ECE 445 Final Report – Spring 2017

# Automated Trash Can

Michael Gao, Aditya Sule, and Eshwar Cheekati

## Abstract

This report documents the design and implementation process for the Autonomous Trash Can design project. The purpose of our project is to design an intelligent trash can that keeps track of its own trash level and navigates to the door when it reaches a certain threshold fullness. By the end of the project timeline, trash level detection and web server data collection function as described in our requirements. However, we ran into issues with our navigation and control component. This will be described in detail in our verifications.

# Contents

# 1. Introduction

Trash management is an integral part of our daily lives. Just by doing simple everyday tasks like cooking or cleaning, we generate a substantial amount of waste. However, without proper monitoring, waste can stagnate in our trash cans, leading to unpleasant odors that spread throughout the household. A possible solution is to take out the trash frequently, but that may also be wasteful if the trash can is mostly empty. Likewise, different amounts of waste are generated day by day, so predicting exactly when to take out the trash becomes more difficult. Moreover, since the average person generates roughly 4.6 pounds of trash a day [1], the frequency in which we need to check and maintain our trash cans becomes greater. This becomes even more difficult considering the number of trash cans in a household. We can expect a kitchen trash can, individual room waste bins, and bathroom waste bins.

Our goal is to improve the trash can by automating parts of the trash management process. We will integrate a microcontroller chip, sensors, as well as a DC stepper motor connected to wheels to a small waste bin. Using an ultrasonic sensor placed on the lid of our trash can, we can detect the level of waste. When our waste level passes a certain threshold, our microcontroller sends a notification to our phones via wifi. The trash can then move itself to the door using a predetermined route. By doing so, we eliminate the need to constantly check the trash and provide a convenient reminder when the trash is full. If we see that the trash can is by the door, we know we can take out the trash on the way out. By doing these things, we essentially have an IoT trash can that simplifies the majority of trash management.

# 2. Design

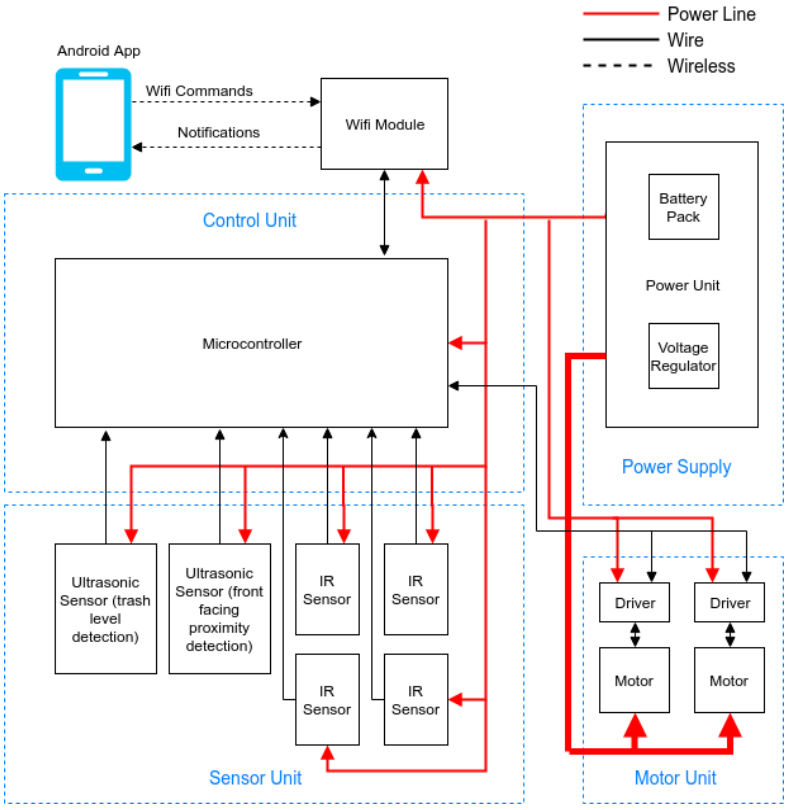



*Figure 1: High Level Block Diagram*

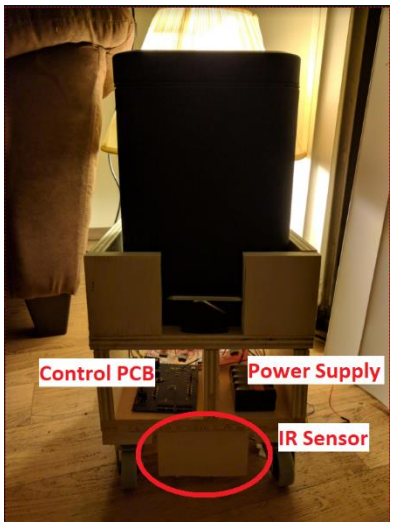



*Figure 2: Front View Prototype*

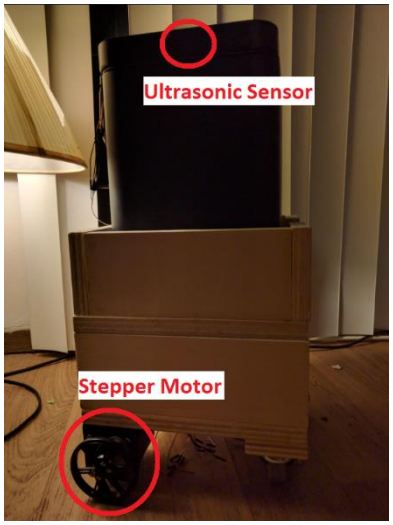



*Figure 3: Side View Prototype*

## 2.1. Physical Design

For our physical design, we needed to support our three PCBs, stepper motors, sensors, and house our trash can. To do this, we decided to use ¾ inch plywood since it is relatively cheap and the added width helps with chassis stability. The physical design contains an upper and lower level. The upper level has 4 walls to secure the trash can in place. The lower level houses our PCBs and power unit. The lower unit has our two stepper motors mounted with a mounting plate and two idler wheels for added stability. The front of the chassis also has a block that extends towards the ground with a 3mm clearance for our IR sensors.

## 2.2. Power Supply

It is a good idea to first calculate a rough estimate of the amount of power required for our project. To do this we decided to calculate the power required for the part of the trashcan that will require the most power, which is locomotion. This estimate will also be useful later on when we select an appropriate motor for our project. The power required to move the trashcan is consumed by the motors. The motors turn the wheels by applying a torque, which we must estimate first if we want to estimate the power. We will also have to assume that the motors are not a 100% efficient at converting power into torque and some of this energy is dissipated into the environment as heat. Incidentally, the torque we calculate here will also be useful in selecting an appropriate motor.

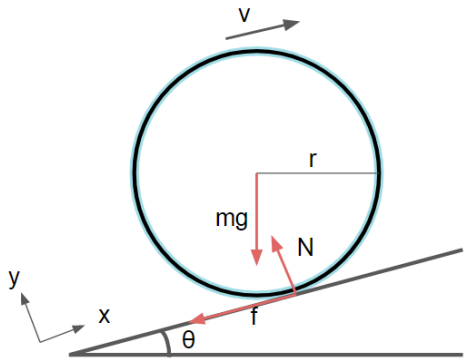



*Figure 4: Free Body Diagram*

To do this we will model our project with a simple system of a wheel on an incline with a weight pushing down on the wheel. We put the wheel on an incline to account for uneven flooring which might not be level. We will calculate the power required to keep the trashcan moving in uniform motion. We will make the following assumptions: mass $m = 10kg$, incline $\theta = 3°$, radius of wheel $r = 5cm$, efficiency $E = 0.6$, velocity $v = 5cm/s$. We need one more value, which is the coefficient of rolling friction between the wheels and the floor. In [2] it was found that a good estimate for the coefficient of friction between a rubber wheel and carpeted flooring is $\mu = 0.6$.

Fig 4 shows a freebody diagram of our system. Here f is the force of friction, $N$ is the normal force, $mg$ is the force of gravity. We can sum up the forces in the x and y directions to get the following equations.

$$f + mg\,sin(\theta) = F \tag{1}$$

Where $F$ is the total force acting on the trashcan.

$$N = mg\,cos(\theta) \tag{2}$$

We know that the force of friction is given by the formula

$$f = \mu N \tag{3}$$

By combining (1), (2) and (3) we get the torque required to turn the wheel. We can further plug in our assumptions to get a numerical estimate.

$$\tau = Fr = mg\,(sin(\theta) + \mu\,cos(\theta))\,r \tag{4}$$

$$\tau = 3.19\,Nm \tag{5}$$

Finally we can calculate the power required as follows.

$$P = \tau\omega/E \tag{6}$$

$$P = \tau v/Er = 5.31W \tag{7}$$

So we need about 5.5W of power.

### 2.2.1. NiMH Battery

As we have calculated we require a large portable source of electric power which is also rechargeable. We researched various different possible rechargeable battery types and the discussion basically boiled down to two battery types, Li-ion (Lithium ion) and NiMH (Nickel Metal-Hydride) batteries. We decided to go with NiMH batteries because they have several advantages over Li-ion batteries for our application. Li-ion batteries are highly susceptible to blunt-force damage and can catch fire if exposed to piercing damage. They also require much more complex battery protection circuitry during charge and discharge. While NiMH batteries are also vulnerable to damage from over-discharge, this is easier to design around than the problems with Li-ion batteries. NiMH batteries also have low internal resistance which is advantageous for our high voltage, high current draw.

For our project, we purchased a commercially available 12V NiMH battery that provides 2000mAH of power.

### 2.2.2. Voltage Regulator

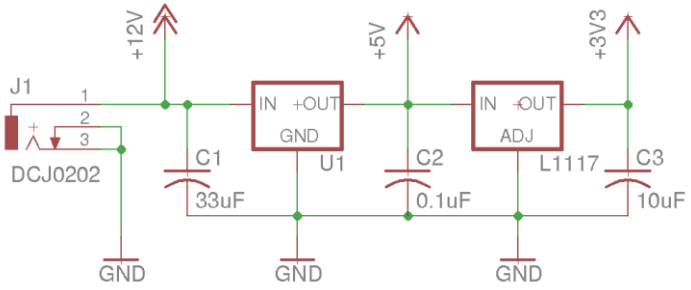



*Figure 5: Voltage Regulator Circuit*

For the electronic components of our project, we have two supply voltage requirements that are lower than the voltage supplied by our battery. All sensors and the microcontroller require a 5V operating voltage while the wifi module of our choosing requires a 3V power supply. For this reason we designed a simple voltage regulator circuitry. We used the popular L7805 and LD1117 voltage regulators that regulate voltage to 5V and 3.3V respectively [3]. To connect our battery to the voltage regulator circuit that is located on our main PCB we used a DCJ0202 jack.

## 2.3. Control Unit

The control unit consists of our microcontroller and our USB to serial converter which we use to communicate with the microcontroller. The microcontroller is responsible for the following tasks: read sensor data, run navigation logic, control motor units and the wifi module.

### 2.3.1. Microcontroller

The microcontroller we selected was the ATMega328pb which is part of Atmel's AVR line of microcontrollers. We selected this particular microcontroller because they are used on the popular Arduino development board and thus using this it gives us access to the extensive Arduino libraries, IDE and documentation. Arduino also provides a bootloader for the AVR that sets the sets the fuses and flags required to run the microcontroller. It also allows us to interface the microcontroller with the Arduino IDE using serial UART. The microcontroller has an internal crystal that provides a clock

frequency of 8MHz, but also provides the option of using an external crystal [4]. We used an external 16MHz crystal to provide a clock signal to the microcontroller because this is the frequency used by the Arduino boards and thus allows us to use the Arduino library's timing functions correctly. The microcontroller can be programmed by using ISP (in-circuit serial programmer), however we decided to use an Arduino board to program the microcontroller with the Arduino bootloader and then subsequently using a Serial to USB convertor to upload programs. This microcontroller also has 24 digital IO pins which is enough for the number of sensors and digital control signals that we ended up in our design [4].

### 2.3.2. CP2102 USB to Serial Converter

Once the microcontroller has been programmed with the bootloader, which only needs to be done once, we can upload programs to the microcontroller using UART Serial. To achieve this we used the commercially available USB to Serial converter CP1202 module. This module has a 6-pin connector that we connect to our main circuit board.

## 2.4. Sensor Unit

### 2.4.1. Ultrasonic Sensors

For trash level detection, we first had to compare weight and volume to find which would be more appropriate as our metric for fullness. After some preliminary research, we found that low cost weight sensors tend to drift in accuracy over time when placed under constant load. This would mean that our ability to log trash level data would skew over time and we would need a way to reset the sensor periodically. Furthermore, a heavy trash can does not guarantee a full one, so we decided to use volume as our fullness metric.

We looked into ranging sensors to help us calculate trash volume. By placing a ranging sensor on top of our trash can lid, we can read the distance between the sensor and the top of the trash surface. Using this measurement, we can calculate a percentage fullness based on the height of the trash can (24cm). The ranging sensor would also be used for our navigation proximity sensors.

We chose the HC-SR04 ultrasonic sensor as our ranging device. This sensor was chosen since it was relatively low cost and interfaced easily with our ATMEGA328p microcontroller. This ranging sensor has a detection distance from 2cm to 4m with up to 3mm accuracy. It operates on 5V DC and 15mA [5]. The sensor has 4 pins: Vcc, Trig, Echo and GND. A pulse is sent to the trig pin to start the ranging and the echo pin returns a signal when the ranging is complete.

The ultrasonic sensor works by sending out a sound wave and waiting to hear it back as an echo. By calculating the time difference between when the wave is sent and received and our knowledge that the wave travels at the speed of sound, we can calculate the distance traveled by the wave. To calculate the ranging estimate, we used the following equation:

$$Distance = \frac{T}{2} \times \frac{1}{Speed\ of\ Sound} \tag{8}$$

$T$ represents the time difference between when the wave is emitted and received as an echo. Since this is the time it takes for the wave to propagate back and forth between an object and the sensor and we

only care about the distance traveled one way, we divide $T$ by 2. By multiplying the result with 1 over the speed of sound in centimeters, we can get a ranging estimate in centimeters.

Since there is slightly error with reads using this sensor, we take the average of 50 ultrasonic sensor measurements to determine our final ranging estimate. This keeps our estimate within 1cm of the true distance value. For our obstacle avoidance protocol, we can use these ranging estimates directly. However, for trash level detection, we will need to make further calculations. Since we know that the distance between the sensor and the bottom of the trash can is 24cm, we can calculate the percentage fullness of the trash can with:

$$Fullness = \frac{24 - Distance(cm)}{24} \times 100\ \% \quad (9)$$

This gives us a percentage fullness for our trash can.

### 2.4.2. Infrared Sensors

We use 4 QRE1113 analog IR sensors to follow white tape as part of our system navigation. The sensors have an IR-emitting LED and the output of the sensors depends on how much of the IR light is reflected back. By using the analogRead() Arduino function, we can read the outputs of the sensors and calculate whether or not we are seeing the white tape based on the output of function. We determined that an analogRead() output of < 900 indicates the sensor reading white tape [6].

## 2.5. Motor/Navigation Unit

### 2.5.1. Stepper Motor

Our first major decision towards the navigation aspect of our project was deciding which type of motor to use. We initially selected a stepper motor over a normal DC motor due to the high torque offered by the stepper motor, the great positional control, and the low rpm. Due to the nature of the project, the trash can may take a long time to get to the destination but the priority is that it gets there. Therefore, the low rpm but high torque from stepper motors was an advantage. In deciding whether we wanted a unipolar or bipolar stepper motor, we selected a bipolar since it offers higher torque. The motor that we ended up selecting (RB-PHI-210) operates at a voltage of 8.6V, current of 2A, and has a holding torque of 2.35Nm.

After selecting the stepper motors and constructing the motor driver which is explained in the next section, we ran some tests on the stepper motor to see how it operates based on the clock frequency provided to the motor driver circuit. The results are in the following table.

*Table 1: Effect of Frequency on RPS for Stepper Motor*

| Frequency | Revolutions | Time | RPS |
|---|---|---|---|
| 500 | 50 | 19.88 | 2.52 |
| 400 | 50 | 25.91 | 1.93 |
| 350 | 50 | 28.55 | 1.75 |

| | | | |
|---|---|---|---|
| 300 | 50 | 33.33 | 1.5 |
| 250 | 50 | 33.45 | 1.49 |
| 200 | 10 | 10.10 | 0.99 |

We found that the motor rotates very smoothly at a clock frequency of 200-350Hz with little slippage.

### 2.5.2. Motor Driver

The stepper motors need their own driver circuit that will be able to generate the appropriate phase drive signals for the coils in the motor. These phase drive signals are what will allow the motor rod to rotate. This driver circuit will also act as the intermediary between the control unit microcontroller and the motors and provide us with control over the speed of the motors, the direction in which the motors will turn, and whether or not the driver circuit is even enabled to turn the motors. We opted with constructing a driver circuit consisting of the L297, L298, and Schottky diodes due to their documentation and applications specific to stepper motors. The schematic for our implementation is described by figure 7.

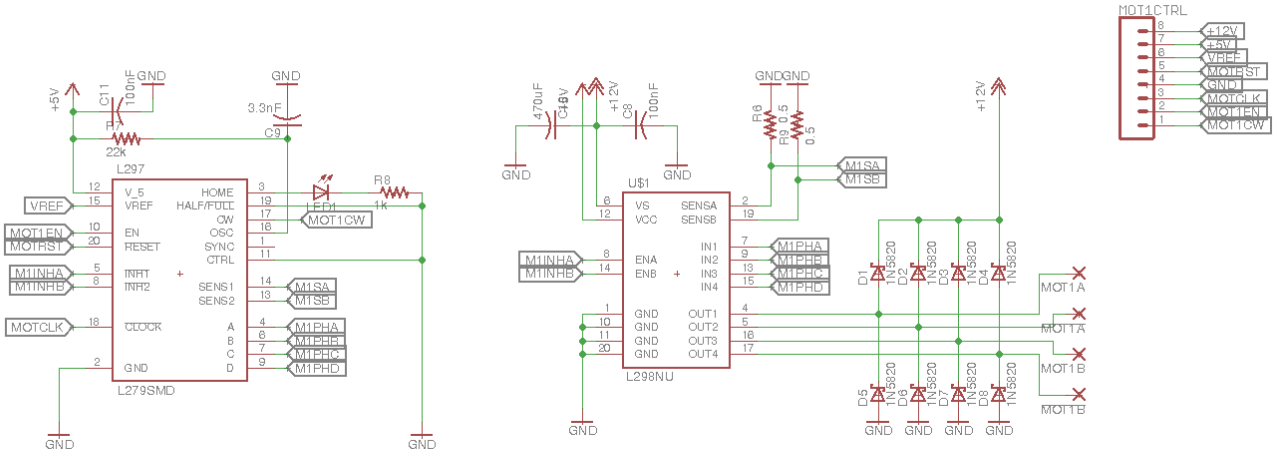



*Figure 6: Motor Driver Schematic*

The L297 in the schematic generates the proper phase drive signals for the necessary full step and half step rotations of the motor [7]. The L298N acts as a full bridge that enables voltage to be applied across a load in either so it allows the current direction to be changed in the coils for the motor [8]. The 8 fast diodes on the right are used to keep voltage from being generated from the inductance in the motor windings. These diodes have to meet the requirement shown in figure 7 of having a reverse recovery time <= 200ns and a forward voltage <= 1.2V at a current of 2A. Therefore, we selected the Schottky 1N5820 diode since it has a forward voltage of 500mV @ a current of 3A and a reverse recovery time that fits the requirement. The L298N will heat up due to the immense power dissipation so an additional multi-watt heatsink was used as well.

The driver circuit takes three logical inputs. They are the clock frequency which affects the speed and rotation of the motor, the enable, and the clockwise or counterclockwise direction of rotation. For the enable and the direction, we feed those inputs from the control unit's microcontroller since we want to modify those based on our navigation algorithm. For the clock frequency, we initially had it coming from the microcontroller as well. However, after doing more research, we found out that using the Atmega328PB's internal PWM to generate the optimal frequencies we need couldn't be done without the microcontroller having timing related issues. This would affect the functionality of certain functions such as the delay() function or the micros() function which are used extensively in our navigation algorithm as well as in our wifi module. To not compromise the efficiency of the navigation or wifi, we opted to hardwire the clock frequency to a set value of 291 Hz using the NE555 timer IC. The schematic for generating the desired frequency using the NE555 is as follows.

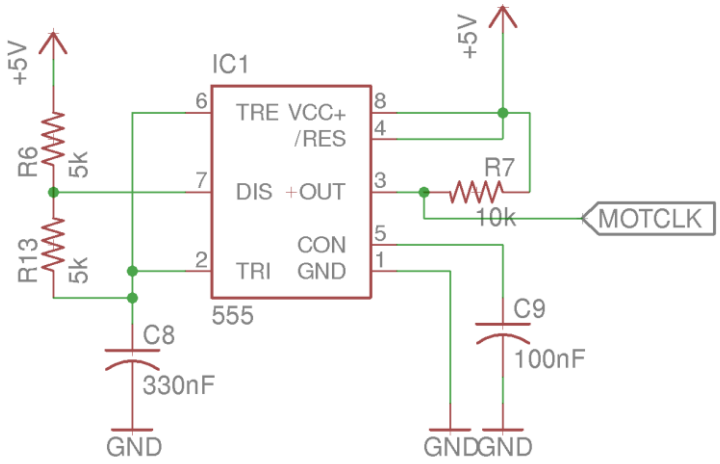



*Figure 7: Timer Schematic*

By using the NE555 in astable operation mode, we could generate a square wave output with a frequency and duty cycle that is dependent on R6, R13, and C8 in figure 8 [9]. These are the following formulas for calculating the frequency and duty cycle.

$$frequency = \frac{1.44}{(R6 + 2(R13))C} \tag{10}$$

$$duty\ cycle = 1 - \frac{R13}{R6+2(R13)} \tag{11}$$

Using $R6 = 5k\Omega$, $R13 = 5k\Omega$, and $C = 330nF$, we obtain a desired frequency of 291Hz with a duty cycle of 66.6%.

### 2.5.3. Tape-Following Navigation

To move the trash can to the destination, we set up white tape on the ground for the trash can to follow. Four IR sensors will be used to keep track of the tape and navigate the system. 2 IR sensors are placed within the tape width, and 2 IR sensors are placed outside the tape width. Depending on which sensors detect the line, the enable input and the direction input for the motor drivers are modified. Since we hardwired the clock to a set frequency of 291Hz, based on table 1, the motor revolutions per sec at that frequency is approximately 1.5. We use this RPS value along with the wheel radius of 40mm

to calculate that the linear velocity of the wheel from the motor rotation without any load is ideally 37.7 cm/sec.

$$v = rw \tag{12}$$

$$v = 0.040(1.5)(2\pi) \tag{13}$$

$$v = 37.7\ cm/sec \tag{14}$$

Now, we employ differential steering and accordingly turn the left and right motor on and off depending on which direction the system should move. Turning the motors on and off can be done by modifying the enable input for the driver. The navigation logic tries to always center the middle two IR sensors on the white tape. If the sensors on the right detect the tape, the system will move left and if sensors on the left detect the tape, the system will move right. If none of the sensors see the tape, then we stop moving the system since there has been an error and if all sensors read the tape, then we stop because we reached the destination. Two special cases we handle are when the center two sensors read the tape along with the left sensor in which we make sharp left turn and when the center two read the tape with the right sensor, we make a sharp right turn. Sharp left turns are done by turning both motors on but making the right motor turn clockwise and the left motor turn counter-clockwise. Similar logic applies for the sharp right turns.

### 2.5.4. Obstacle Avoidance

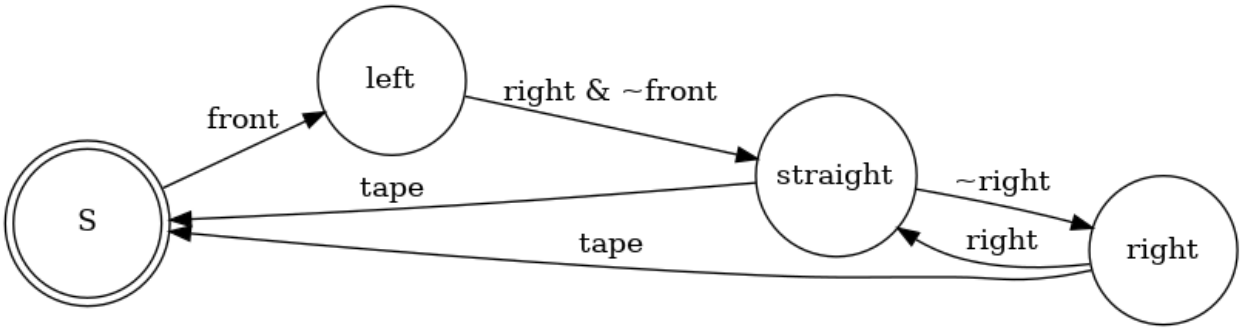



*Figure 8: Obstacle Avoidance State Diagram*

The obstacle avoidance is handled independently from the navigation discussed above. To detect obstacles we three of the same ultrasound sensors used to detect the trash level. However, in this case, we do not care about the actual distance to the obstacle and only care about detecting the obstacle from a sufficient distance. One of the sensors is placed in the front of the trashcan chassis while the other two are mounted on the left and right of the chassis respectively. The obstacle avoidance system kicks in when the front sensor detects an obstacle. From here, the obstacle system makes the choice of whether to turn left to avoid the obstacle or to turn right. If the left sensor is active we turn right, if the right sensor is active, we turn right and if both sensors are active we reverse until we can turn. Once we have turned left or right we hug around the obstacle until we detect the tape again with the IR sensors. To do this, we have a simple finite state machine for the left and the right direction. Figure 9 shows the state machine for the left direction. The transitions are triggered by changes in sensor readings as

labeled. The states are the directions we give to the motors. S is the start state. Once the tape has been found control is returned to the navigation unit.

## 2.6. Wifi Unit

With our fullness measurements derived from our ultrasonic sensor, we needed a way to present that information in a convenient way to the end user. For this we considered using either bluetooth or wifi. For bluetooth, the user could connect to our trash can via bluetooth and access a phone application to view fullness data. For wifi, the fullness data would be hosted on a web server and presented on a front end for the user to view. Using bluetooth, however, placed a severe limitation on user convenience. The user could only access the phone application and view up to date information if in close enough proximity to connect via bluetooth, at which point the user might as well check the trash themselves to check for fullness. On the other hand, uploading fullness data with wifi would allow this information to be accessed anywhere with an internet connection since the user could simply visit our hosted website.

The wifi module we'll be using is the ESP8266. This device is relatively low cost (~$7), supports all the features we need (full TCP/IP stack), and has substantial documentation online. This will be used to allow our trash can to be an IoT device. Then using a computer or smartphone, we can send HTML requests to our trash can to query sensor information or configure other trash can settings.

The device is controlled by sending serial commands over RX and TX pins on the board. UTXD and URXD are tied to two digital IO pins on our ATMEGA328p. VCC and CH_PD (enable) are tied to 3.3V (regulated from 5v with our voltage regulator) and GDN to ground [10]. Then by using the software serial library, we can emulate serial pins and send commands to our wifi device.

## 2.7. Software: Webserver

Our webserver collects and presents fullness data sent from our trash can with the ESP8266 wifi module. The web server is hosted on a local linux machine (separate from our trash can prototype) and handles HTTP request to receive and return fullness data.

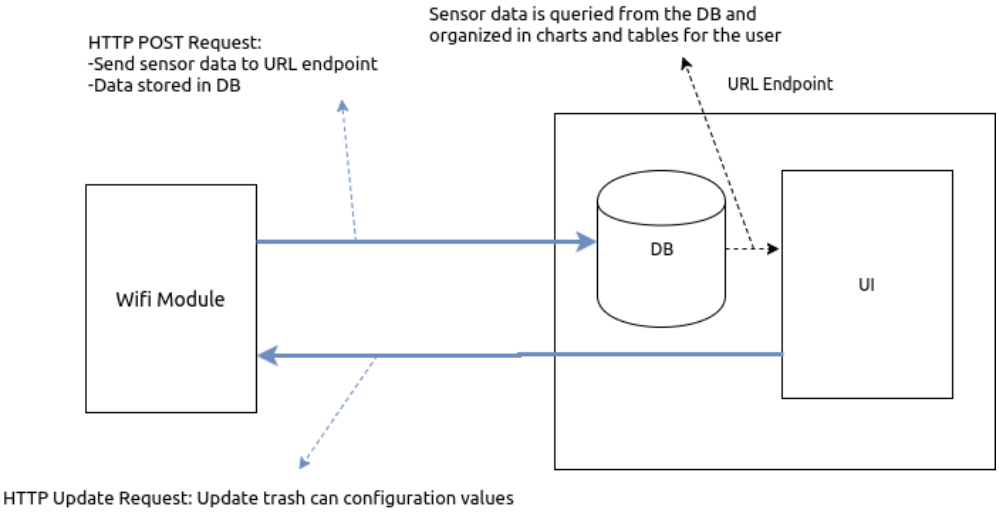



*Figure 9: Web Server Architecture*

The high level block diagram above shows the general design of our wifi module - web server interface. Our wifi module connects to our web server and after doing so, it can make HTTP requests to our server. A feature we wanted to add was the capability to send HTTP request back to our wifi module from our web server front end UI. This would allow the user to configure certain trash can settings, such as how often we check for trash level. Currently, our server is configured to handle HTTP POST and GET request.

HTTP POST request are sent from the wifi module and contain percentage fullness value as a URL-encoded value. The web server receives the fullness data and attaches a timestamp (based on when this request was received) and stores the data sample in a Mongo database.

HTTP GET request are sent from the wifi module and the web server handles these request by returning all the fullness data samples in JSON format.

The figure above shows a section of the returned JSON from a GET request. The first item boxed in red is the trash percentage fullness value sent from our wifi module. The second item boxed in red is the time stamped added by our web server on receipt of a POST request.

### 2.7.1. Front-End UI

The above figure shows the header toolbar for our front end UI. The front end UI queries the Mongo database for trash level data and collects it into convenient views for the user. The user can view fullness data in 4 different views:

- Raw Data: This presents every data sample in a list
- Daily Trend: This presents every data sample from the current day in a chart visualization
- Weekly Trend: This presents every data sample from the past 7 days in a chart visualization
- Monthly Trend: This presents every data sample from the current month in a chart visualization

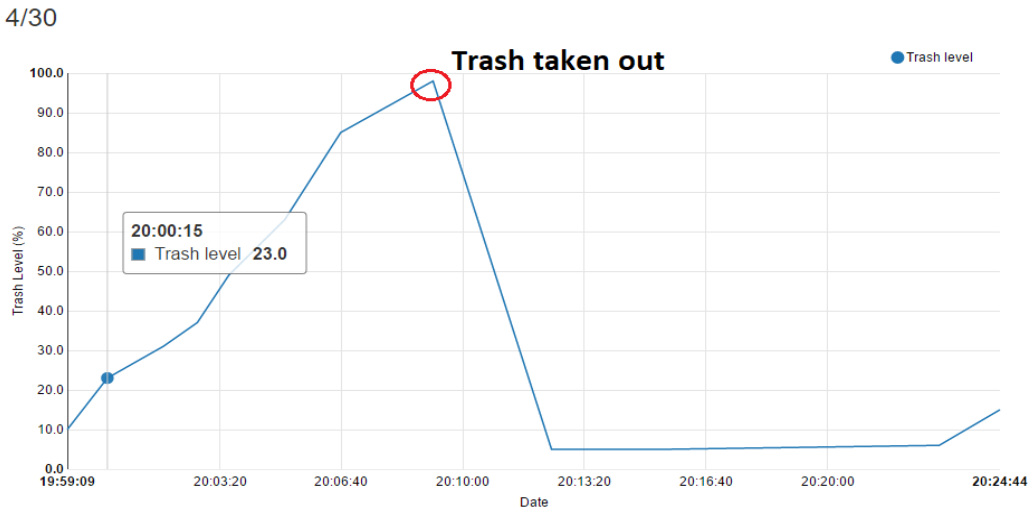



*Figure 11: Daily Trend View*

The above figure shows a daily view for April 30th where the X axis represents time (in Hours:Minutes:Seconds) and the Y axis represents percentage fullness from 0 to 100. Trash level accumulates slowly over time and after reaching a certain threshold (above 90% fullness) the user is

given a notification to take out the trash. When the trash is emptied, the next trash level data sample received indicates an empty trash can. This pattern occurs cyclically in all our views.

## 2.8. PCBs

We designed two PCB designs. The first was our main PCB. The main PCB contained the microcontroller and clock, motor clock signal generator, potentiometer to set Vref for the motor unit, and our voltage regulators. We used male surface mount headers to connect to the sensors and other PCBs. We used a female surface mount header to connect to the wifi module. The wifi module operates on a 3.3V logic voltage and required us to implement some logic level changers which we also placed on the main PCB.

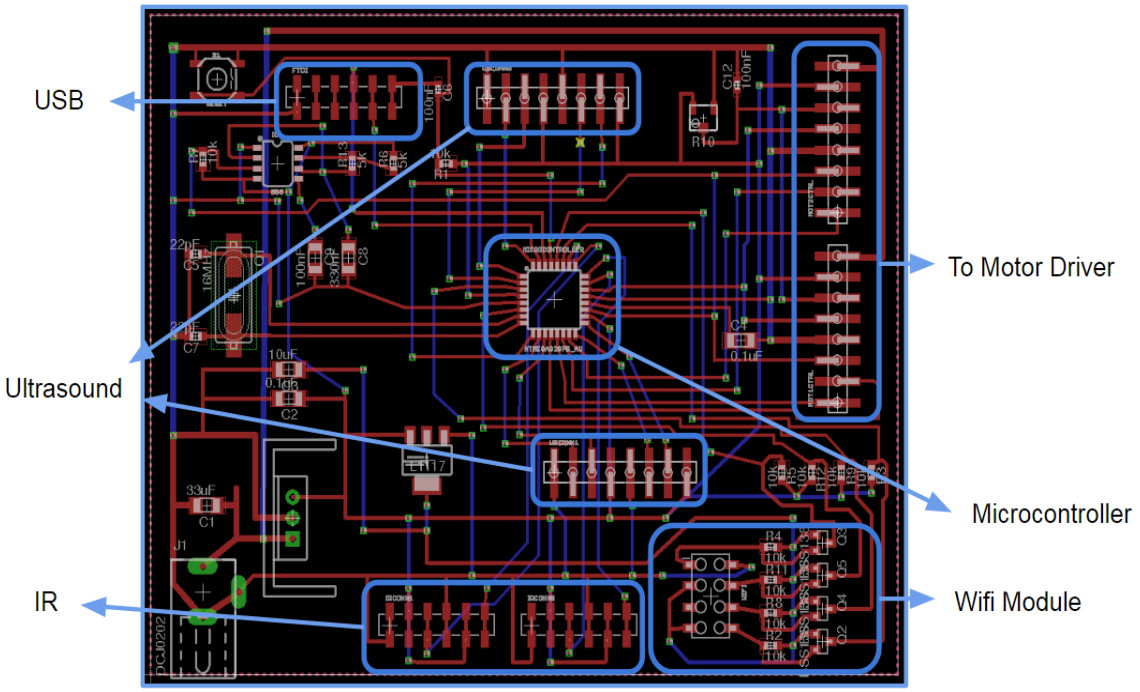
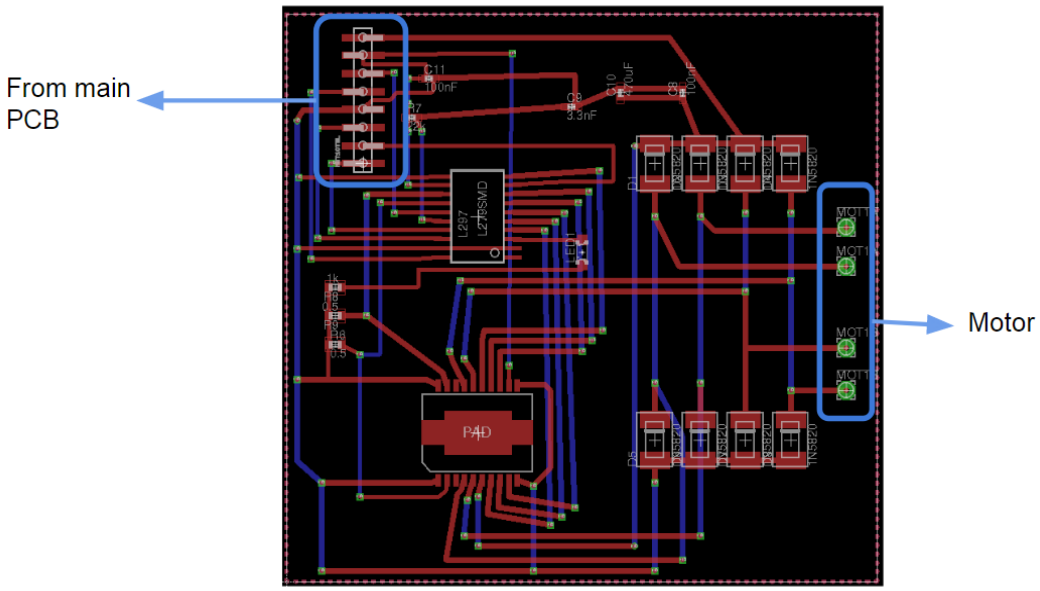



*Figure 12: Control PCB*

We also designed a PCB for a single motor driver. It contains the motor driver circuit discussed previously. It connects to the main PCB using a male surface mount header. The wires for the motor connect to this PCB as well. Since we have two motors in our design, we need two of these PCBs.




E

*Figure 13: Motor Driver PCB*

# 3. Verification

## 3.1. Control Unit and Main PCB

The requirements for our main PCB were to be able to communicate with the microcontroller with the Arduino ISP, read data from each sensor, and generate the required clock signal and Vref signal for the motor PCB. We had several problems with the actual implementation, however, because the microcontroller we ordered was a slightly newer version than the one Arduino library supported. We found the right bootloader and managed to program it and run sketches on it, but when in the process of programming the microcontroller, installing it on and debugging the PCB, we ruined the two units that we had ordered. As it would turn out, not ordering enough spares was one of the biggest reasons we could not meet many of our requirements. Another issue with the main PCB was that the crystal was placed too far away from the microcontroller on the PCB. We were able to program the microcontroller successfully off the circuit boards.

## 3.2. Trash Level Detection

Our original requirement for trash level detection was for the ultrasonic sensor to detect an object 10cm away within 2cm of accuracy. This requirement is met depending on the type of trash in our trash can. If our trash is compact and the trash surface is even, we can detect trash level within 1cm of accuracy. However, if trash is loose with a very uneven surface, our accuracy dips substantially. For verification, I ran the following test (where distances are between trash level and sensor):

- Compact trash at a distance of 5cm: ultrasonic sensor reads 6cm
- Compact trash at a distance of 15cm: ultrasonic sensor reads 15cm
- Loose trash with lowest point at 15cm and highest at 5cm: ultrasonic sensor reads 12cm

- Loose trash with lowest point at 10cm and highest at 5cm: ultrasonic sensor reads 8cm

We can see from these results that our trash level detection is most accurate with compact trash. This is because with compact trash, we have a more even surface for ultrasonic sensor sound waves to reflect. With loose trash, we see that our ranging estimates tend towards the lowest points. Further work that we can do to improve the accuracy of our detection with all types of trash is to have an array of ultrasonic sensors and taking an average of their read values. This, however, also presents other issues since the sensors can't distinguish between their emitted sound waves and sound waves emitted from neighboring ultrasonic sensors. This can be solved by modulating the wave emitted by each sensor so that we can uniquely identify each wave.

## 3.3. Web Server

Our requirements for our web server were to handle HTTP request and present that information on a hosted URL that can be accessed by external networks (external from the network hosting the web server). These requirements are met by the web server since we handle both HTTP GET and POST request and have the information in collected views in a front end UI.

## 3.4. Physical Design

The requirements for our physical design were the following:

- Stand upright without any significant tilt
- IR sensor block should provide less than 1cm of clearance from the ground
- Wheel radius should provide less than 40cm/s
- Unit should be able to withstand and move with up to a 10kg load

The first two requirements were met by our physical design. The unit stood upright without any significant tilt and the IR sensor block had a clearance of 0.5 cm from the ground. However, since we never got our motor driver to work with our portable power supply, we were never able to test the latter 2 requirements.

## 3.5. Navigation

The requirements for the navigation unit were the following.

- Motor driver has short switching time when toggling enable or turn direction.
- Front proximity sensor should be able to detect an obstacle 10-15cm away.
- IR sensors should detect white color with an analog read threshold of less than 900 at 1.2cm clearance.
- System should navigate to the door with 90% success rate with no obstacles
- System should navigate to the door with obstacles with 70% success rate with obstacles

The first three requirements were verified by our navigation unit. The switching time was less than .5 seconds, the front proximity ultrasonic sensor accurately read obstacles, and the IR sensors easily read the tape with a color threshold of 900. Navigating to the door with or without obstacles could not be verified however due to errors that occurred when connecting the battery to the motor driver PCB.

Before explaining the errors, it's important to understand that a reference voltage for the driver circuit was used with the L297. The reference voltage determines the peak load current for the driver and motors. We decided on the peak load current being 2A and calculated the reference voltage according to that. This reference voltage was not a problem when testing the motor driver PCB using a lab bench power supply that limits the current draw to 1A since we found that both motors turn very smoothly at our desired frequency while drawing around 10-11V and .6-.7A of current. However, when we connected our battery to the driver PCBs, we incorrectly connected the battery in and there must've been a high current draw close to the peak load current of 2A. After doing more research, we found that the L298 cannot sustain 2A of current without having extremely high power dissipation which correlates to high temperatures. We presume the L298s on both chips got fried due to our incorrect wiring of the battery into our PCB. To fix this, we will need to include a battery protection circuit with a current regulator that we neglected to implement, adjust the reference voltage for the L297 to handle peak load currents of 1A rather than 2A, and remember to add the heatsink to the L298 before testing the battery pack. Incorrectly wiring the battery was unfortunate and was caused due to our other PCB with the battery jack not being functional as well. Better caution can be taken next time.

## 4. Cost

We estimate our development costs to be $40/hour. Our team of three people will work 8 hours/week each. So for our 10 week design period, the cost would be:

$$3 \times \frac{\$40}{hour} \times \frac{8\ hours}{week} \times 12\ weeks\ = \$11{,}500$$

Our parts cost $469.14 in total.

*Table 2: Costs*

| Part | Cost |
|---|---|
| ATMega328p | $8.96 |
| ESP8266 (Wifi) x 4 | $15.99 |
| CP2102 (FTDI) x 2 | $14.00 |
| 4 x SEN-11769 (IR Sensor) | $11.80 |
| 2 x HC-SR04 | $19.75 |
| Waste Bin | $20.00 |
| 2 x A4988 Stepper Motor Driver | $11.90 |
| 2 x Geared Bipolar Stepper Motor (RB-PHI-210) | $88 |
| 12V Rechargeable NiMh Battery Pack | $15.00 |
| Battery Pack Charger | $15.00 |

| | |
|---|---|
| L7805 5V voltage regulator | $0.75 |
| Plywood | $20 |
| Wood Glue | $5 |
| Sand Paper | $3 |
| Idler Wheel x 2 | $10 |
| Stepper Motor Mount Brackets | $8.99 |
| LD1117 3.3V voltage regulator | $1 |
| PCBWay PCBs | $100 |
| Surface Mount Parts | $100 |

## 5. Conclusion

The purpose of this project was to create an improved trash can that detects its own trash level and navigates to the door upon detecting fullness. This is to enhance waste management in multi person households since managing and delegating who takes out the trash can be troublesome and lack of action results in foul odors for the entire household.

In our project, we strived to adhere to #1 in the IEEE code of ethics by taking responsibility for any safety and welfare issues that any user of our product has. [11] Since our product aims to be used wide-scale, the large user base increases the risk of safety concerns. We addressed concerns the best that we could and updated our product accordingly as according to #9 in the code of ethics. [11]. We should still add a battery protection circuit to further improve the safety for the users. Furthermore, we also adhered to 1.3 in the ACM code of ethics by always being honest and trustworthy in the construction of our product, analysis of the data for our product, and feedback to users of our product. [12]

Our finished prototype was successfully able to detect trash level and upload that information to a web server for the user to view. However, certain issues with our control unit, power supply, and motor driver prevented us from testing our unit in a non-lab setting. For the control unit, slight modification to the design (moving the clock closer to the MCU) would fix the PCB entirely. Likewise, implementing a safety circuit for our current regulator and changing our VREF value to handle a peak current of 1 A instead of 2 A would allow our motor driver to work as intended. In retrospect, we would have been able to show more results if we had ordered sufficient spare parts.

For future work, we can expand our physical design to support larger trash cans allowing our product to be deployed in suburban settings. Our navigation aspect can be improved by using a variable clock frequency with tape-following or using indoor mapping instead of following a fixed route. Lastly, our web application can be made more robust by adding features such as the capability to configure trash can settings from the web server UI.

# 6. References


[1] "Garbage-- Solid Waste." Garbage-- Solid Waste. N.p., n.d. Web. 09 Feb. 2017.

[2] Vol. 7, No. 1, January 2012, and Issn 1819-6608. FRICTION COEFFICIENT OF RUBBER SLIDING AGAINST FLOORING MATERIALS (n.d.): n. pag. Arpnjournals. ARPN, 1 Jan. 2012. Web. 9 Feb. 2017.

[3] ST, “Positive Voltage Regular ICs,” Nov-2016.

[4] Corporation, Atmel. (n.d.): n. pag. *ATMEGA328p*. Web. 6 Mar. 2017.

[5] "Ultrasonic Sensor - HC-SR04." SEN-13959 - SparkFun Electronics. N.p., n.d. Web. 09 Feb. 2017.

[6] *QRE1113, QRE1113GR Minature Reflective Object Sensor* (n.d.): n. pag. Web. 15 Mar. 2017.

[7] "Stepper Motor." *AccessScience* (n.d.): n. pag. *Stepper Motor Controllers*. Web. 13 Feb. 2017.

[8] Stmicroelectronics. *Dual Full-bridge Driver* (n.d.): n. pag. *Dual Full-Bridge Driver*. Web. 12 Feb. 2017.

[9] “2C." (n.d.): n. pag. *Xx555 Precision Timers*. Web. 5 Mar. 2017.

[10] Singh, Pushkar, and Sanghamitra Saikia. "Arduino-based Smart Irrigation Using Water Flow Sensor, Soil Moisture Sensor, Temperature Sensor and ESP8266 WiFi Module." *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*(2016): n. pag. *ESP8266 Module*. Web. 10 Feb. 2017.

[11] "IEEE IEEE Code of Ethics." *IEEE - IEEE Code of Ethics*. N.p., n.d. Web. 17 Mar. 2017. <http://www.ieee.org/about/corporate/governance/p7-8.html>.

[12] "Code of Ethics." *ACM Ethics*. N.p., 29 July 2016. Web. 15 Mar. 2017. <http://ethics.acm.org/code-of-ethics/>.