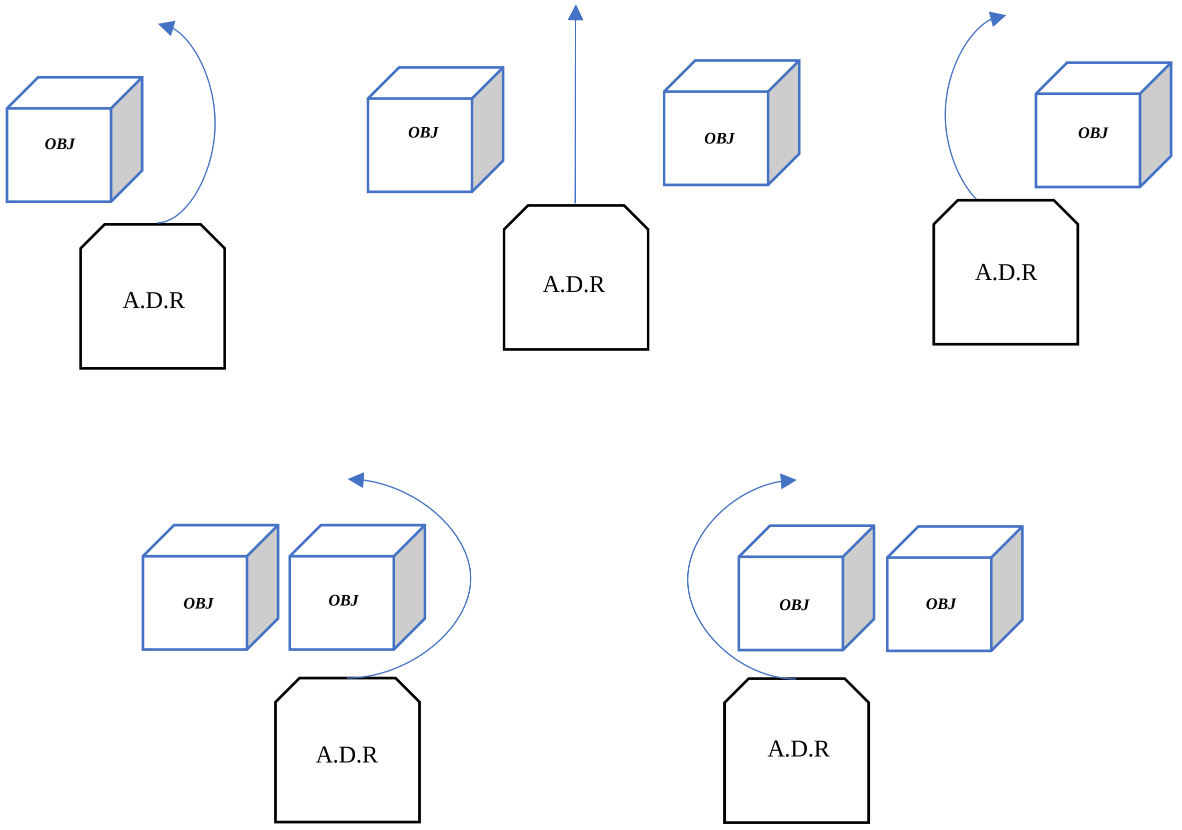


Figure 18 —Various Object Avoidance Protocols

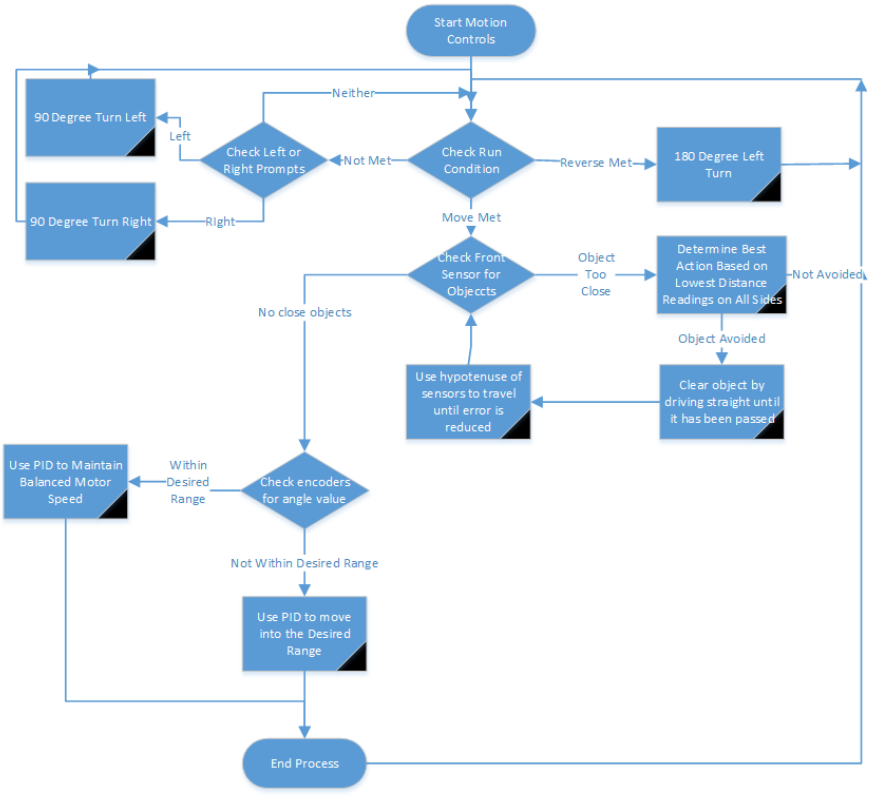

Start Motion Controls
90 Degree Turn Left
Neither
Left
Check Left or Right Prompts
Not Met
Check Run Condition
Reverse Met
180 Degree Left Turn
90 Degree Turn Right
Right
Move Met
Check Front Sensor for Objeccts
Object Too Close
Determine Best Action Based on Lowest Distance Readings on All Sides
Not Avoided
No close objects
Object Avoided
Use hypotenuse of sensors to travel until error is reduced
Clear object by driving straight until it has been passed
Use PID to Maintain Balanced Motor Speed
Within Desired Range
Check encoders for angle value
Not Within Desired Range
Use PID to move into the Desired Range
End Process

Figure 19 —Motor Control Flowchart

E. Room Recognition

This system uses a RFID reader to identify tags placed near each room to identify when the defined destination has been reached. This portion of the project is run on an Arduino mega along with a SparkFun Simultaneous RFID Reader - M6E Nano. The RFID reader is attached to the Arduino mega with wires connecting power, ground, RX, and TX. This RFID mega is connected to Motor Control mega which is in turn connected to the GUI. The GUI didn't like the soft serial communication used to communicate to a RFID, so the motor acts a relay since it already needed to receive information telling it when to move, turn and stop at the room.

The only thing needed to use the RFID was an Arduino library, but the antenna caused a bit of an issue. It required a soldered connection to be unsoldered and a new connection to be made. There was also an issue with soft serial not working with certain pins on the Mega, so it had to be picked from a select few.

To use the RFID tags, each one had to initially be scanned and given a label along with the direction it was in relation to the robot's path from home. It also needed to be specified whether it would be to the left or right of home so that the robot knew which way to start its delivery. While out on a delivery, the RFID reader will continuously scan for rooms and analyze if any corrections needed to be made in its course such as if it went too far off if it had reached its destination. The positions of these tags can be anywhere within an 8-ft. radius as long the metal strip is facing towards the RFID reader, it can even sense tags through objects such as the wooden box that sits above it.

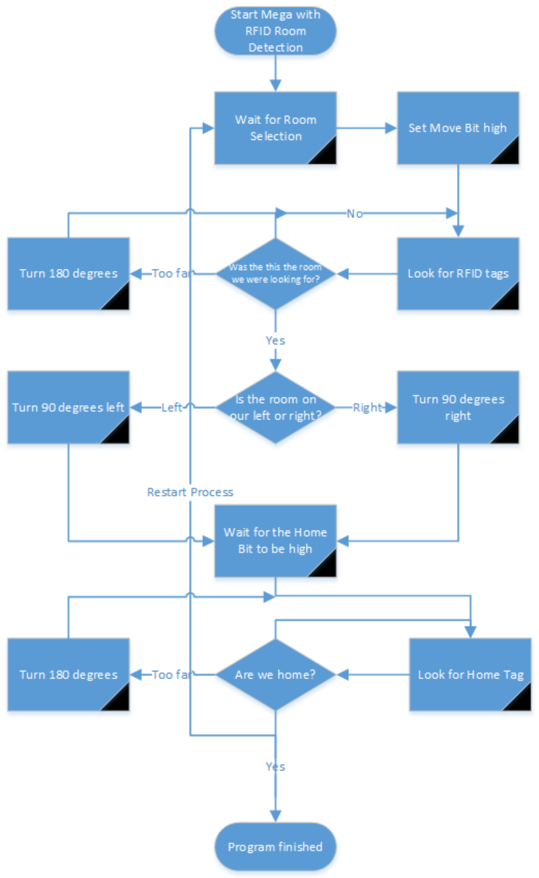


Figure 20—Room Recognition Flowchart

E. Graphical User Interface

The GUI is created using a computer program called NetBeans. NetBeans is a software development platform written in Java. In NetBeans, ten extremely user-friendly screens were incorporated into the GUI. Each screen is stored in a card layout format. Each screen must be set to public and static, so the GUI can access it. Each screen has several "JLabels." Many of these JLabels are coded to perform an event when they are pressed and released. In every screen, a JLabel is placed over the entire screen to include a blue and white background simply for design purposes.

The first screen is called the Start Screen. This screen is composed of four "JLabels". The first JLabel is the *helloLabel* which displays, "Hello!" The second JLabel is the *deliverLabel* which displays, "I'm ready to start a delivery for you!" If this label is pressed, the GUI will transition to the Entry Screen. The third JLabel is the *whatLabel* which displays, "What can I do for you?" If this label is pressed, the GUI will transition to the Details Screen. The fourth JLabel is the *settingsLabel* which displays a gears icon. If this label is pressed, the GUI will transition to the Settings Screen.

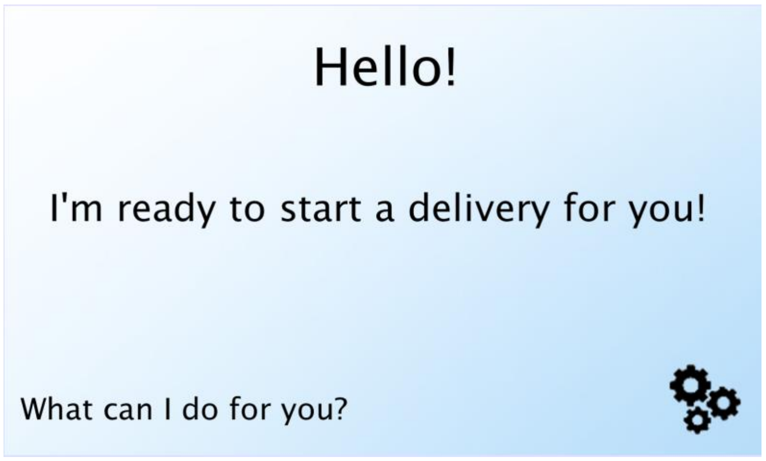
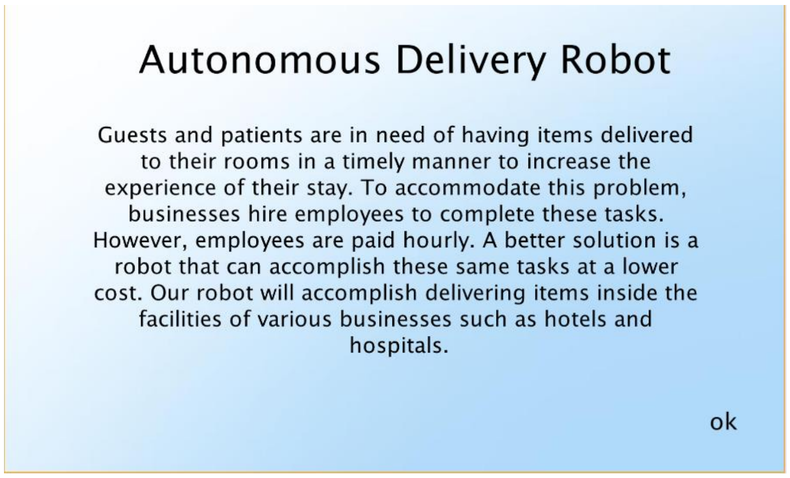


Figure 21 — Start Screen

The second screen is called the Details Screens. This screen is simply to give information to the user about the primary purpose of the robot. This screen is composed of three "JLabels". The first JLabel is the *detailsLabel* which displays, "Autonomous Delivery Label". The second JLabel is the *infoLabel* which displays the purpose. The third JLabel is the *okLabel* which displays, "ok". If this label is pressed, the GUI will transition back to the Start Screen.



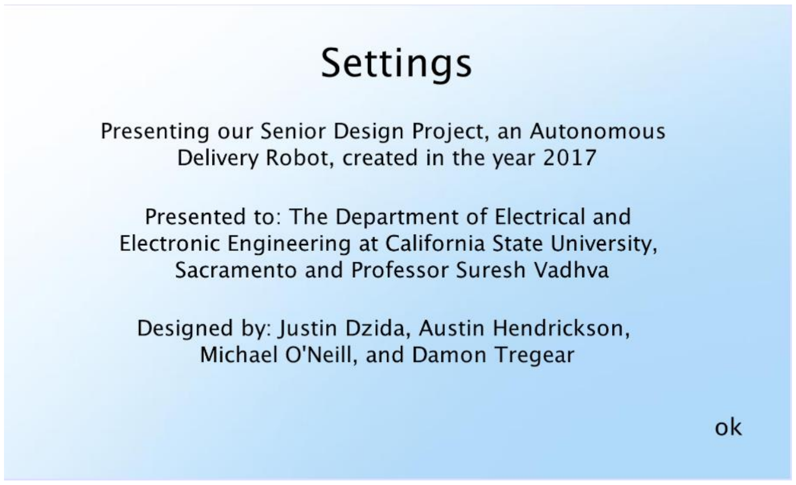
Autonomous Delivery Robot

Guests and patients are in need of having items delivered to their rooms in a timely manner to increase the experience of their stay. To accommodate this problem, businesses hire employees to complete these tasks. However, employees are paid hourly. A better solution is a robot that can accomplish these same tasks at a lower cost. Our robot will accomplish delivering items inside the facilities of various businesses such as hotels and hospitals.

ok

Figure 22 — Details Screen

The third screen is called the Setting Screens. This screen is simply to give information to the user about who it was presented to, when is was created, and who designed the robot. This screen is composed of three “JLabels”. The first JLabel is the detailsLabel which displays, “Settings”. The second JLabel is the infoLabel which displays the information. The third JLabel is the okLabel which displays, “ok”. If this label is pressed, the GUI will transition back to the Start Screen.



Settings

Presenting our Senior Design Project, an Autonomous Delivery Robot, created in the year 2017

Presented to: The Department of Electrical and Electronic Engineering at California State University, Sacramento and Professor Suresh Vadhva

Designed by: Justin Dzida, Austin Hendrickson, Michael O'Neill, and Damon Tregear

ok

Figure 23 — Settings Screen

The fourth screen is the Entry Screen. The entry process requires the operator or desk attendant to enter the recipient's phone number and room number. The recipient’s ten-digit phone number will be entered followed by their four-digit room number. JLabels are needed to display each number (0-9), cancel, back, delete, and enter. Once a number is pressed and released it will be displayed as a string in a JLabel called the *answerLabel*. Once the recipients ten-digit phone number is entered and the enter label is pressed and released, the GUI will transition to the room number screen. Once the recipients four-digit room number is entered and the enter label is pressed and released, the GUI will transition to Load Item Screen.

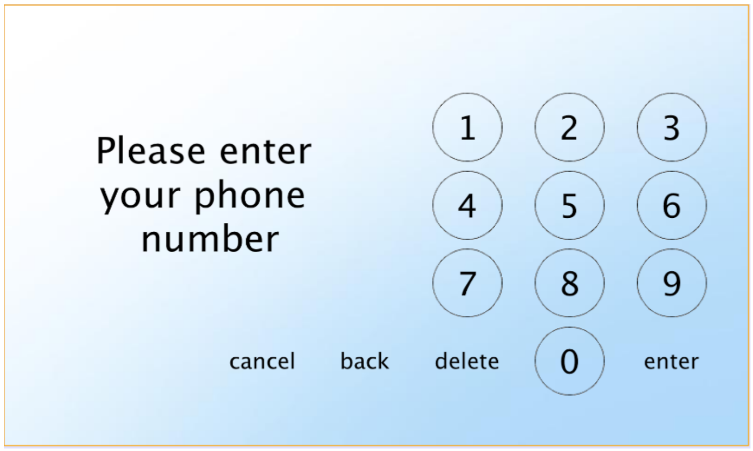


Figure 24a — Entry Screen (Phone Number)

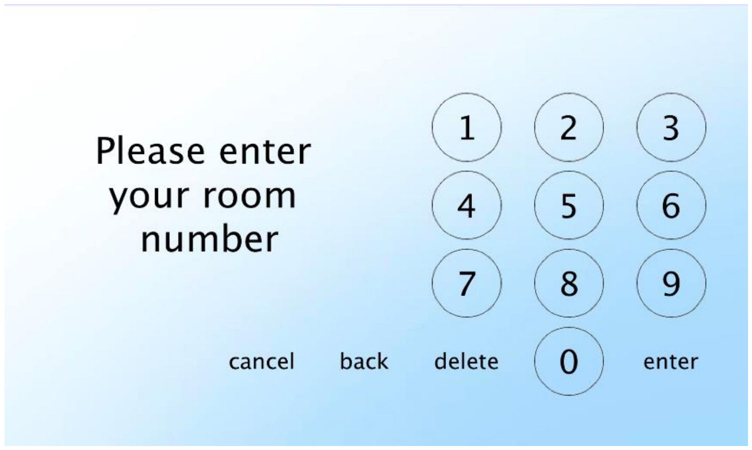


Figure 24b — Entry Screen (Room Number)

The fifth screen is the Load Item Screen. Once the recipients four-digit room number is entered and the enter label is pressed and released, the GUI will transition to Load Item Screen and simultaneously open a compartment box using a servo motor. The load item process requires the operator or desk attendant to load an item into the compartment box located on top of the robot. Once the item is loaded and the Go label is pressed and released, the Autonomous Delivery Robot will begin its delivery process and a SMS message will be sent alerting the recipient that their item is on the way. The SMS message will also send the recipient a four-digit passcode to open the compartment box when their item has arrived. To incorporate this texting service into our GUI, we downloaded and installed a library called Bulk_SMS_Library.jar into NetBeans. JLabels are needed in the GUI to display the load item picture, cancel, back, and Go. Once the Go label is pressed and released, the GUI will transition to the Delivery Screen.

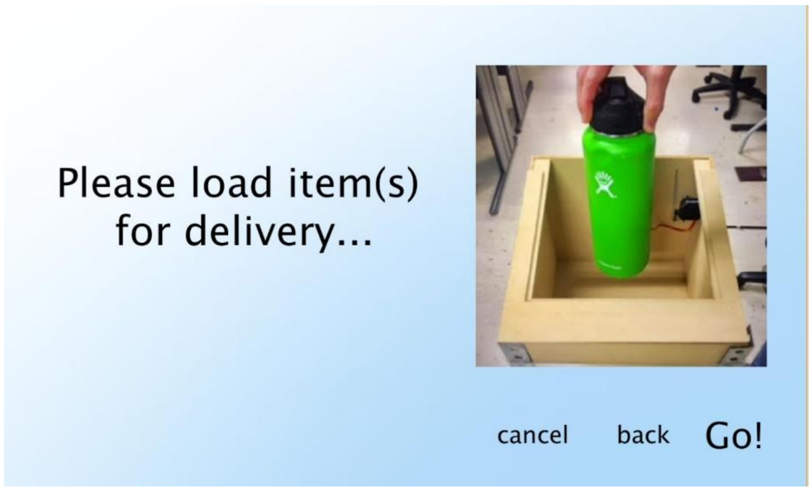
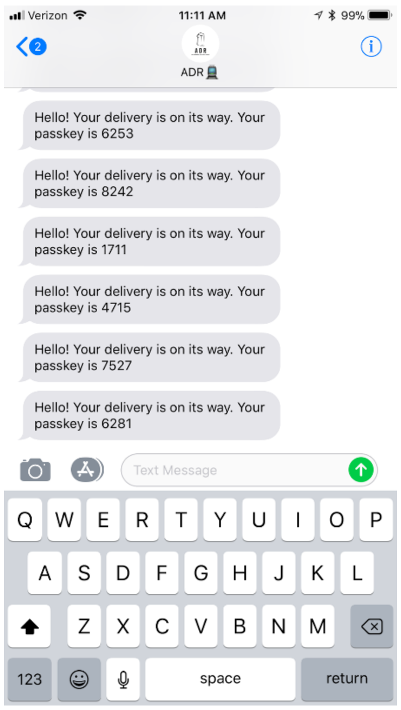


Figure 25a — Load Item Screen



Hello! Your delivery is on its way. Your passkey is 6253

Hello! Your delivery is on its way. Your passkey is 8242

Hello! Your delivery is on its way. Your passkey is 1711

Hello! Your delivery is on its way. Your passkey is 4715

Hello! Your delivery is on its way. Your passkey is 7527

Hello! Your delivery is on its way. Your passkey is 6281

Figure 25b — Delivery SMS Message

The sixth screen is the Delivery Screen. This screen will be displayed from the time the Go Label is pressed and released to the time the Autonomous Delivery Robot reaches the recipient's room. Throughout the delivery process, a Radio-frequency identification (RFID) sensor will be constantly searching the surrounding area for RFID tags preplaced on the doors of the rooms. A JLabel was needed to display the message, “I am currently making a delivery.” Once the robot reaches the recipient’s room the RFID will communicate to the GUI to transition to the Arrival Screen.

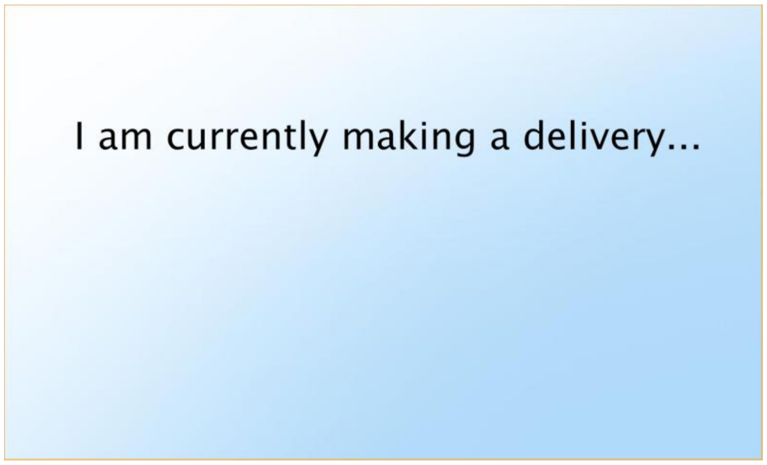
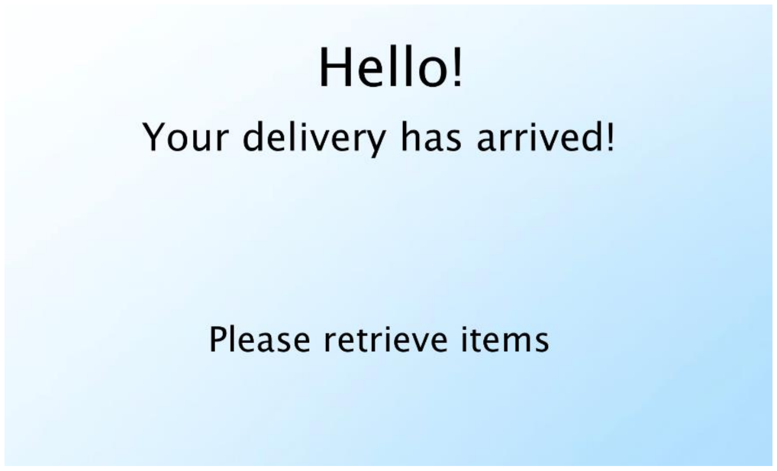

I am currently making a delivery...

Figure 26 — Delivery Screen

The seventh screen is the Arrival Screen. The purpose of the Arrival Screen is to greet the user that their requested item has arrived. This is the first screen the recipient will see when the recipient opens the door to retrieve their item. This screen is composed of three “JLabels”. The first JLabel is the *helloLabel* which displays, “Hello”. The second JLabel is the *deliverLabel* which displays, “Here is your delivery”. The third JLabel is the *itemsLabel* which displays, “Please retrieve your items”. Once this label is pressed and released, the GUI will transition to the Passcode Screen.


Hello!

Your delivery has arrived!

Please retrieve items

Figure 27 — Arrival Screen

The eighth screen is the Passcode Screen. The Passcode Screen requires the recipient to enter the four-digit passcode that they received via SMS message on their cell phone. JLabels are needed to display each number (0-9), delete, and enter. Once a number is pressed and released, it will be displayed as a string in a JLabel called the *answerLabel*. Once the recipient enters the four-digit passcode and the enter label is pressed and released, the compartment box will open the recipient will retrieved their item. Once the recipient has confirmed their items are retrieved, the GUI will transition to the Rating Screen.

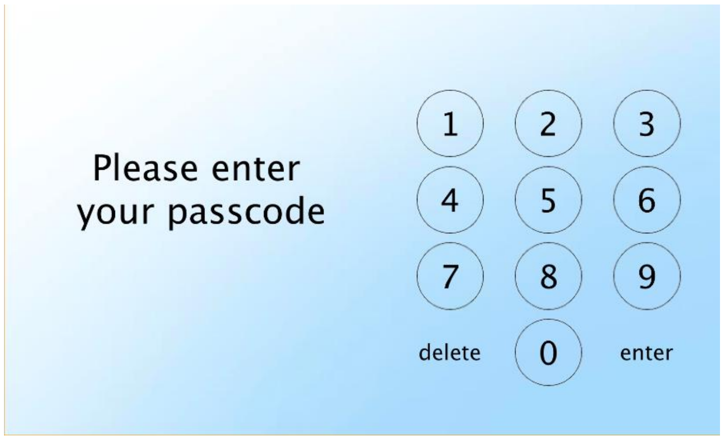


Figure 28 — Package Arrival Slide

The ninth screen is the Rating Screen. The Rating Screen offer the chance for the recipient to provide simple feedback to the creators about the delivery experience. The recipient can rate their experience on a scale of 1 to 5 stars. JLabels are needed to display each number (1-5). Once any number is pressed and released, the GUI will transition to the Return Screen.

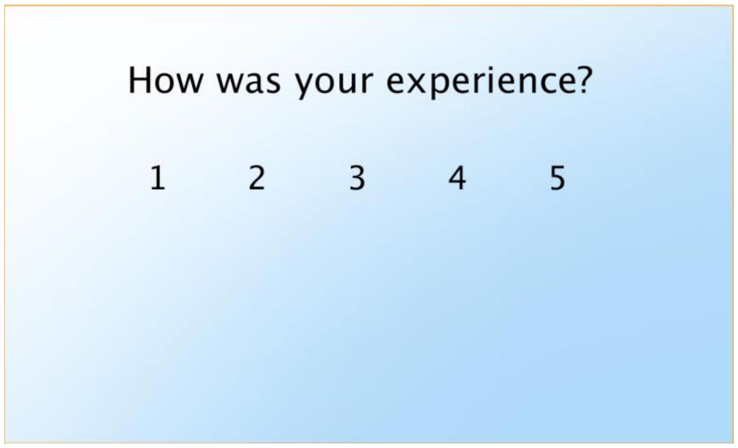
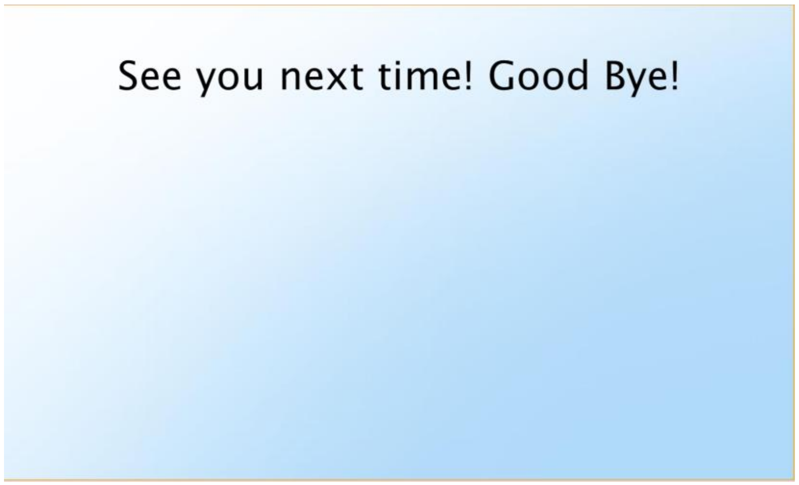


Figure 29 — Rating Screen

The tenth slide is the Return Screen. The Return Screen says goodbye to recipient and returns to the home base. The robot will turn 90 degrees from the door and begin traveling in direction where it came from. Once again, the RFID sensor will constantly search the surrounding area for the home base RFID tag preplaced on the wall where it started. Once the home base is identified by the RFID, the robot will stop, and the GUI will transition back to the Start Screen for another delivery.

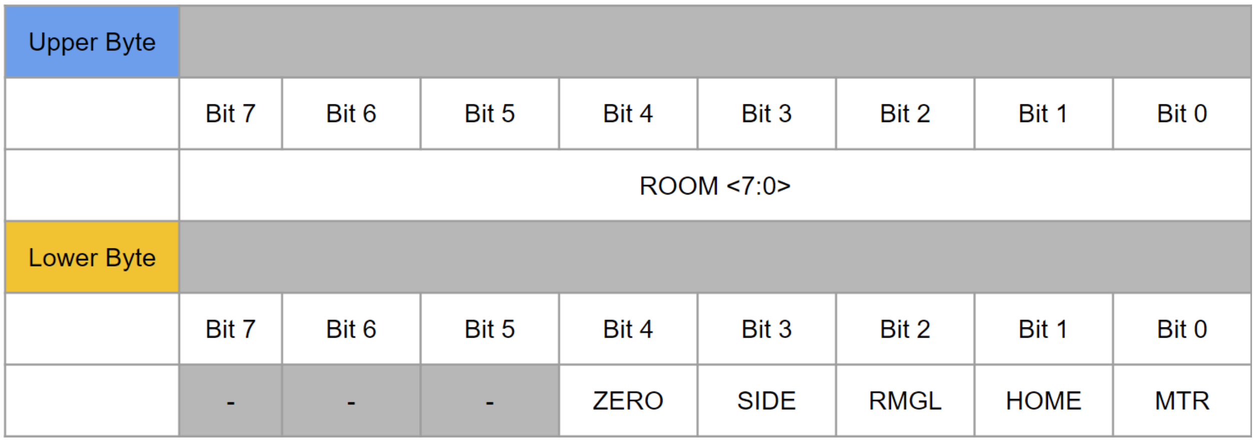


See you next time! Good Bye!

Figure 30 — Return Screen

G. Subsystem Communication

Integration of the Graphical User Interface, RFID Room Recognition, and Motor Control subsystems was done using multiple Universal Asynchronous Receive/Transmit (UART) channels. Both the main Arduino Mega and the RFID Arduino Mega used two UART channels. On the main Arduino Mega Channel 1 links the GUI to the main Mega, while Channel 2 links the RFID Arduino Mega. On the RFID Arduino Mega Channel 1 links the M6E Nano RFID Reader to the RFID Arduino Mega, while Channel 2 links up to the main Arduino Mega. This allows for each subsystem to properly communicate with one another based on the current stage of the delivery process. All UART channels are set to operate at a baud rate of 115.2 kbps with 8 data bits, 1 stop bit, and no parity bits. To simplify the communication process and ensure quick computation time, all data was sent in two bytes. This can be seen below in Figure 31.



Upper Byte								
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	ROOM <7:0>							
Lower Byte								
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	-	-	-	ZERO	SIDE	RMGL	HOME	MTR

Figure 31 — Subsystem Communication Data Bytes

The upper byte allocates all 8 bits for the room number information. This allows for a total of 255 rooms to be integrated into the system. The lower byte allocates individual bits to various intrinsic data that is required for the delivery algorithm.

- Upper Byte
  - Bit 7 – 0: Room number bits

- Lower Byte
    - Bit 7: Unimplemented – Read as 0
    - Bit 6: Unimplemented – Read as 0
    - Bit 5: Unimplemented – Read as 0
    - Bit 4: ZERO – Informs direction of zero radius turn to perform
        - 0 – Left
        - 1 – Right
    - Bit 3: SIDE – Side of hallway that desired room is on
        - 0 – Left
        - 1 – Right
    - Bit 2: RMGL – Informs if the robot has reached the desired room
        - 0 – No
        - 1 – Yes
    - Bit 1: HOME – Informs if the robot is at the home base
        - 0 – No
        - 1 – Yes
    - Bit 0: MTR – Informs whether the motors should be on/off
        - 0 – No
        - 1 – Yes

During the first stage of the delivery process, the user interface will transmit the room number to the various subsystems; this sets the upper byte. Depending on the room location, the RFID Arduino Mega will either clear or set individual bits of the lower byte to specify the direction of travel and the side of the hallway the room is on with reference to the robot. This ensures that the robot will travel in the desired direction during the second stage of the delivery process. Once the robot reaches the desired room, the room number is reset to an unused value, 0x20, for the home base location. Upon the completion of delivery, the third stage, the individual bits of the lower byte will either be cleared or set based on the location of home base. The robot will then move into the fourth and final stage of the delivery process, which is travelling back to home base. Upon arrival, both the upper and lower bytes are reset to 0x00. This allows us to refresh the system and user interface, and prepare it for the next delivery request.

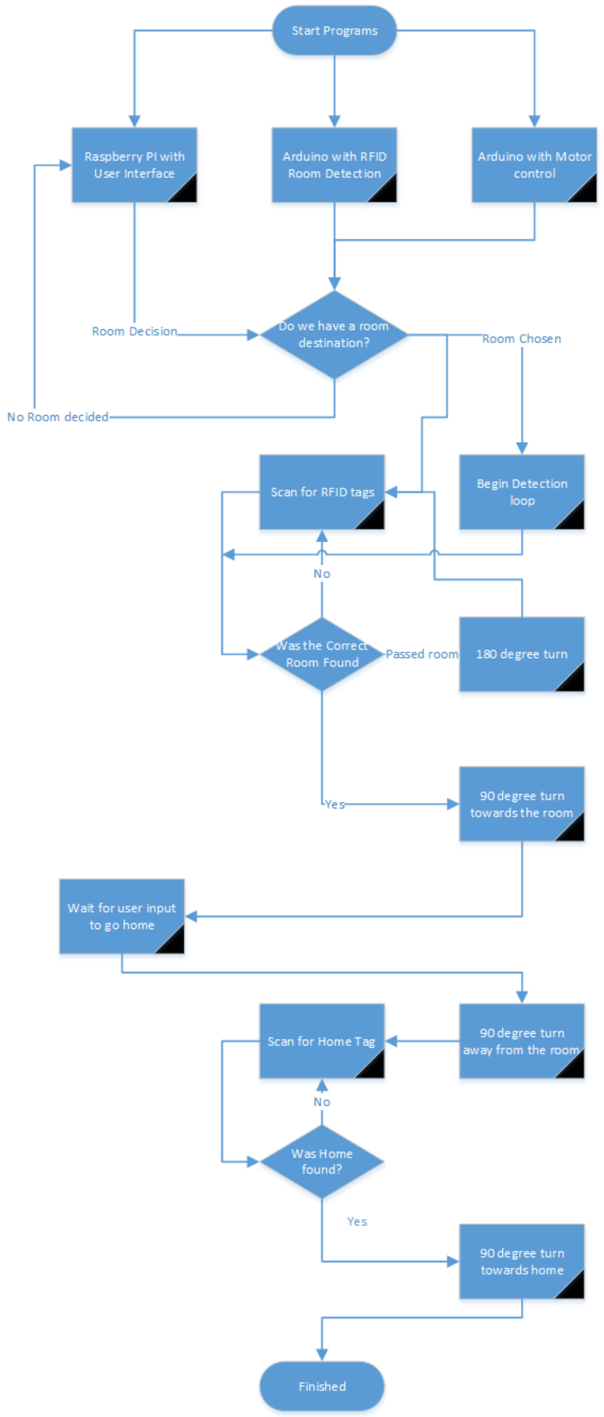

Start Programs
Raspberry PI with User Interface
Arduino with RFID Room Detection
Arduino with Motor control
Room Decision
Do we have a room destination?
Room Chosen
No Room decided
Scan for RFID tags
Begin Detection loop
No
Was the Correct Room Found
Passed room
180 degree turn
Yes
90 degree turn towards the room
Wait for user input to go home
Scan for Home Tag
90 degree turn away from the room
No
Was Home found?
Yes
90 degree turn towards home
Finished

Figure 32—Hardware Layout Flowchart

VIII. Integration Plans

As discussed in the Introduction, the Autonomous Delivery Robot has the potential to be integrated into any business model imaginable. Whether it be for hospitality, commercial, or mailing services, the Autonomous Delivery Robot can be tailored to suit any company's needs. In addition, a new model could implement the following upgrades:

- SLAM (Simultaneous Localization and Mapping) or SURF (Speeded Up Robust Features) Mapping
    - This feature will see to the replacement of the QR Room Recognition system. With SLAM or SURF Mapping, any controlled environment can be mapped out using various feature detectors and shape descriptors to create reference points within the environment. This would allow the system to recall various locations from memory and consistently compare its current location to these reference points to establish where the system is and where it needs to go.
- Machine Vision
    - This feature too would see to the replacement of the RFID Room Recognition system. Using machine vision, the system could use cameras to detect objects, rooms via number detection, and its location in the immediate environment. This would allow the robot to move more freely, but in a controlled, appropriate manner where it reacts to the changes in the environment.
- Multi-Container Storage for Multiple Deliveries
    - Instead of having the Autonomous Delivery System return to its starting position after each individual delivery, a multi-container storage system may be implemented to improve product efficiency. This installment would allow the system to make multiple deliveries before returning to its starting position, which would significantly improve the amount of deliveries capable of being made each day.

IX. Accomplishments

A. Justin Dzida

- Designed/built the Item Compartment
- Assisted in RFID research, setup, and testing
- Designed Raspberry Pi Screen mounting brackets
- Created servo functions for proper Item Compartment lid operation
- Assisted with subsystem testing and integration

B. Austin Hendrickson

- Testing and debugging PIDs, Ultrasonic sensors and motor functions
- Created program to detect and change functions depending on the different RFID tags detected
- Explored different options for controlling motors to correct overshoot and avoid objects without losing orientation

C. Michael O'Neill

- Physical Build Mechanical Design Drawings
- Assisted Physical Build Chassis
- Assisted Physical Build Frame
- Assisted Physical Build Shell
- Assisted Physical Build Compartment Box
- Installed Netbeans GUI Program on Raspberry Pi 3
- Installed SMS Bulk Texting File into Netbeans

- Netbeans GUI Starts Screen Design
- Netbeans GUI Details Screen Design
- Netbeans GUI Settings Screen Design
- Netbeans GUI Entry Screen Design
- Netbeans GUI Phone Screen Design
- Netbeans GUI Room Screen Design
- Netbeans GUI Load Item Screen Design
- Netbeans GUI Delivery Screen Design
- Netbeans GUI Arrival Screen Design
- Netbeans GUI Passcode Screen Design
- Netbeans GUI Rating Screen Design
- Netbeans GUI Return Screen Design
- Assisted with Overall Testing of Robot

D. Damon Tregear

- Encoder Functionality
- Robot Orientation P.I.D. Control
- Motor Zero Radius Turning
- Distance Detection of Wall Ultrasonic Sensors
- Assisted Determining Parameters for Object Avoidance
- Object Avoidance Motor P.I. Control
- Object Avoidance Orientation Correction
- UART Subsystem Communication

X. Summary

Working on improvements to this system with a little more experience under our belts was a valuable process. Going into this semester we had the physical robot mostly constructed and just needed to finalize the build, and improve upon the GUI, movement, communication, and navigation. As stated previously, this system can potentially be implemented into any business model and provide excellent assistance to humans at a low cost. As always improvements are possible, but the system is currently able to complete delivery tasks and navigation. In addition to previous functions the system now includes RFID Room Recognition, SMS Security Messages, and improved object avoidance, communication, user interface, and navigation. The process began with small scale testing to ensure that the necessary components purchased for large scale would work properly. Issues with the project were less drastic than with the prototype build, mainly consisting of random errors such as faulty PWM pins that only took a day or two to solve. Developing from the original QR room recognition, RFID room recognition replaced the camera to allow detection from longer range and at different orientations making for more reliable reads. The GUI was improved and switched from Kivy to Java which allowed for smoother transitions and the implementation of SMS Security Messages to be sent and received. Finally, by switching the object avoidance and navigation controllers to be based on the robot’s orientation angle allowed for smoother and more stable operation in straight-line movement and object avoidance situations. Being able to navigate to designated rooms and avoid objects along the way, this system can be implemented into a business to deliver objects as needed.

XI. Appendix

A. Main Motor Delivery (Arduino)

```
void setup() {
  Serial.begin(115200);   // Arduino to Java connection
  Serial1.begin(115200);  // Arduino to RFID connection

  /****Motor/Encoder Setup****/
  attachMotors();
  attachEncoders();

  attachInterrupt(0, rightEncoder, CHANGE);
  attachInterrupt(1, leftEncoder, CHANGE);

  /****Ultrasonic Setup****/
  attachUltrasonics();

  /****LED Setup****/
  attachLED();

  motorStop();
  delay(2000);
}

void loop()
{
  /***** Clear rx/tx messages and flags *****/
  for(int i = 0; i < 2; i++)
  {
    rx_msg[i] = 0x00;
    tx_msg[i] = 0x00;
    temp_msg[i] = 0x00;
  }

  byte SETUP    = TRUE;
  byte DELIVERY = FALSE;
  byte AT_ROOM  = FALSE;
  byte RETURN   = FALSE;
  byte AT_HOME  = TRUE;

  RCV_MSG = FALSE;
  ROOM_SET = FALSE;

  setDis = 0;

  digitalWrite(LED1, LOW);
  digitalWrite(LED2, LOW);
  digitalWrite(LED3, LOW);
  digitalWrite(LED4, LOW);
  digitalWrite(LED_STATUS, LOW);
  /*****************************************/

  while(SETUP == TRUE)  // Wait for GUI input...
  {
    waitLED(1, 200);  // Wait for room number...

    // Read from Port 0 (GUI)
    if(Serial.available() > 0) {
      rx_msg[0] = Serial.read();
      delay(1);
      rx_msg[1] = Serial.read();

      for(int i = 0; i < 2; i++)
        tx_msg[i] = rx_msg[i];  // Set tx_msg to rx_msg

      RCV_MSG = TRUE;
    }
    // GUI input rx'd
    if(RCV_MSG == TRUE)
    {
      digitalWrite(LED1, HIGH);   // Room number received!
      delay(200);

      // Clear rx_msg
      for(int i = 0; i < 2; i++)
        rx_msg[i] = 0x00;

      // Mega to RFID
      Serial1.write(tx_msg, 2);
      xmitLED(MEGA_2_RFID);

      // Clear tx_msg
      for(int i = 0; i < 2; i++)
        tx_msg[i] = 0x00;

      while(ROOM_SET == FALSE) // Wait for room location info...
      {
        // Read from Port 1 (RFID)
        if(Serial1.available() > 0) {
          rx_msg[0] = Serial1.read();
          delay(1);
          rx_msg[1] = Serial1.read();

          ROOM_SET = TRUE; // Room location information rcv'd from RFID Arduino
        }

        for(int i = 0; i < 2; i++)
          temp_msg[i] = rx_msg[i];  // Set temp_msg to rx_msg
        for(int i = 0; i < 2; i++)
          rx_msg[i] = 0x00;         // Clear rx_msg

        // RFID confirmed room info
        if(ROOM_SET == TRUE)
        {
          SETUP = FALSE;    // Setup complete! Begin delivery
          DELIVERY = TRUE;

          // Move to desired wall distance
          if( (bitRead(temp_msg[1], MTR_BIT) == 1) && (bitRead(temp_msg[1], HOME_BIT) == 0) )
          {
            // Motor stop
            motorStop();

            // Zero turn 90 deg based on room direction
            if(bitRead(temp_msg[1], Z_BIT) == LEFT)
               zeroTurn(LEFT, 90);
            else if(bitRead(temp_msg[1], Z_BIT) == RIGHT)
               zeroTurn(RIGHT, 90);
          }
        }
      }
    }
  }

  while(DELIVERY == TRUE) // Wait to arrive at room...
  {
    angleControl();

    waitLED(2, 200);  // Wait to arrive at room...

    // Read from Port 1 (RFID)
    if(Serial1.available() > 0) {
      rx_msg[0] = Serial1.read();
      delay(1);
      rx_msg[1] = Serial1.read();

      if(rx_msg[0] == temp_msg[0])
      {
        DELIVERY = FALSE; // Desired room reached
        AT_ROOM = TRUE;
      }
    }

    if(DELIVERY == FALSE && AT_ROOM == TRUE)
    {
      digitalWrite(LED2, HIGH);   // Arrived at room!
      delay(200);

      for(int i = 0; i < 2; i++)
        temp_msg[i] = rx_msg[i];  // Set temp_msg to rx_msg
      for(int i = 0; i < 2; i++)
        rx_msg[i] = 0x00;         // Clear rx_msg

      // Signifies room reached
      if(bitRead(temp_msg[1], MTR_BIT) == 0 && bitRead(temp_msg[1], ROOM_BIT) == 1)
      {
        motorStop();

        if(bitRead(temp_msg[1], Z_BIT) == LEFT)
          zeroTurn(LEFT, 90);
        else
          zeroTurn(RIGHT, 90);
      }

      for(int i = 0; i < 2; i++)
        tx_msg[i] = temp_msg[i];    // Set temp_msg to rx_msg
      for(int i = 0; i < 2; i++)
        temp_msg[i] = 0x00;         // Clear rx_msg

      // Mega to GUI
      Serial.print(tx_msg[0], HEX);
      Serial.println(tx_msg[1], HEX);
      xmitLED(MEGA_2_GUI);
    }
  }

  while(AT_ROOM == TRUE)  // Wait to complete delivery...
  {
    waitLED(3, 200);  // Wait to complete delivery...

    // Read from Port 0 (Java)
    if(Serial.available() > 0)
    {
      rx_msg[0] = Serial.read();
      delay(1);
      rx_msg[1] = Serial.read();

      for(int i = 0; i < 2; i++)
        tx_msg[i] = rx_msg[i];  // Set tx_msg to rx_msg
    }

    if(rx_msg[0] == 0x20) // Home room number received!
    {
      digitalWrite(LED3, HIGH);   // Home room number received!
      delay(200);

      for(int i = 0; i < 2; i++)
        rx_msg[i] = 0x00; // Clear rx_msg

      // Mega to RFID
      Serial1.write(tx_msg, 2);
      xmitLED(MEGA_2_RFID);

      for(int i = 0; i < 2; i++)
        tx_msg[i] = 0x00;  // Clear tx_msg

      ROOM_SET = FALSE;

      while(ROOM_SET == FALSE)
      {
        // Read from Port 1 (RFID)
        if(Serial1.available() > 0)
        {
          rx_msg[0] = Serial1.read();
          delay(1);
          rx_msg[1] = Serial1.read();

          ROOM_SET = TRUE;
        }

        for(int i = 0; i < 2; i++)
          temp_msg[i] = rx_msg[i];  // Set temp_msg to rx_msg
        for(int i = 0; i < 2; i++)
          rx_msg[i] = 0x00;         // Clear rx_msg

        if(ROOM_SET == TRUE)
        {
          AT_ROOM = FALSE;
          RETURN = TRUE;

          // Return home
          if( (bitRead(temp_msg[1], MTR_BIT) == 1) && (bitRead(temp_msg[1], ROOM_BIT) == 0) )
          {
            if(bitRead(temp_msg[1], Z_BIT) == LEFT)
              zeroTurn(LEFT, 90);
            else
              zeroTurn(RIGHT, 90);
          }
        }
      }
    }
  }

  while(RETURN == TRUE) // Wait to arrive at home...
  {
    angleControl();

    waitLED(4, 200);  // Wait to arrive at home...

    // Read from Port 1 (RFID)
    if(Serial1.available() > 0) {
      rx_msg[0] = Serial1.read();
      delay(1);
      rx_msg[1] = Serial1.read();

      if(rx_msg[0] == temp_msg[0])
      {
        RETURN = FALSE;
        AT_HOME = TRUE;
      }
    }

    if(RETURN == FALSE && AT_HOME == TRUE)  // Arrived at home!
    {
      digitalWrite(LED4, HIGH);   // Arrived at home!
      delay(200);

      for(int i = 0; i < 2; i++)
        temp_msg[i] = rx_msg[i];    // Set temp_msg to rx_msg
      for(int i = 0; i < 2; i++)
        rx_msg[i] = 0x00;         // Clear rx_msg

      if( (bitRead(temp_msg[1], MTR_BIT) == 0) && (bitRead(temp_msg[1], HOME_BIT) == 1) )
      {
        // Zero turn 90 deg based on room direction
        if(bitRead(temp_msg[1], Z_BIT) == LEFT)
          zeroTurn(LEFT, 90);
        else if(bitRead(temp_msg[1], Z_BIT) == RIGHT)
          zeroTurn(RIGHT, 90);
          // Motor stop
          motorStop();
        }

        for(int i = 0; i < 2; i++)
          tx_msg[i] = temp_msg[i];    // Set temp_msg to rx_msg
        for(int i = 0; i < 2; i++)
          temp_msg[i] = 0x00;         // Clear rx_msg

        // Mega to GUI
        Serial.print(tx_msg[0], HEX);
        Serial.println(tx_msg[1], HEX);
        xmitLED(MEGA_2_GUI);

        homeLED();
      }
   }
}
```

B. Main RFID Delivery (Arduino)

```
void setup() {
  Serial.begin(115200);   // RFID reader port
  Serial1.begin(115200);  // Main Arduino port

  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
  pinMode(LED_STATUS, OUTPUT);

  for(int i = 0; i < 5; i++)
  {
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, HIGH);
    digitalWrite(LED4, HIGH);
    digitalWrite(LED_STATUS, HIGH);
    delay(50);
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED4, LOW);
    digitalWrite(LED_STATUS, LOW);
    delay(50);
  }

  if(setupNano(38400) == false)  // Configure nano to run at 38400bps
    while (1); //Freeze!

  nano.setRegion(REGION_NORTHAMERICA);
  nano.setReadPower(2000); // 25 dBm
}

void loop()
{
  /***** Clear rx/tx messages and flags *****/
  temp_msg = 0x00;

  for(int i = 0; i < 2; i++)
  {
    rx_msg[i] = 0x00;
    tx_msg[i] = 0x00;
  }

  currentRoom = homeRoom;
  desiredRoom = 0x00;
  prevRoom = 0x00;

  roomDirection = 0;
  roomSide = 0;
```

```
for(int i = 0; i < 12; i++)
  myEPC[i] = 0;

SETUP    = TRUE;
DELIVERY = FALSE;
AT_ROOM  = FALSE;
RETURN   = FALSE;
AT_HOME  = TRUE;

RCV_MSG = FALSE;

digitalWrite(LED1, LOW);
digitalWrite(LED2, LOW);
digitalWrite(LED3, LOW);
digitalWrite(LED4, LOW);
digitalWrite(LED_STATUS, LOW);
/****************************************/

while(SETUP == TRUE)  // Wait for GUI input...
{
  waitLED(1, 200);  // Wait for room number...

  // Read from Port 1 (MEGA)
  if(Serial1.available() > 0) {
    rx_msg[0] = Serial1.read();  // Upper byte stores room #
    delay(1);
    rx_msg[1] = Serial1.read();  // Lower byte stores bit checks

    RCV_MSG = TRUE;
  }

  // GUI input rx'd
  if(RCV_MSG == TRUE)
  {
    digitalWrite(LED1, HIGH);   // Room number received!
    delay(200);

    desiredRoom = rx_msg[0];          // Set desired room to the rx'd room #
    checkRoomLocation(desiredRoom);   // Check direction of room from base & send info to mega

    temp_msg = tx_msg[1];

    // Set room location (which side of hallway)
    if(bitRead(temp_msg, SIDE_BIT) == LEFT)
      roomSide = LEFT;
    else
      roomSide = RIGHT;

    SETUP = FALSE;
    DELIVERY = TRUE;
  }
}

while(DELIVERY == TRUE) // Wait to arrive at room...
{
  waitLED(2, 200);
  scanRFIDs();  // Scan RFID's until reach desired room

  if(currentRoom == desiredRoom && desiredRoom != homeRoom)
  {
    digitalWrite(LED2, HIGH);  // Arrived at room!
    delay(200);

    bitClear(temp_msg, MTR_BIT);  // Set motors off
    bitSet(temp_msg, ROOM_BIT);   // Desired room reached

    if(roomSide == LEFT)
      bitClear(temp_msg, Z_BIT);  // Turn to left
    else
      bitSet(temp_msg, Z_BIT);    // Turn to right
    tx_msg[1] = temp_msg;

    // RFID to Mega
    Serial1.write(tx_msg, 2);
    xmitLED(RFID_2_MEGA);

    DELIVERY = FALSE;
    AT_ROOM = TRUE;

    RCV_MSG = FALSE;
  }
  else
    prevRoom = currentRoom;
}

while(AT_ROOM == TRUE)  // Wait to complete delivery...
{
  waitLED(3, 200);  // Wait to complete delivery...

  // Read from Port 1 (MEGA)
  if(Serial1.available() > 0)
  {
    rx_msg[0] = Serial1.read();  // Upper byte stores room #
    delay(1);
    rx_msg[1] = Serial1.read();  // Lower byte stores bit checks
  }

  if(rx_msg[0] == homeRoom) // Delivery complete!
  {
    digitalWrite(LED3, HIGH);  // Delivery complete!
    delay(200);

    desiredRoom = rx_msg[0];
    tx_msg[0] = desiredRoom;

    bitSet(temp_msg, MTR_BIT);  // Turn motors on
    bitClear(temp_msg, ROOM_BIT);   // Clear room bit

    // Zero turn based off previous zero turn
    if(roomSide == LEFT)
      bitClear(temp_msg, Z_BIT);
    else
      bitSet(temp_msg, Z_BIT);

    // Set side of home for wall following
    if(roomDirection == LEFT)
      bitClear(temp_msg, SIDE_BIT);
    else
      bitSet(temp_msg, SIDE_BIT);

    tx_msg[1] = temp_msg;

    // RFID to Mega
    Serial1.write(tx_msg, 2);
    xmitLED(RFID_2_MEGA);

    AT_ROOM = FALSE;
    RETURN = TRUE;
  }
}

while(RETURN == TRUE)   // Wait to arrive at home...
{
  waitLED(4, 200);  // Wait to arrive at home...
  scanRFIDs();  // Scan RFID's until reach desired room

  if(currentRoom == desiredRoom && desiredRoom == homeRoom)
  {
    digitalWrite(LED4, HIGH);  // Arrived at room!
    delay(200);
```

```
        bitClear(temp_msg, MTR_BIT);    // Set motors off
        bitSet(temp_msg, HOME_BIT);     // Home reached
        bitClear(temp_msg, ROOM_BIT);   // Desired room clear

        if(roomDirection == LEFT)
          bitClear(temp_msg, Z_BIT);
        else
          bitSet(temp_msg, Z_BIT);

        tx_msg[0] = homeRoom;
        tx_msg[1] = temp_msg;

        // RFID to Mega
        Serial1.write(tx_msg, 2);
        xmitLED(RFID_2_MEGA);

        RETURN = FALSE;
        AT_HOME = TRUE;

        if(AT_HOME == TRUE)
          homeLED();
      }
      else
        prevRoom = currentRoom;
    }
}
```

C. UARTCommunication (Java)

```
  public class SerialIO implements SerialPortEventListener
  {
      SerialPort;

      private static final int TIME_OUT = 250;
      private static final int DATA_RATE = 115200;

      private BufferedReader input;
      private OutputStream output;

      private final Object stopRead = new Object();
      private final Object stopWrite = new Object();
      public boolean stopR = false;
      public boolean stopW = false;

      String inputLine;
      String hexCheck = "-?[0-9A-F]+";
      String alphaCheck = "-?[A-F]+";
      String numCheck = "-?[0-9]+";

      private int[] nib = new int[4];
      public int[] inputVal = new int[2];
      public int byteVal;

      public static String cfmMessage;
      public static final String opPhone = "18881234567";

      public void initialize()
      {
          String portID = null;
          Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();

          while(portEnum.hasMoreElements())
          {
              CommPortIdentifier portIdentifier = (CommPortIdentifier) portEnum.nextElement();
              if(!portIdentifier.isCurrentlyOwned()     &&     portIdentifier.getPortType()     ==
CommPortIdentifier.PORT_SERIAL)
              {
                  portID = portIdentifier.getName();
                  System.out.println(portID);
                  break;
```

```
            }
        }

        if(portID == null)
        {
            System.out.println("Could not find COM port.");
            return;
        }

        try
        {
            CommPortIdentifier currPort = CommPortIdentifier.getPortIdentifier(portID);
            serialPort = (SerialPort) currPort.open(this.getClass().getName(),
                        TIME_OUT);
            serialPort.setSerialPortParams(DATA_RATE,
                        SerialPort.DATABITS_8,
                        SerialPort.STOPBITS_1,
                        SerialPort.PARITY_NONE);

            input   =   new   BufferedReader(new   InputStreamReader(serialPort.getInputStream(),
StandardCharsets.US_ASCII));
            output = serialPort.getOutputStream();

            serialPort.addEventListener(this);
            serialPort.notifyOnDataAvailable(true);
        } catch (Exception e) {System.err.println(e.toString());}
    }

    public synchronized void close()
    {
        if (serialPort != null)
        {
            serialPort.removeEventListener();
            serialPort.close();
        }
    }

    @Override
    public synchronized void serialEvent(SerialPortEvent s_evt)
    {
        if(s_evt.getEventType() == SerialPortEvent.DATA_AVAILABLE)
        {
            try
            {
                inputLine = input.readLine();
                inputToByte();

                if((MainFrame.deliveryPanel.isVisible())        &&        (inputVal[0]        ==
MainFrame.loadPanel.roomVal[0]))
                {
                    MainFrame.deliveryPanel.setVisible(false);
                    MainFrame.arrivalPanel.setVisible(true);

                    cfmMessage = "ADR has arrived safely at Room " + MainFrame.entryPanel.room;
                    //MainFrame.sms.SendSMS(username, password, cfmMessage, opPhone, url);
                }

                if((MainFrame.returnPanel.isVisible())        &&        ((inputVal[0]        ==
MainFrame.ratingPanel.homeVal[0]) || (inputVal[0] << 4) == MainFrame.ratingPanel.homeVal[0]))
                {
                    MainFrame.entryPanel.entryCount = 0;
                    MainFrame.entryPanel.chCount = 0;

                    /**** Entry Panel Reset ****/
                    Arrays.fill(MainFrame.entryPanel.phoneNumber, 0, 10, '\0');
                    Arrays.fill(MainFrame.entryPanel.roomNumber, 0, 4, '\0');
                    MainFrame.entryPanel.phone = "";
                    MainFrame.entryPanel.room = "";

                    MainFrame.entryPanel.answrLabel.setText("");
                    MainFrame.entryPanel.enter.setEnabled(false);
                    MainFrame.entryPanel.delete.setEnabled(false);

                    MainFrame.entryPanel.msgLabel.setText("<html><center>Please enter your phone
number:</center></html>");   // Reset
                    MainFrame.entryPanel.back.setText("cancel");     // Reset
                    MainFrame.entryPanel.cancel.setVisible(false);   // Reset
                    MainFrame.entryPanel.cancel.setEnabled(false);   // Reset

                    /**** LoadItem Panel Reset ****/
                    MainFrame.loadPanel.pcHolder.delete(0, 4);
                    Arrays.fill(MainFrame.loadPanel.passCode, 0, 4, '\0');
                    MainFrame.loadPanel.pc = "";

                    /**** Pass code Panel Reset ****/
                    MainFrame.passcodePanel.pcCount = 0;

                    Arrays.fill(MainFrame.passcodePanel.userPC, 0, 4, '\0');
                    MainFrame.passcodePanel.userPasscode = "";

                    MainFrame.passcodePanel.answrLabel.setText("");
                    MainFrame.passcodePanel.enter.setEnabled(false);
                    MainFrame.passcodePanel.delete.setEnabled(false);

                    /**** Arrival Panel Reset ****/
                    MainFrame.arrivalPanel.items.setText("Retrieve items");

                    MainFrame.returnPanel.setVisible(false);
                    MainFrame.startPanel.setVisible(true);

                    /**** Rating Panel Reset ****/
                    MainFrame.ratingPanel.homeVal[0] = 0x00;
                }
            }
            catch(IOException e) {System.err.println(s_evt.toString());}
      }
    }

    public void inputToByte()
    {
        String[] str = new String[4];

        if(inputLine.length() == 2)
        {
            inputLine = '0' + inputLine.substring(0, 1) + '0' + inputLine.substring(1, 2);
        }
        else if(inputLine.length() == 3)
        {
            inputLine = '0' + inputLine;
        }

        if(inputLine.length() == 4 && inputLine.matches(hexCheck))
        {
            for(int i = 0; i < inputLine.length(); i++)
            {
                str[i] = inputLine.substring(i,i+1);

                if(str[i].matches(numCheck))
                    nib[i] = str[i].charAt(0) - '0';
                else if(str[i].matches(alphaCheck))
                    nib[i] = str[i].charAt(0) - '7';
            }

            byteVal = (((nib[0] << 4) | nib[1]) << 8) | ((nib[2] << 4) | nib[3]);

            inputVal[0] = (nib[0] << 4) | nib[1];   // Room #
            inputVal[1] = (nib[2] << 4) | nib[3];   // Flags
        }
    }

    public void transmit(byte[] b)
    {
         try
         {
             output.write(b);
             synchronized(stopWrite) {stopW = false;}
         }
         catch(IOException e) {}
        }
}
```

References

[1] Ada, Lady. "Wiring & Test | Adafruit LSM9DS0 Accelerometer + Gyro + Magnetometer 9-DOF Breakouts | Adafruit Learning System". *Learn.adafruit.com*. N.p., 2017. Web. 23 Mar. 2017.

[2] "Amazon.Com: Uniquegoods H-Bridge DC Dual Motor Driver PWM Module DC 3~36V 10A Peak 30A IRF3205 High Power Control Board For Arduino Robot Smart Car: Home Improvement". *Amazon.com*. N.p., 2017. Web. 24 Apr. 2017.

[3] Beauregard, Brett. "Improving The Beginner'S PID – Introduction « Project Blog". *Brettbeauregard.com*. N.p., 2017. Web. 15 Mar. 2017.

[4] Enginoglu, Ozan. "DC Motor Speed Control With PID". *Hackaday.io*. N.p., 2017. Web. 10 Apr. 2017.

[5] "Five Ways To Run A Program On Your Raspberry Pi At Startup". *Dexter Industries*. N.p., 2017. Web. 7 May 2017.

[6] "IRF3205 HEXFET Power MOSFET". *irf.com*. N.p., 2017. Web. 24 Apr. 2017.

[7] Kohanbash, David. "PID Control". *Robotsforroboticists.com*. N.p., 2017. Web. 5 Mar. 2017.

[8] Lee, Zx. "Wall Following Robot". *Zx Lee*. N.p., 2017. Web. 9 Apr. 2017.

[9] Nedelkovski, Dejan. "Ultrasonic Sensor HC-SR04 And Arduino Tutorial". *HowToMechatronics*. N.p., 2017. Web. 22 Feb. 2017.

[10] "Pololu - 131:1 Metal Gearmotor 37Dx73l Mm With 64 CPR Encoder". *Pololu.com*. N.p., 2017. Web. 3 Mar. 2017.

[11] Raspberry Pi SD-Card Image - German-Robot.Com | Open Source Humanoid Robot". *German-Robot.com | Open Source Humanoid Robot*. N.p., 2017. Web. 10 Apr. 2017.

[12] "Robot Platform | Knowledge | Types of Robot Sensors", *Robotplatform.com*, 2017. [Online]. Available: http://www.robotplatform.com/knowledge/sensors/types_of_robot_sensors.html. [Accessed: 25- Feb- 2017].

[13] Simeonidis, Benedetta. "Lab: Using A Transistor To Control High Current Loads With An Arduino – ITP Physical Computing". *Itp.nyu.edu*. N.p., 2017. Web. 28 Feb. 2017.

[14] Simeonidis, Benedetta. "Lab: DC Motor Control Using An H-Bridge – ITP Physical Computing". *Itp.nyu.edu*. N.p., 2017. Web. 28 Feb. 2017.

[15] "TIP 120/ TIP 121/ TIP 122 NPN Epitaxial Darlington Transistor". *Mouser.com*. N.p., 2017. Web. 27 Feb. 2017.

[16] "TIP 125/TIP 126/TIP 127 PNP Epitaxial Darlington Transistor". *Redrok.com*. N.p., 2017. Web. 27 Feb. 2017.

[17] Tontos, Andras. "H-Bridges – The Basics | Modular Circuits". *Modularcircuits.com*. N.p., 2017. Web. 4 Mar. 2017.

[18] "Ultrasonic Ranging Module HC - SR04". *Micropik.com*. N.p., 2017. Web. 22 Feb. 2017.