

AUTONOMOUS 2D MAP MAKING ROBOT

**FINAL GROUP REPORT**

**Group 2F:**

Sashank Bandemegala

Nicholas DiPaolo

Adam O’Reilly

Chantel Lepage

Luwan Wang

Aaron Chiu



Contents

[1.0 Executive Summary – Sashank Bandemegala 1](#_30j0zll)

[2.0 Introduction – Sashank Bandemegala 2](#_1fob9te)

[3.0 Technical Component Breakdown 2](#_3znysh7)

[3.1 Microcontroller – Sashank Bandemegala 2](#_2et92p0)

[3.2 Power - A](#_tyjcwt)aron Chiu 2

[3.3 Ultrasonic Sensor – Sashank Bandemegala 2](#_3dy6vkm)

[3.4 DC Motor Plug - Nichola](#_1t3h5sf)s DiPaolo 2

[3.5 Autonomous Driving - Luwan Wang 2](#_4d34og8)

[3.6 Bluetooth Module - Adam O](#_2s8eyo1)’Reilly 2

[3.7 2D Mapping - Chantel Lepage 2](#_17dp8vu)

[4.0 Specifications 2](#_3rdcrjn)

[5.0](#_26in1rg) Future Work-Luwan Wang

6.0 Conclusion -Luwan Wang

[References 3](#_lnxbz9)

[Appendices 3](#_35nkun2)

# Executive Summary – Sashank Bandemegala

# Introduction – Sashank Bandemegala

This project is based on an autonomous robot and implementing various sensors and modules in order to achieve the ultimate goal of generating a 2D map of the robots surroundings. A microcontroller is the main component that controls the various parts individually. The microcontroller executes commands that have been input through coding software on a computer. Since the robot has multiple goals, they must have different components to complete each of its goals. Various sensors could be used to achieve the autonomous movement of the robot such as ultrasonic sensors, Radar sensors or camera sensors. They all have a similar process in which the module sends out a pulse of waves and measures the reflection off an object. The information received from the sensors could be analyzed by software to provide the robot with its next move. The movement of the robot involves 3 main components, an h-bridge to allow for forward and backward movement, motors to convert electrical current into mechanical torque, and wheels to use the torque and move the robot. Together, the components work alongside the software for an autonomous robot.

The project that the team has come up with, was an autonomous robot that uses the method above but also include the ability to create a 2-dimensional map of the surroundings. This would all be done autonomously without the need for a human to control where the robot goes. The map making process would be done through software implementation reading the information provided by the ultrasonic sensor through the Arduino microcontroller.

# Technical Component Breakdown

The project being split into two parts with three people in each; hardware and software sides of the team, each member was in charge of one or more components. With each component having different specifications and schematics, they must be researched and made sure they are the perfect fit for the project at hand. Each component must be researched in order to ensure the perfect compatibility with the microcontroller being used. Since the microcontroller is the main brains behind the robot, the components must communicate flawlessly. The microcontroller along with the various components will be explained in detail below.

## Microcontroller – Sashank Bandemegala

The main hardware components that were used were the Arduino microcontroller, motors, motor driver, and the overall body of the robot. These components have been experimented a great deal in order to ensure flawless operations. The main components that have been experimented but changed along the process include the motor driver as well as the body of the robot.

A microcontroller has the ability to take information from sensors such as temperature, humidity, distances, and many other sensors. It can use this information to be used as a trigger to make changes to the environment. For example, if the temperature sensor sends information to the microcontroller that the temperature drops below the minimum set in the programming, the microcontroller will control the heating device to turn it on and heat the area up. Microcontrollers control the external devices it connects through programming language. The programming is very similar to C programming with the way it works. Although all microcontrollers work the same way, there is something different about the JeeNode microcontroller.

In order to choose the perfect microcontroller for the project, its specifications needed to be researched and chosen accordingly. The two options include the Jeenode Microcontroller and the Arduino Nano Microcontroller. Each of the microcontrollers have its own pros and cons. These must be weighed and taken into consideration depending on the projects needs.

## 3.1.1 The Jeenode Microcontroller

The JeeNode microcontroller is a microcontroller that has the ability to be designed for a variety of tasks. The JeeNode microcontroller has a built in Radio Frequency module to allow for information transfer without the need of external devices such as Wi-Fi or Bluetooth modules. This came to the groups interest as the information received by the sensors needed to be transmitted to the main computer to make the 2D map of the robots surroundings. The JeeNode Microcontroller is shown in Figure 2.1 below.



Figure 2.1: JeeNode Microcontroller [2]

The JeeNode has pins, ports, and built-in modules to allow for information transfer. The JeeNode includes a 3.3V power regulator which accepts 3.5V – 13V as an external power source using the 6 pin FTDI- compatible serial I/O port [2]. This power regulator can be seen on the bottom left of the JeeNode. The integrated circuit that can be seen in the center of the JeeNode includes the main microcontroller. The JeeNode includes a RFM12B radio frequency module with a radio antenna which can be seen on the right side of the JeeNode. This module can be used for various things such as; connect to another JeeNode microcontroller, or connect to any device that are able to communicate with 433 MHz , 868 MHz , or 915 MHz ISM bands [2]. The JeeNode also has 4 ports for external devices to be able to connect to it whether it be for information transfer or for the external device to be controlled by the microcontroller.

This microcontroller would have been the perfect fit for this project but since 4 ports are required for just the ultrasonic sensors itself and one port is required for the motor driver, this device has less ports than the ports needed. Since the number of ports seemed to bring an issue to the project, the group decided to switch the microcontroller and use an an alternate microcontroller instead. The other microcontroller that is being looked at is the Arduino Nano.

## 3.1.2 The Arduino Nano Microcontroller

The Arduino Nano is a small and breadboard friendly microcontroller that does the same job of any other microcontroller more or less [3]. The difference between the Arduino Nano and the other microcontrollers that Arduino makes is that it lacks the DC power supply port. But since the Nano is breadboard friendly, it has pins that take in the power supply through breadboard connections. The Nano is given command through the Arduino Software (IDE). The Arduino Nano Microcontroller along with its pin names is shown in Figure 2.2 below.

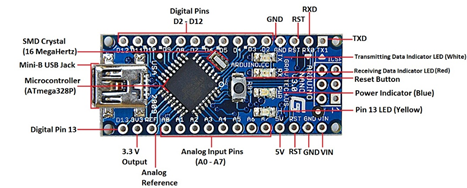


Figure 2.2: Arduino Nano Microcontroller [4]

This microcontroller was chosen over the JeeNode because of the amount of pins available. There are 8 analog IN pins and 14 Digital I/O pins available on the Nano. This gives us a lot of room to work with since each sensor only needs to take up 2 pins on the Nano. Since the Nano will be put into a PCB, the wires can be put in parallel to save space and ports. The wires to power the ultrasonic sensors could be put in parallel to make sure the voltage is the same among all the sensors and the only pins that need to be used on the Nano are the digital pins. The ultrasonic sensor has 4 pins, one for input voltage (Vcc), one for ground (GND), one for the trigger and one for the echo. The trigger and echo pins will be connected to the Nano for information transfer. Using a Nano will save us plenty of ports for other things such as the motor driver. The breakdown of the Arduino Nano is given below.

## 3.1.2.1 Input and Output Pins

Each of the 14 digital pins can be used as an input, they can be controlled to be an input or an output through the programming. The pins all operate at 5V and can provide or receive a maximum of 40mA [3]. In addition to the 8 analog pins and 14 digital pins, some pins have specialized pins such as; receiving and transmitting data, interrupts and PWM outputs.

## 3.1.2.2 Programming

The Arduino does what it does due to its programming, the programming has various functions to do different things much like C programming language. In order to select the pins for input or output, the functions pinMode(), digitalRead(), and digitalWrite() are used [3]. The programming is what is used to determine the robots next step after taking the information from the sensors, determining what to do with the information and outputting it to the motor drivers.

All in all, the best microcontroller for this project is the Arduino Nano microcontroller. This microcontroller has the perfect number amount of port and complies with the power requirements of the robot. The power of the robot will be explained below.

## Power - Aaron Chiu

This project required many different modules and components that works on different supply voltage and current requirements. There are motors which required the greatest voltage at 6v, Ultrasonic Sensors that required 5v, and bluetooth module which requires 3.3v. Below is a simple block diagram of the power connections on this board.

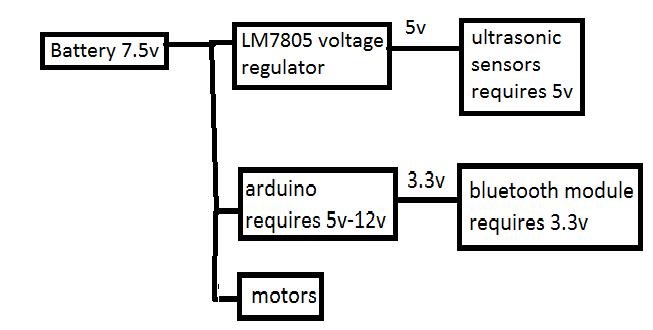


Figure 3.2.1: Simplified block diagram of power connections

The car must be powered with at least the highest voltage required since we did not implement any boost converter or switching power supply which would allow us to power the car at a much lower voltage. starting from the 7.5v battery, we connect the arduino, motors, and the voltage regulator. The voltage regulator outputs a stable 5v which is used to power the sensors. The arduino itself is has a 3.3v regulator which is used to power the bluetooth module.

## 3.2.1 Battery

A 7.5v battery was chosen since it was not to much greater then the motors recommended supply voltage of 6v, this reduce the chances of burning our motor coils. Initially testing was done using a 6v battery however a voltage drop of less than 1.25v was present across the input and output terminals of the voltage regulator which resulted in an unstable 5v at the output of the voltage regulator. Five double a batteries were connected in series which gave us a total of 1.5v\*5=7.5v shown below.



Figure 3.2.1.1: 7.5v Battery pack used

A Battery pack was used to allow for easy replacement of the batteries when they died. During testing, many batteries were used since the motor required a lot of current.

## 3.2.2 LM7805 Voltage Regulator

To power the sensors, an LM7805 voltage regulator was used. This type of voltage regulator was chosen since is outputs a constant 5v and the only supporting components needed were capacitors. From the datasheet, the LM7805 voltage regulator has a peak current of 2.2A, which was more than enough to be used to power our sensors. The HC-SR04 sensors we used had a working current of 15mA each. In total the 4 sensors working at the same time had an average current draw of 60mA, which is under our 2.2A limit. The recommended design from the datasheet of the Lm7805 voltage regulator shown below, was put together on a breadboard to allow for testing before implementation into our circuit board.

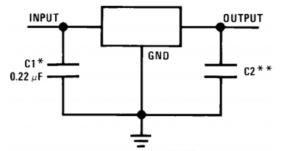


Figure 3.2.2.1: LM7805 Recommended Circuit

C2 is not necessary however, it is still used to allow for a more stable output voltage. Initially, the circuit was designed to use a 6v battery pack. For C1 we used a 0.22uF capacitor as recommended by the datasheet, For C2 started off with a 10uF electrolytic capacitor. However, when the power supply was connected to the sensors, the voltage regulator became extremely unstable. The voltage regulator output 1v which made the sensors give us random data that did not correspond to the distance for the object that was in front of it. The problem persisted for 2 weeks until we realized that for optimum performance, the LM7805 voltage regulator needed at least a 1.25v drop between the input and output terminals. Since the battery used was only 6v the max voltage drop across the input and output terminals were 1v, and as the voltage on the battery was reduced due to it being consumed, the output voltage dropped below 1v. the fix used was to add 2 more 10uf capacitors in parallel with C2.

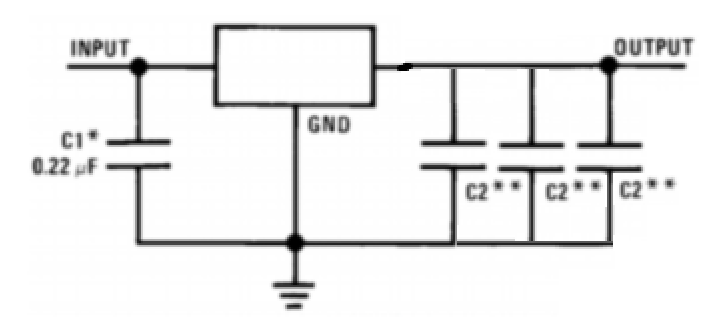


Figure 33.2.2.2: voltage regulator circuit implemented on the board

This allows the output side of the voltage regulator to supply more current since the capacitors act as a storage device and would supply the sensors with more current and kept the voltage regulator at an optimum state. more testing was done to ensure that the fix was actually working and it was then built on the circuit board. This fix worked for awhile until It was observed that when the the robot was moving, the sensors would output random values, and only when the motors were stopped that the sensors were operating normally again. The problem that was found was that every time the motors switched on, it would starve the LM7805 voltage regulator of current since the motors are directly connected to the battery. This would produce the same result as before giving us approximately 1v at the output of the voltage regulator and force the sensors to operate improperly. This problem persisted for another 2 weeks and eventually it was fixed by adding another double a battery which gave a total of 7.5v to power out circuit. The motors were still able to handle this voltage with no new issues.

## 3.2.3 Arduino

The arduino was directly connected to the battery since it has its own voltage regulator and and handle up to 12v between its Vin and Gnd pins.

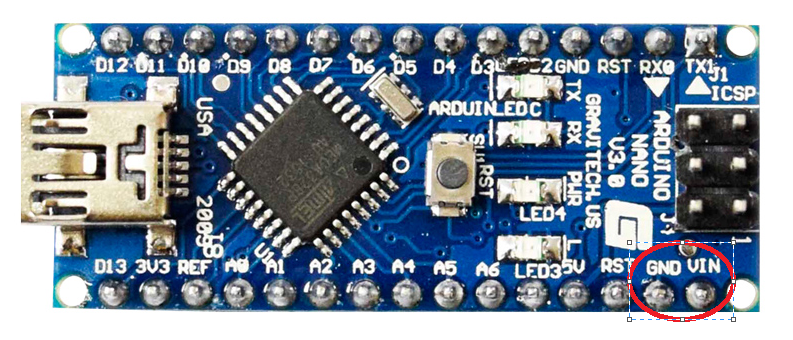


Figure 3.2.3.1: Vin and Gnd pins on arduino nano rated for up to 12

This method was chosen instead of connecting the arduino to the voltage regulator because it reduces the load on the voltage regulator preventing it from having anymore instability issues. Also, the arduino is isolated from the voltage regulator because if the voltage regulator became unstable again, a 1v supply voltage to the arduino would no be enough to power it. If the arduino were to shut down, all control of the robot would be lost. The arduino conveniently also had a 3.3v supply pin. The bluetooth module was then able to be connected to it with no issues.

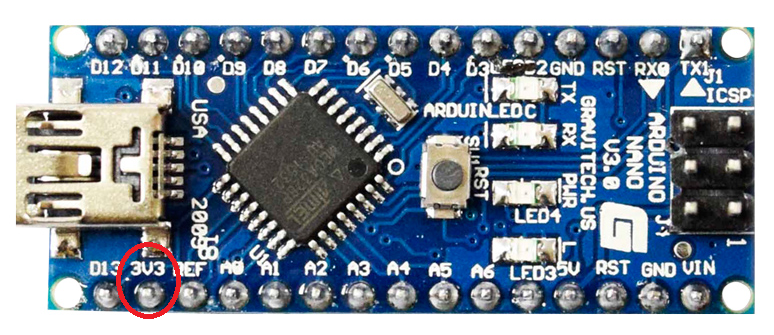


Figure 3.2: 3.3v pin on arduino nano

## Ultrasonic Sensor – Sashank Bandemegala

An Ultrasonic sensor is a device that can measure the distance to an object by using sound waves. The sensor emits a wave at a specific frequency and listens for the same sound wave to bounce back. The elapsed time between the sound being emitted and received determines the distance between the sensor and the object.

The ultrasonic sensors will be used on the robot to allow it to avoid obstacles. Once connected to the Nano and positioned correctly on the robot, the ultrasonic sensors will return a value depending on the distance of the obstacle in front of it.

3.3.1 Operation of Ultrasonic Sensor

The ultrasonic sensor consists of two main components, an echo

3.3.2 Breakdown of Ultrasonic Sensor

The ultrasonic sensor has 4 pins, one for input voltage (Vcc), one for ground (GND), one for the trigger and one for the echo. The trigger is the pin that is used to emit the wave and echo pin is used to receive the information. The two will be connected to 2 pins on the Nano for information transfer. The ultrasonic sensor’s pins are shown in Figure ## below.

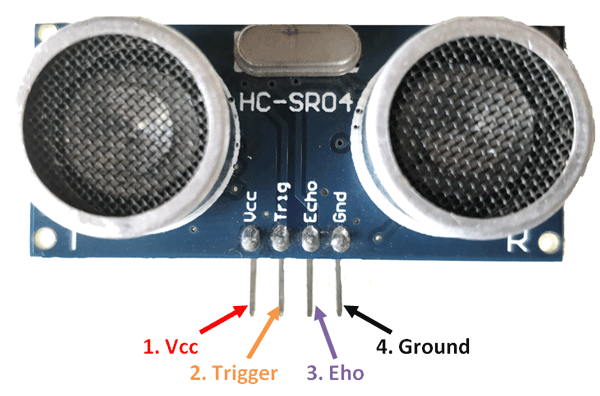


Figure ##: Schematic for Connecting an Ultrasonic Sensor to the Nano [5]

The Vcc pin is an input pin that needs a 5V supply in order for it to perform its function. This pin is connected directly to the Vcc pin of the Arduino. The trigger pin is an input pin that takes a digital signal and uses that to send a pulse of sound out. The echo pin is an output pin that receives the trigger pulse as an input and outputs the corresponding distance value. The ground pin is to close the loop and allow the ultrasonic sensor to work. The working current of the ultrasonic sensor is 15mA and has a range of 4 meters so this would be the perfect component for the project.

The wires to power the ultrasonic sensors could be put in parallel to make sure the voltage is the same among all the sensors and the only pins that need to be used on the Nano are the digital pins. The schematic for connecting one ultrasonic sensor is shown in Figure 2.2 below.

Figure 2.2: Schematic for Connecting an Ultrasonic Sensor to the Nano [5]

Using the schematic above, it can be seen that only 2 pins are being used for connecting the ultrasonic sensor. Two digital pins are being used, one for the trigger and one for the echo. The information can then be used in the programming to determine its next movement.

### **1.1.1 Ultrasonic Sensor Programming**

Once the sensor is connected and positioned correctly, the output of the sensors could be measured numerically, if the output is less than a specific number, the Arduino can determine the next move. For example, if the programming can be written to match the dimensions of the body of the robot, if a certain number for the output is found, the robot can turn left or right a certain degree to avoid the obstacle.

After the definitions like done in C programming, the main code can start. The initializing void function is given below.

*void setup()*

*{*

*// Serial monitoring*

*Serial.begin(9600);*

*// Initializing Trigger Output and Echo Input*

*pinMode(TRIGGER\_PIN, OUTPUT);*

*pinMode(ECHO\_PIN, INPUT);*

*// Reset the trigger pin and wait a half a second*

*digitalWrite(TRIGGER\_PIN, LOW);*

*delayMicroseconds(500);*

*}*

This code is to initialize the USB port as well as state the trigger pin on the sensor is the output and the echo pin is the input. The next step is to create a loop to trigger the sensor to take measurements.

*void loop()*

*{*

*delay(MEASURE\_DELAY);*

*long distance = measure();*

*Serial.print("Distance: ");*

*Serial.print(distance);*

*Serial.println(" mm");*

*}*

This code is to simply take measurements and output the values as Distance: ## mm. The next part of the code is to take the measurements and record them, the measure method is used to sample the measurements. Since the sensors work with an equation, it must be input into the programming. The equation is:

*long measure()*

*{*

*long measureSum = 0;*

*for (int i = 0; i < MEASURE\_SAMPLES; i++)*

*{*

*delay(MEASURE\_SAMPLE\_DELAY);*

*measureSum += singleMeasurement();*

*}*

*return measureSum / MEASURE\_SAMPLES;*

*}*

Once all this code is input into the IDE software, the output of the information in numerical value is given. The programs output is given as:

Distance: 90 mm

Distance: 90 mm

Distance: 89 mm

Distance: 92 mm

Distance: 90 mm

Distance: 197 mm

Distance: 207 mm

Distance: 1371 mm

Using this numerical information, the Nano can decide what to do as the next movement and control the motor drivers as such. If the value is low, this means the distance is very small and the robot is approaching an obstacle.

## DC Motor Driver - Nicholas DiPaolo

There are many different ways to drive a DC motor, rather its through our own half/full H-Bridge, motor plug, or an IC motor driver. All possibilities were evaluated and it was determined to use an IC motor driver. The reason the other options weren’t good for our project was because, if we built our own H-Bridge there would be too much power consumption and not cost efficient due to the MOSFETS needed. The other option of using a DC motor plug would have worked until the team decision to switch to the Arduino Nano instead of the JeeNode, the motor plug wouldn’t work with the Nano since the plug was made to be used with the JeeNode.

**3.4.1 H-Bridge**

A H-Bridge is a circuit that consist of transistors that allow for current to travel in either direction.

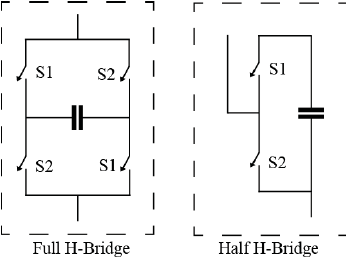


Figure 1 Full vs Half H-Bridge [1]

[1]<https://www.researchgate.net/figure/Figure-A1-Full-and-half-H-bridge-configuration_fig8_260741710> reference for the picture

There are two kinds of H-Bridges, a half and a full H-Bridge, with a full H-Bridge consisting of two half’s (figure… ). The advantages of the full bridge it will allow for a AC signal to pass through, well a half bridge will clip and only half the sine wave will be passed. For our project, one half bridge would have been go enough for each motor, but as a team we decided on 1 full bridge for each motor.

### **3.4.2 SN754410**

The IC that was used was the SN754410, the SN754410 is an IC that consist of 4 half H-Bridges.

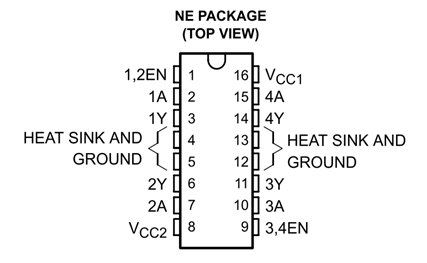


Figure 1 Pin Alloactions for the SN754410 [2]

[2]<http://www.ti.com/lit/ds/symlink/sn754410.pdf>

The VCC2 pin is the drivers power, were the VCC1 is a 5V supply for the internal logic translation, the driver power can range from 4.5V to 36V. Each side of the chip has connections for 2 half bridges, with 1 and 2 sharing a ground and power supply and 3 and 4 sharing theirs. The chip also comes with pin allocations for a heat sink just in case the chip is consuming a lot of power, but for this project those pins will not be used. The SN754410 allows us to control the motors with low power as well as small area, giving us more room for other components on the vehicle.

### **3.4.3 On Board Connection**

When connecting the motors to the SN754410, we allocated 2 half bridges to 1 DC motor making a full H-Bridge to control each motor. This will allow us to control the motors separately, giving us the ability to turn left or right by having one wheel go forward and the other go backwards. The power supply for the chip was 4 AA batteries that provided 7 Volts to the driver.

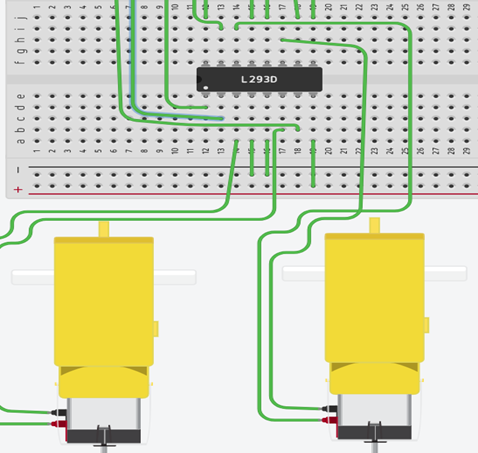


Figure: Connections made on the prototype board (different chip used on simulation, acts the same as the SN754410)

## Autonomous Driving - Luwan Wang

An autonomous driving vehicle is a vehicle that guides itself without human input. The Autonomous driving function in this project drives an Arduino nano to control the motors to avoid obstacles based on measurements from an ultrasonic sensor. The autonomous driving software was programmed using Arduino IDE. Four ultrasonic sensors have been incorporated to make both navigation decisions and to generate 2D maps. The distance information from all four sensors are transmitted to the central computer to generate a 2D map, while the navigation decision relies on only the front ultrasonic sensor

**3.5.1 Logic functions overview**

The block diagram in Figure 3.5.1.1 shows the schematics of the implementation of autonomous driving comprising of different functions in software.

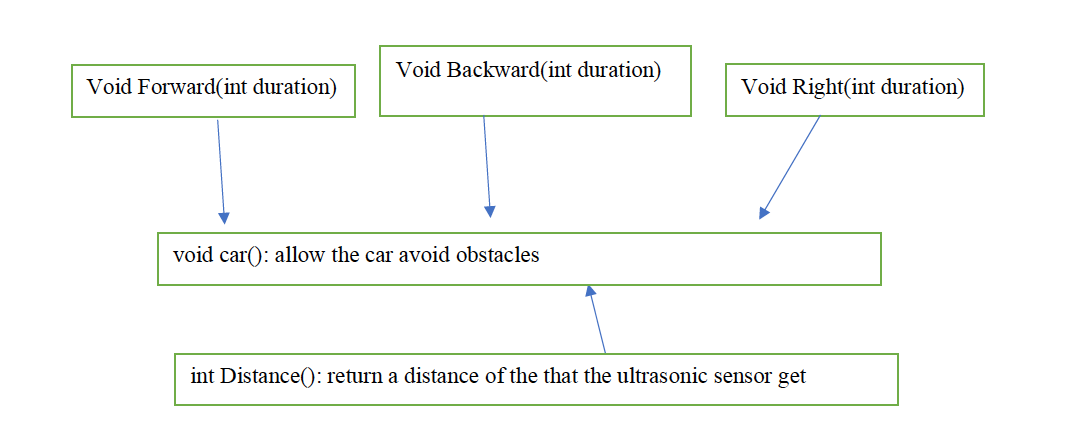


Figure 3.5.1.1: Block diagram of the autonomous driving software implementation

As shown in the block diagram in Figure 1, there are five functions that control the motors. The direction of the arrow points to the calling function. The functions are:

*Forward( int duration )* takes in an integer and control the vehicle to move forward for *duration* milliseconds.

*void backward( int duration )* takes in an integer and control the vehicle to move backward for *duration* milliseconds.

*void right ( int duration )* takes in an integer and control the vehicle to turn right for *duration* milliseconds.

*int Distance (int echoPin) takes* the echoPin from an ultrasonic sensor in different position and returns the distance from the sensor to the obstacle.

*void car()* controls the car to keeps moving forward if the distance to the to the front obstacle is larger than 30cm. Else the robot keeps backing up until the distance to the front sensor is larger than 30cm and turn right to avoid the obstacles.

**3.5.2**  **Motor control**

The car can be can be controlled to move forward, backward, and turn left or right by the actuation of the motors based on the command from the motor control function.

An H bridge was connected in the circuit to allow the motor to rotate both directions. Digital pin 2 and 3 was set to control motor A, digital 4 and 5 to control motor B as shown in Figure 3.5.2.1 below.

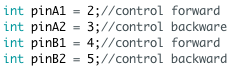


Figure 3.5.2.1: Motor set up code

Several functions are required to control the motors to rotate forward or backward and also to stop by setting the respective pins to high or low as shown below in Figure 3.5.2.2. The hardware H bridge theory behind this has been discussed section 3.4.

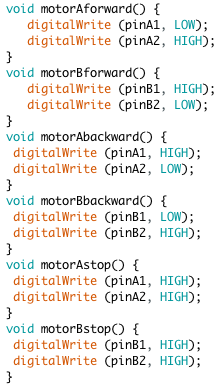


Figure 3.5.2.2: Single motor control functions

The digitalWrite (pin, value) is used to write a high or low value to a digital pin[1]. For example, in void motorAforward() function, it writes PinA1( pin 2) to low and Pin A2( pin 3) to high. Pin 2 and 3 are the pins from the H bridge which controls motor A to roll clockwise. A similar theory can be applied to other single motor control functions. When both A1 and A2 are both pulled high, motor A stops. Single motor control functions are used for the vehicle movement control as shown below.

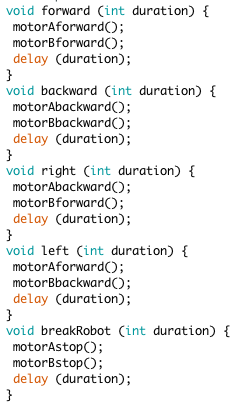


Figure 3.5.2.3: Vehicle movement control function

In Figure 3.5.2.3, the vehicle movement control function enables both motors spin clockwise in order to control the vehicle to move forward. Similarly, both motors spin counterclockwise to control the vehicle to roll backward. The logic is similar for other movement control functions. delay (duration) allows the Arduino to control the vehicle to run relative function for *duration* ms.

The vehicle movement functions were implemented and tested in an infinite loop as shown in Figure 3.5.2.4, by putting all the functions in the loop and pass in a 500 ms duration, the robot moved forward, backward, left, right and stopped each for 500ms.

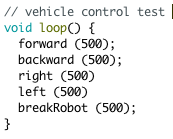


Figure 3.5.2.4: Testing of Vehicle Control Functions

**3.5.3**  **Calculate the distance from the obstacle to the ultrasonic sensors**

Ultrasonic sensors can be used to calculate distance from the obstacle to the vehicle. There are four ultrasonic sensors in this autonomous 2d mapping robot. In order to simplify the circuit and the code, all ultrasonic sensors share the same trigger pin, since the four sensors are facing left, right, back and right respectively, the signal from different sensors will not likely be overlapping to affect the distance measurement.

The set-up code is as shown below in Figure 3.5.3.1. All the trigger Pins are connected together to pin 7. Digital pin 8 is connected to echoPin of the front sensor, pin 9 is connected to echoPin of the left sensor, pin 10 is connected to echoPin of the right sensor, pin 11 is connected to the echoPin of the back sensor.

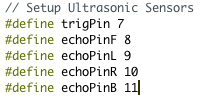


Figure 3.5.3.1: ultrasonic sensor setup

In order to calculate the distance from the ultrasonic sensor to the obstacle, the following function is written. The distance (int echoPin) function in Figure 3.5.3.2 takes in the echoPin of different sensors and returns the calculated the distance from the obstacles respect to that sensor. First, it pulls the trigger pin high to allow it to send the sound wave and delay for 10ms then pulls the trigger pin low. pulseIn(echoPin, High) times the duration of the echoPin to go from high to low [2] which mean the time of receiving the reflected signal. Then Arduino waits for 70 ms to make sure all the reflected signal are gone. A formula for calculating the distance (D= (V\*T)/2) is used the end of this function to give the distance output.

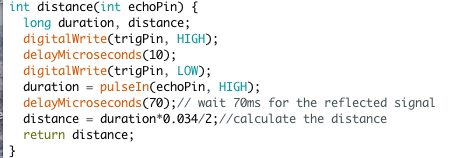


Figure 3.5.3.2: The Distance function [1]

The distance of any sensor can be calculated by calling this function with the respective echoPin passed in. This function can be used repeatedly and thus increases the efficiency of calculation. The distance function is used for autonomous driving and also for obtaining the distance data for 2d mapping.

**3.5.4 Avoid obstacles**

One of the goals of autonomous driving is obstacle avoidance, when the vehicle “sees” the obstacle, it moves backward and turns right to avoid obstacles.

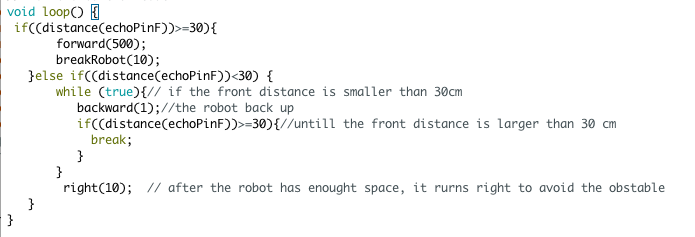
****

Figure 3.5.4.1: Logic Code for Avoiding Obstacles

The code above illustrates how the robot avoids the obstacles. The robot checks if the distance to the front sensor is larger than 30 cm. If it is larger than 30 cm which means there is no obstacle in front of the robot, the robot goes forward for 500 ms and pause for 10 ms. As it is in a loop, it will keep doing this action. The robot has to pause for 10 ms, otherwise, since the robot is moving too fast, it will hit the obstacles before receiving the reflected signal. Having the robot to stop for 10ms gives the robot more time for measuring the distance.

When the distance from the obstacle to the front sensor smaller than 30 ms which means that there is an obstacle in front of the robot, it will keep backing up until the distance to the obstacle is larger than 30cm. After that, The robot turns right to avoid the obstacle.

10 sets of experiments were conducted to determine the best distance to have the vehicle back up before it hits the obstacle. It has been found that 30 cm has the highest number of success.

This experiment is to compare what value of cm is the best distance to set to avoid obstacles. Each distance value was tested 10 times. The autonomous vehicle was released for 20 seconds, and the number of different performance was recorded as shown below in Table 3.5.4.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Distance | times of being trapped | times of hitting on obstacles | times of perfectly avoid obstacles |
| 10 cm | 1 | 5 | 4 |
| 20 cm | 1 | 3 | 6 |
| 30 cm | 1 | 1 | 8 |
| 40 cm | 3 | 1 | 6 |

Table 3.5.4.1: Experiment of setting avoid distance.

As shown in Table 3.5.4.1, when the distance is set to 10 cm, the vehicle cannot move back fast enough to avoid hitting on the obstacle(5 times out of 10 experiments). While when the distance increased to 20 cm, the performance of the robot is better as it perfectly avoids obstacles for 6 times out of 10 experiments. The percentage of success reached 80% when the distance is set to 30 cm which is the value programmed in the final project code. As the distance increased to 40 cm, the vehicle was trapped more. If the surrounding environment space is small then the vehicle would keep backing up without turning, therefore it cannot see the exit and got trapped.

After comparison, it has been decided to set the vehicle to avoid the obstacle when it is 30 cm away from the obstacle.

**3.5.5 Send serial output data for mapping purposes.**

The following code in FIgure 3.5.5.1 is used to send the distance information to a laptop for mapping purpose. The distance from the obstacle to different sensors is stored in variable F, R, L, B respectively. Information is printed by using Serial.println(). The purpose of using serial.flush() is to clean all the data before sending out the data in the next iteration.

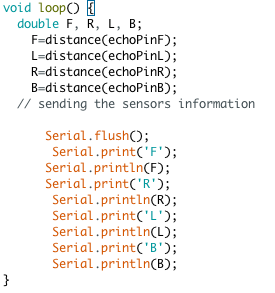


Figure 3.5.5.1 : Printing information

The Bluetooth module will send out the printed data to a laptop with Bluetooth function. The data is printed on serial monitor on the Bluetooth paired device. Then the data is taken into Matlab for mapping purposes.

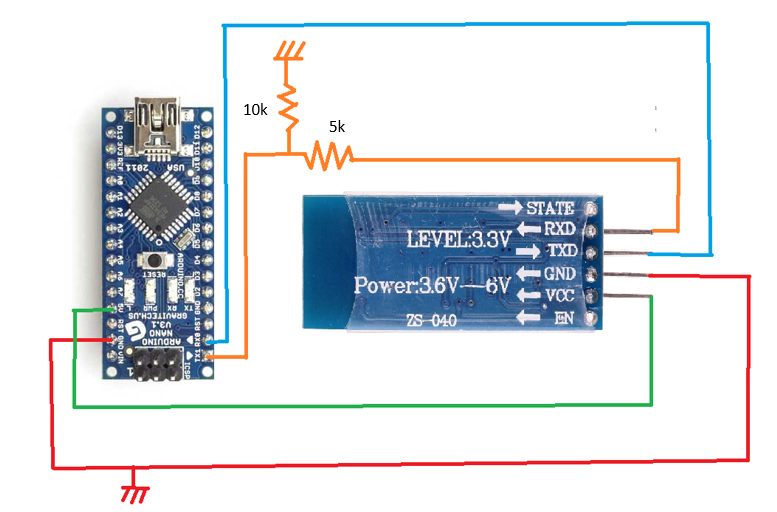
## Wireless Communication - Adam O’Reilly

In order to be able to create the map the car needed to be able to communicate to a computer wirelessly in order to send the readings of the ultrasonic sensor. The group thought of two options to be able to accomplish this communication. One option was to use wifi and the other is to use Bluetooth communication. The positive for wifi is that it works over the internet and therefore it can work from any distance away with internet connection but the negative is it would be quite complicated to set up because a server and clients would need to be added. The plus for Bluetooth is it is quite simple to add to the project since there is already library’s online for the module to work properly but its downfall is they have a fairly small distance at to which it can work at, for example like a 10 meter range. Taking all of this into account the group decided to go with the Bluetooth module as we were under a time constraint so to finish on time the simpler option would be preferred if possible and for our purposes to make a prototype the range of 10 meters would not be a problem.

### **3.6.1 – HC-06 Bluetooth Module**

The Bluetooth module the group selected was HC-06 as it is not very expensive and easy to use. It only works with a 9 meter range so when testing out our robot we just have to stay within 9 meters of the laptop for the mapping to work, which is not a problem for our purposes. Connecting it to the nano is not very difficult either, HC-06 has only 4 pins which are: RXD, TXD, VCC, and GND. The GND pin simply goes to ground and the Vcc pin can take between 3.6V to 6V so connecting it to the 5V pin of the nano works perfectly. The RXD is for transmitting and TXD is for receiving, this means the RXD of the Bluetooth module will be connected to the TXD of the Arduino since when the Arduino is transmitting the Bluetooth is receiving it. Similarly, when the Bluetooth is transmitting data it will be sending it to the Arduino from the TX of the Bluetooth to the RX of the Arduino. The only issue with these connections are the TX of the Arduino sends out 5 volts but the RXD on the Bluetooth can only take in 3.3V so a voltage divider is needed. Using a 10k resistor and 5k resistor, the 5 volts can be converted to 3.3 volts and sent to the Bluetooth’s RX without damaging it, shown by:

The connections are shown below:

Fig##: Circuit connections for Bluetooth module

## 2D Mapping - Chantel Lepage

2D mapping is used to create a basic floor plan of a room. The map would show obstacles that the vehicle would have to navigate around, and the map would display any walls that the vehicle came near. The 2D map will be outputted in MATLAB. The information the that would create the map comes from the ultrasonic sensor readings. These readings then get passed through from the Arduino to the Bluetooth chip then get read into MATLAB.

The code that was written for this project was altered from the original idea. Instead of 2D mapping the code was changed to display a live update of the distance of an object on each side of the vehicle. The code for this can be seen below in Appendix C.

**3.7.1 Reading Values and Converting**

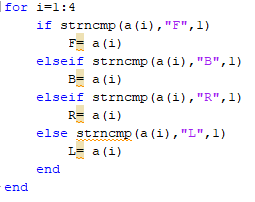
The code below allows for the communication between the Arduino and the MATLAB program. The first line sets up the parameters of the Arduino, COM7 defines what port the data will be coming from, which in this case COM7 is the Bluetooth module, and the BaudRate that is defined is the speed in bits per second that data is sent to the communication port which is 9600 bits per second.

The next line of code opens the port that allows for the values sent from the Arduino to be read into MATLAB. The last line of code in this figure allows for the data to stop being read into MATLAB.





Since the values get read into MATLAB as a direction with a value in a string format the string had to be parsed and then converted from a string value to an double value. The string values that were passed into MATLAB were first set into an array and sorted by their direction using a string compare.



After the inputs were sorted they where then parsed from the direction defining component of the value using a strsplit function. Below the f parameter is now what will hold the string value of the distance and be split from the direction defining component of F.

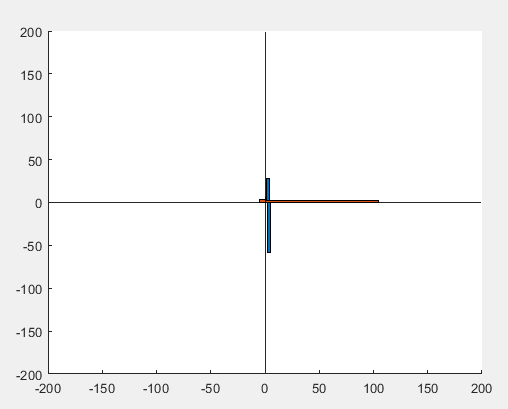


Next the values are cast in to doubles so that they can plotted the str2double function allows for this to be easily done.



**3.7.2 Plotting the Graph**

Finally, the last step was to plot the results. This was done by plotting the left and right values horizontally and the front and back values vertically. And exam of the plot can be seen below.



The in the figure above the point (0,0) represents the car, and the bars away from it represent how far an object is to the car.

To make the graph update to show the latest reading the code was placed in a while loop and the function below where used. The clf function allows for the graph to cleared without closing the figure, and the drawnow function allows for the plot to update and display the new readings.





# Specifications +

The overall cost of the project is only 68.99 which is summarized in Table A below. The cost is reasonable for an autonomous 2d map making vehicle.

|  |  |
| --- | --- |
| **Required Parts** | **Cost** |
| Rubber Wheels | 2 @ $2.20 each = $5.20 |
| Small Wheel | 3D printed = $0 |
| Ultrasonic Sensors | 4 @ $4.39 = $17.96 |
| Motors | 2 @ $3.50 = $7.00 |
| Prototype Board | 2 @ $1.92 each = $3.84 |
| Arduino Nano | $9.99 |
| Dual H bridge | $5.00 |
| Bluetooth module | $10.00 |
| Power supply | $10.00 |
| **Total** | **$68.99** |

Table A: Cost for all components

The minimum operating voltage is 6V and the maximum operating voltage is 7.5V. The microcontroller used in this robot is Arduino Nano as it is cheap as while as having enough digital pins for this project. For autonomous driving part, 80% of the time that the robot can avoid the obstacles successfully. This yield is acceptable for this project but not for real life application. More investigation should be done to increase the yield. For 2d mapping part, the time to refresh the 2d mapping is approximately 3 seconds. Similarly, 3 seconds is acceptable for this project but not for real life application because the vehicle can move 0.3m/s \* 3 s = 0.9 m which is large distance. The batteries can last for approximately 2 hours to support the robot when the batteries are in full charge. The specifications are summarized in Table B below.

|  |  |
| --- | --- |
| Operating Voltage | 6.0v min, 7.5v max |
| Microcontroller | Arduino Nano |
| Yield | 80% |
| Mapping time | 3 seconds |
| Moving Speed | 0.3m/s |
| Battery life time | 2 hours |
| Cost | $ 58.99 |

Table B: Robot Specifications

The robot is easy to use for a client. The robot can be switched on by plugging in the battery pack. The autonomous driving code was loaded in the Arduino nano, thus the robot runs autonomously once it is powered on. Simply pair up the Bluetooth module with a laptop (with the Matlab code load in), the 2d map will automatically pop up and refresh after hitting the run button in Matlab. This robot could be used for rescuing, smart vacuum cleaner etc. The 2d mapping accuracy and yield still needs to be improved for real-life application.

# Future Work-Luwan Wang

Autonomous driving and real-time 2d mapping were successfully tested on the 2d autonomous mapping robot.Several functions have been developed in this project, these functions can be improvements to enhance the robot with more capabilities.

Firstly, the initial goal is also to add a camera module to give a real-time feedback to the user. Owning to the inefficient time, the camera module was not implemented.

Secondly, 80% success to perfectly avoids obstacles is not high enough for a real autonomous driving vehicle. This is due to the blind spots for the autonomous vehicle. The ultrasonic sensor only has a range of 15 degrees, which is not enough to cover the whole front view of the vehicle. Therefore, adding two more ultrasonic sensors on the left corner and right corner can be helpful in eliminating the blind spots and increase the yield.

Thirdly, there are only four data points that are taken for mapping. The real-time 2d mapping can be more accurate if a swiveling motor with an ultrasonic sensor is added to the robot to record more data.

Lastly, another perspective of the mapping is to create a map of the environment instead of a real-time map. For example, if the robot is placed in a maze then it should be able to create the map of the maze. In this project, no GPS or digital compass was used which means the robot cannot record its path or the direction it turned. More investigation and experiment can be done in this area.

# Conclusion- Luwan Wang

The goal of this autonomous 2D map making robot is to have a robot vehicle with the ability of autonomous driving and making a real-time 2d map of its surroundings. Four main processes were done to accomplish the goal: motor control, autonomous driving, Bluetooth module implementation, and 2D mapping. Even though all the main functions are achieved in this project, there are still some functions that needs to be improved such as blind spot problem, inaccurate mapping. An image of the completed project is shown below in Figure 5.1.

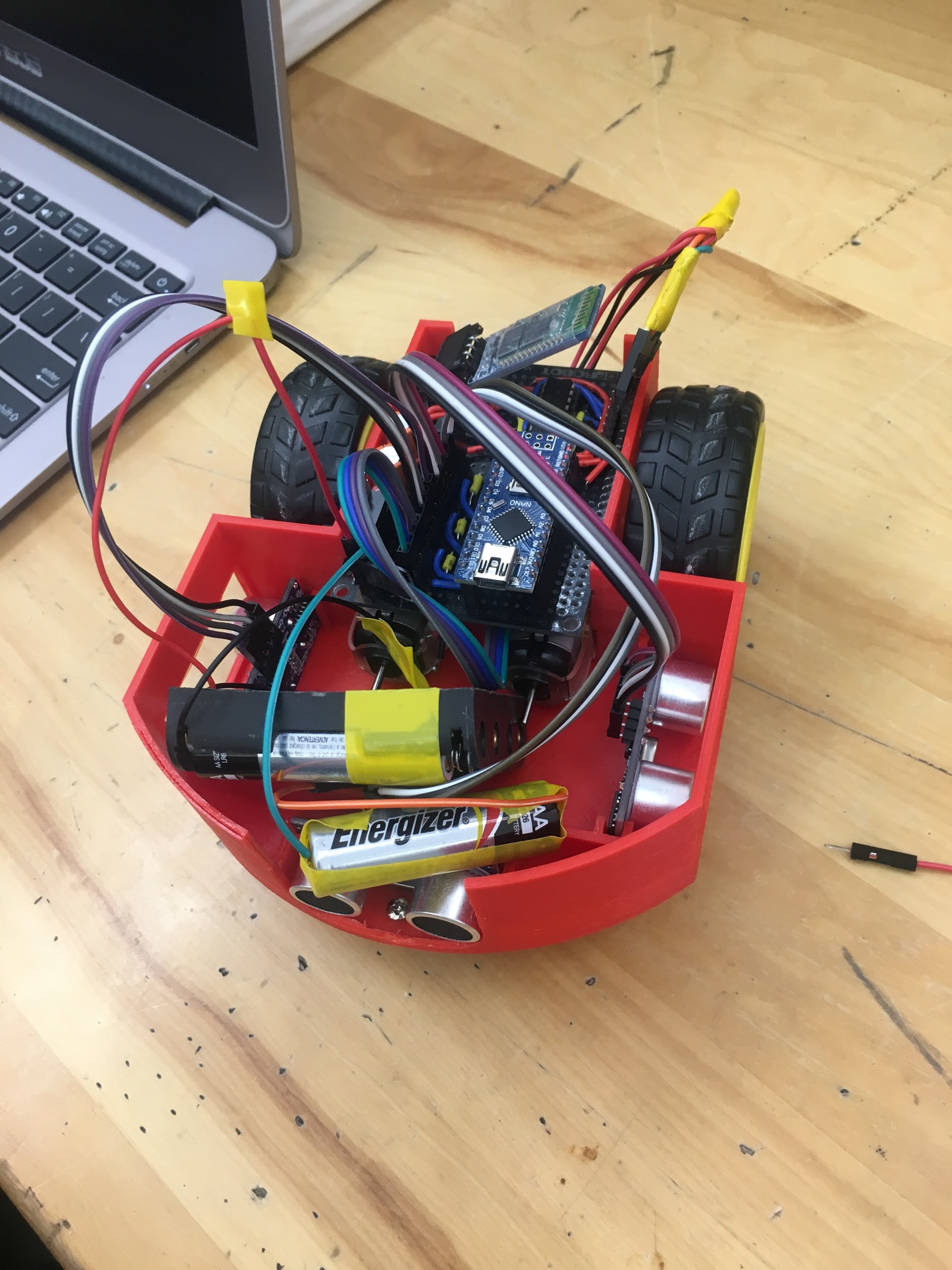


Figure 5.1 Autonomous 2d Map Making Robot

# 

# 

# 

# References

Section 3.5 [Autonomous Driving](#_4d34og8):

[1]"Arduino Reference", *Arduino.cc*, 2018. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/. [Accessed: 10- Apr- 2018].

[2] "Arduino Reference", *Arduino.cc*, 2018. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/. [Accessed: 11- Apr- 2018].

# 

# 

# Appendices

# Appendix A: Agreed assessment

The agreed percentage contribution of each member to the group is shown in the table below.

|  |  |
| --- | --- |
| Name | Percentage |
| Sashank Bandemegala |  |
| Nicholas DiPaolo |  |
| Adam O’Reilly |  |
| Chantel Lepage |  |
| Luwan Wang |  |
| Aaron Chiu |  |

# 

# Appendix B: Autonomous Driving Arduino Code

# #include <Wire.h>

# int pinA1 = 2;//control forward

# int pinA2 = 3;//control backware

# int pinB1 = 4;//control forward

# int pinB2 = 5;//control backward

# // Setup Ultrasonic Sensors

# #define trigPin 7

# #define echoPinF 8

# #define echoPinL 9

# #define echoPinR 10

# #define echoPinB 11

# 

# void setup() {

# // The setup code goes here and runs once only

# // Configure the pin modes for each drive motor

# // set up motor A

# pinMode (pinA1, OUTPUT);

# pinMode (pinA2, OUTPUT);

# // set up motor B

# pinMode (pinB1, OUTPUT);

# pinMode (pinB2, OUTPUT);

# 

# // Configure the pin modes for the Ultrasonic Sensor

# //Front sensor set up

# pinMode(trigPin, OUTPUT);

# pinMode(echoPinF, INPUT);

# //left sensor set up

# pinMode(trigPin, OUTPUT);

# pinMode(echoPinL, INPUT);

# //right sensor set up

# pinMode(trigPin, OUTPUT);

# pinMode(echoPinR, INPUT);

# 

# //back sensor set up

# pinMode(trigPin, OUTPUT);

# pinMode(echoPinB, INPUT);

# 

# //back sensor set up

# Serial.begin (9600);

# 

# }

# void loop() {

# // Main code goes here and will run repeatedly:

# car(); // function that allows the robot to avoid obstacles

# 

# }

# // Create motor control functions

# void motorAforward() {

# digitalWrite (pinA1, LOW);

# digitalWrite (pinA2, HIGH);

# }

# void motorBforward() {

# digitalWrite (pinB1, HIGH);

# digitalWrite (pinB2, LOW);

# }

# void motorAbackward() {

# digitalWrite (pinA1, HIGH);

# digitalWrite (pinA2, LOW);

# }

# void motorBbackward() {

# digitalWrite (pinB1, LOW);

# digitalWrite (pinB2, HIGH);

# }

# void motorAstop() {

# digitalWrite (pinA1, HIGH);

# digitalWrite (pinA2, HIGH);

# }

# void motorBstop() {

# digitalWrite (pinB1, HIGH);

# digitalWrite (pinB2, HIGH);

# }

# // Setup movement functions

# void forward (int duration) {

# motorAforward();

# motorBforward();

# delay (duration);

# }

# void backward (int duration) {

# motorAbackward();

# motorBbackward();

# delay (duration);

# }

# void right (int duration) {

# motorAbackward();

# motorBforward();

# delay (duration);

# }

# void left (int duration) {

# motorAforward();

# motorBbackward();

# delay (duration);

# }

# void breakRobot (int duration) {

# motorAstop();

# motorBstop();

# delay (duration);

# }

# // Setup Ultrasonic Sensor distance measuring

# int distance(int echoPin) {

# long duration, distance;

# digitalWrite(trigPin, HIGH);

# delayMicroseconds(10);

# digitalWrite(trigPin, LOW);

# duration = pulseIn(echoPin, HIGH);

# delayMicroseconds(70);// wait 70ms for the reflected signal

# distance = duration\*0.034/2;//calculate the distance

# return distance;

# }

# // Setup the main car function

# void car() {

# 

# double F, R, L, B;

# F=distance(echoPinF);

# L=distance(echoPinL);

# R=distance(echoPinR);

# B=distance(echoPinB);

# // sending the sensors information

# 

# Serial.flush();

# Serial.print('F');

# Serial.println(F);

# Serial.print('R');

# Serial.println(R);

# Serial.print('L');

# Serial.println(L);

# Serial.print('B');

# Serial.println(B);

# /\* Compare if the front sensor is larger than 30 cm, if it is

# \* if it is then move forward for 500 ms and pause 10ms and keep moving for 500ms and again

# \*

# \*/

# 

# if((distance(echoPinF))>=30){

# forward(500);

# breakRobot(10);

# }else if((distance(echoPinF))<30) {

# while (true){// if the front distance is smaller than 30cm , the robot back up untill the front distance is larger than 39 cm

# backward(1);

# if((distance(echoPinF))>=30){

# break;

# }

# }

# right(10); // after the robot has enought space, it rurns right to avoid the obstable

# }

# 

# 

# 

# }

# // Go back and turn slightly right to move car in new direction

# // This function only runs if an obstacle within 15cm is detected

# 

# 

# 

# Appendix C: Matlab code for 2d mapping

# 

clear all

clc

a=["0" "0" "0" "0"];

b=["0" "0" "0" "0"];

%creates parameters for the ardiuno to connect to matlab

arduino=serial('COM7','BaudRate',9600);

fopen(arduino); %opens connection with ardiuno

while true

for i=1:4

a(i)=fscanf(arduino,'%d');% stores scaned in values

end

%compares the scaned in string values so that they can be sorted as the

%values can start scaning in at any point

for i=1:4

if strncmp(a(i),"F",1)

F= a(i)

elseif strncmp(a(i),"B",1)

B= a(i)

elseif strncmp(a(i),"R",1)

R= a(i)

else strncmp(a(i),"L",1)

L= a(i)

end

end

%splits the letter from the value and creates a string array for the

%indivual direction

[f,m] = strsplit(F,"F")

[b,n] = strsplit(B,"B")

[r,o] = strsplit(R,"R")

[l,p] = strsplit(L,"L")

%turns array string at position 2 to a double

f2=str2double(f(2))

b2=str2double(b(2))

r2=str2double(r(2))

l2=str2double(l(2))

x = [0 0 f2 (b2.\*-1)] %creats the distance arrays

y = [r2 (l2k.\*-1) 0 0]

y1 = linspace(-200,200,400); %sets the 400x400 area that the sensor can read

x1 = linspace(-200,200,400);

clf %clears the graph for new values

hold on

bar(x,3) %plots the up and down values

barh(y,3) %plots the left and right distances

plot (x1,y1,'LineStyle','none')%creats the 400x400 area that the sensor can read

hold off

drawnow %updates plot

end

fclose(arduino);

# 