AROID 26.01

TABLE OF CONTENTS

INTRODUCTION

Project Summary

Aroid is a four-wheeled robot with self-driving abilities that can navigate green coloured roads autonomously while avoiding obstacles. Capabilities of Aroid include:

- Navigating green coloured paths autonomously.
- Avoiding obstacles in the path and not running into things.
- Accurately knowing how far it has travelled (odometry) and in what direction it has travelled.
- Sensing what direction is robot heading.
- Moving exact distances as it is asked to.
- Rotating exactly by an angle it is asked to.
- Building a map/history of the trajectory the robot has taken to get to the current position while navigating paths autonomously (as of now the robot has required data but I didn't get enough time to write code for visualizing this data).
- Ability to look in different directions and sense objects in front of it.
- Ability to be controlled remotely (wirelessly) from anywhere.
- Ability to sense light levels and temperature of the environment the robot is located in.
- Ability to send sensory readings to a remote when requested.

Note: This project is entirely open-source. Refer to this document/wiki for the circuit diagram and assembly process. Code for it is freely available here:
https://github.com/ArshSekhon/Aroid26.01

Here is the playlist showcasing capabilities for Aroid:
https://www.youtube.com/playlist?list=PLiJfy90wLClfj7bWsmUF1Zp-9bVlLG8do

P.S. When I started working on this project I had no experience in the field of Machine Vision and the only experience I had was from COMP 444. So if you want to build something similar or something as ambitious as this one don't hesitate, give it a shot. You'll learn a lot on the way and in the end, would have something you can be really proud of.

Concepts Used

Aroid uses the following major concepts taught in COMP 444:

Concept	Description
Behaviour-based Control	Aroid uses behaviour-based control. It has the following major behaviours: ● Staying in the middle of the green path and navigating the path. ● Avoiding obstacles, if an obstacle is detected then change direction and proceed. ● Moving until a specified distance has been covered (Behaviour triggered manually from remote). ● Rotating until Aroid rotates by an angle specified (Behaviour triggered manually from remote)
Behaviour Arbitration	Aroid uses a fixed behaviour hierarchy. For example, if a robot is trying to navigate the green path controller can ask it to proceed moving forward but if there is an obstacle in front of it, Aroid will rotate in order to avoid the obstacle. This is because the Obstacle Avoidance behaviour has a higher priority in the behaviour hierarchy. This example of behaviour arbitration can be seen in the obstacle avoidance testing video.
Closed-Loop Control	Aroid uses closed-loop control as it uses input from its sensors to determine the next action for example for navigating the path Aroid takes input from the camera, analyzes the video frame to find the next action it should take.
Feed-Forward Control	Aroid also makes use of open-loop control. When asked to move straight certain distance Aroid relies on it assumes that the movement of the robot is in a straight line and there is no sideways error/wheel slippage.
PID Control	Aroid uses PID control to stay in the middle of the green path. For more details refer to PID control under **“Programming Python Controller: Self Driving”** section.
Representation	Aroid when navigating paths creates an internal representation of its trajectory. Refer to “Keeping a travel history” in **Programming Python Controller: Self Driving** section for more details.
Complex and Simple Sensors Usage	Aroid uses a huge range of complex and simple sensors to sense its environment and its position in space. Refer to the **Hardware Assembly** section to see all the sensors Aroid uses.

Machine Vision	Aroid's self-driving abilities heavily rely on machine vision. I used machine vision concepts like colour range detection, edge detection, contour detection (used during the development phase but removed in the final version as it didn't do any better for my problem), Hough lines etc.
Emergent Behaviour	Emergent behaviour by definition is not something that was explicitly coded into the robot but I think Aroid does a great job at demonstrating emergent behaviour. For e.g. Aroid when not placed on a green path, it would try to find a green path and would travel random directions. During this process, it would try to avoid hitting obstacles and change direction by rotating if it encounters any obstacles then it continues to look for a green path by trying random directions.
Teleoperation	Aroid can be remotely controlled via HTTP requests. I wanted to create a simple dashboard application where you can control the robot by clicking buttons or other kinds of traditional controls but unfortunately, I did not have enough time for building it.

Motivation behind building Aroid

The Internet is filled with Arduino projects of all kinds but there was not a single Arduino based robot that is capable of autonomously navigating a path. There were some similar projects that used Raspberry Pi and Pixy Cam but they cost a lot to make. To give you an idea Pixy Cam alone costs CAD 80 and building such robots can cost several hundred dollars. Such a large investment would undoubtedly scare away some people who wish to build something similar.

I wanted to build a relatively cheaper robot that can accomplish the same goals. I wanted to give people the design of a relatively cheaper robot that they can use as a platform to play around with different machine vision and robotics control algorithms to achieve autonomous driving. Although I finished working on the project, there is still no limit to new features that can be added and enhancements possible.

Cutting the cost

In order to cut the cost significantly I had to find alternatives to both Raspberry Pi and Pixy Cam. I was able to find a cheaper alternative (although a bit low quality) to Pixy Cam: ESP32 Camera module. For Raspberry Pi, it wasn't as easy as just swapping it out for a different board. To cut costs I decided to use Arduino as the micro-controller but Arduino was undoubtedly underpowered for working with video streams.

So how did I solve this problem? I designed the robot to use 2 controlling modules:

- One lower-level controller that would be Arduino. The lower-level controller would directly interact with the hardware and will be able to execute instructions. The lower-level controller would handle tasks like spinning motors at a certain speed, rotating robot by given angle etc. It would also accept action requests from remote on the same WiFi network and perform the requested task.
- The second controlling module would be a higher-level controller which would do the computation-intensive work. This would be a python script running on a computer connected to the same WiFi network as Arduino. This controller would handle the task of analyzing the video stream from the ESP32, determine the best action and then send an action request to the Arduino which would execute them.

AROID IN ACTION: TESTING AND RESULTS

Note: I have used my main channel to upload these testing videos instead of the channel I used for my other COMP 444 videos.

Testing Odometry: Break Beam Sensors

Aroid successfully passed this test. The robot was able to accurately travel the distances as defined in the instruction sent to it. There was a slight error which I think is natural.

Here the video of me performing the testing: https://youtu.be/-CxYG-r3qTI

Testing Precise Rotation Abilities: MPU-9250

Aroid successfully passed this test. The robot was able to accurately rotate by a given angle accurately.

Here the video of me performing the testing: https://youtu.be/yHbv01Xk2-A

Note: Aroid is capable of rotating by any angle. I used 90 and 180 degrees Clockwise and Anticlockwise in the video to make it easier to judge how accurate the rotation was.

Testing Sensor Queries

Aroid allows reading from its sensors via HTTP requests. Anything ranging from mobile app, a desktop app to a web application can be used to make these requests but for this test, I used Postman to make these requests because it was just a bit more convenient. These queries worked seamlessly.

Here is a testing video for the same: https://youtu.be/6HbeKbQZ3Ag

Testing Control Queries

Aroid is controlled wirelessly via HTTP requests. Again, anything ranging from mobile app, a desktop app to a web application can be used to make these requests. Aroid performs this perfectly as well.

I did not create a separate video final test video for this one as controls via HTTP requests are the core of the robot's operation. The python controller for the autonomous drive part for the robot

makes use of HTTP requests to interact with the robot and it can be clearly seen working perfectly in the autonomous drive testing videos.

Testing Self Driving Abilities

This was the most challenging capability of the robot that required significant work. It did require some iterations to get this one right.

So my first goal was to make the robot travel on a straight or slightly curved road. I was able to accomplish this using Approach #1 as discussed in **Programming Python Controller: Self Driving** section and here is one of my testing videos for the same: https://youtu.be/HS_6owVqRBs

The problem with this approach was it relied on detecting lanes and was not able to handle sharp turns as it was picking up a lot of noise. So I decided to improve the robot to be able to handle sharp turns. Refer to Approach #2 in **Programming Python Controller: Self Driving** section for details on how I accomplished this. This approach worked flawlessly Aroid was able to handle sharp turns and navigate straight paths without a problem.

During all of my final tests, Aroid was always on the path (sometimes it was almost on the edge but never went off) and was able to navigate sharp turns perfectly. Here is a video that showcases several final test runs for Aroid:
https://youtu.be/G5iDnwRx6rU

Aroid definitely deserved a pass for this test.

Testing Obstacle Avoidance

Obstacle Avoidance behaviour was both easy to implement and test. So if Aroid detects an obstacle in front of it should stop and change direction and then continue moving. Here is a testing video for the same:
https://youtu.be/ZXR7tbBhWys

Aroid successfully passed this test.

Testing Trajectory Mapping

Aroid did pass this test as it was able to accurately store all the required data regarding the path it followed. Here is a video of me discussing how it works and in it, I also went over a test trajectory history that Aroid had saved from one of my self-driving tests: https://youtu.be/KV3M5oKyBm0

INITIAL DESIGN AND PLANNING

Initial Idea

As I mentioned in some of my discussion posts, I bought an ESP32 Camera module. Playing around with the module was super fun and interesting. So I decided that this was something I really wanted to use in my project.

So what I want to do for my final project is a 4 wheeled robot that uses machine vision (using ESP-32) to navigate. Yes you're thinking right a self-driving robot, I know it might be a bit too ambitious but I really really want to do it.

The abilities of the robot will include:

- Navigating paths autonomously.
- Avoiding obstacles in the path / not running into things.
- Accurately knowing how far it has travelled (odometry) and in what direction it has travelled.
- Sensing what direction is robot heading.
- Moving exact distances as it is asked to.
- Building a map/history of the trajectory the robot has taken to get to the current position while navigating paths autonomously.
- Ability to look in different directions and sense objects in front of it.
- Ability to be controlled remotely (wirelessly) from anywhere.
- Being able to get back home using its trajectory map without the assistance of Camera Stream.
- Being able to sense light levels and temperature of the environment the robot is located in.
- All the sensory readings of the robot should be accessible remotely.

The abilities listed above make my project satisfies the requirements for being classified as a robot: "A robot is an autonomous system which exists in the physical world, can sense its environment, and can act on it to achieve some goals." (Matarić, 2008)

Before proceeding any further I decided to give a name to my robot. So, I named it: **Aroid**.

Also, another important thing that I would like to mention here is that I have no prior experience in the field of machine vision and all my experience in robotics comes from COMP 444.

Initial Design: How it would work?

Aroid would consist of three pieces and this how it would work:

- ESP32 Camera would make the stream from the camera available over the WiFi network.
- Aroid would have the ability to be controlled remotely. It would use an Arduino Mega with onboard ESP8266. Onboard ESP8266 would allow any device on the same network to send commands to Arduino using HTTP requests and make it perform required actions. Aroid can be requested to perform actions like: positioning its camera in a certain way, spinning its motors at a given speed, rotating a given angle, moving a given distance in a given direction etc. Arduino would also take care of other things like reading from sensors. Detecting sudden obstacles, performing odometry etc.

  Note: Anything that is capable of making an HTTP request can be used to control Aroid and read from its sensors. It could be anything ranging from a web app, mobile app or any script.
- A script running my computer would act as a middle-man and it would bring the self-driving capabilities to Aroid. It would fetch the video stream from the ESP32 Camera module and process the video and would then ask Arduino to take the required action like turn left, go

straight etc. This logic would be implemented as a python script and would exploit the remote control (over WiFi) abilities offered by Arduino Mega.

- This script would also be able to get the sensory readings from the Aroid. So it would be in charge of things like: making Aroid navigate the path, building a trajectory map etc. It would also be able to take readings from Aroid's sensors and perform any action.

So as we can see from above there would be 2 controllers for Aroid: one lower-level controller that is used to control the actuators, read from sensors etc. and one upper-level controller that would be sending more abstract instructions to the lower level controller and let it handle the details. Both of them would work together to accomplish the task of navigating the path, creating a trajectory etc.

HARDWARE ASSEMBLY

Hardware List

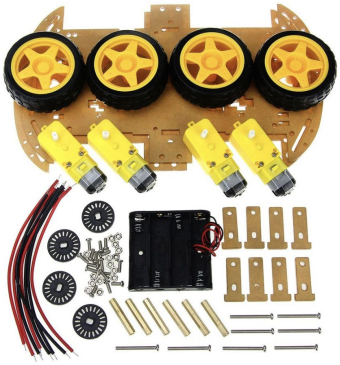
Following things went into the making of Aroid:

(Refer to **Appendix: Hardware Used** to get more information on each of them and the links to buy)

Hardware	
Component	**Price**
Chassis	USD 14.49
Camera: ESP32-S Ai Thinker Module	USD 7.45
Arduino Mega2560 + ESP8266	USD 11.94
Breadboard + MB102 White Power Module + Wires	USD 3.45
SSD1306 OLED Display Module 128X64 I2C	USD 1.76
Dupont Jumper wire 10CM Male to Male + Female to Male + Female to Female	USD 1.78
Elegoo 120pcs Multicolored Dupont Wires	USD 9.00
Willwin 5pcs LM393 Chip IR Optocoupler	USD 9.39
2pc SparkFun Motor Driver - Dual TB6612FNG (with Headers)	USD 10.90 (One was part of SIK)
2pcs Lofty Ambition MG90S 9g Metal Gear Servos	USD 5.88
Ultrasonic Distance Sensor - HC-SR04	USD 3.95 (Was part of SIK)
Resistor 10K Ohm 1/4 Watt PTH - 20 pack	USD 1.20 (Was part of SIK)
Mini Photocell	USD 1.50(Was part of SIK)
Trimpot 10K Ohm with Knob	USD 0.95(Was part of SIK)
Mini Speaker - PC Mount 12mm 2.048kHz	USD 1.95(Was part of SIK)
Total Cost (Excluding SIK Parts)	USD 70.59
Total Cost (All Parts)	USD 85.59

Assembling Chassis

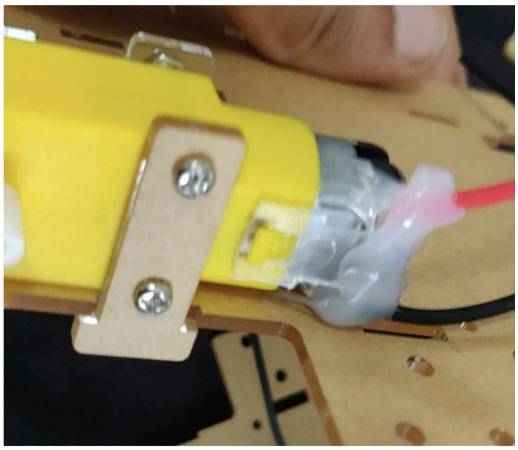
The starting point of building the robot was assembling the chassis. I used a 4WD Robot Chassis that I bought from AliExpress. The chassis was sold as a bundle that included wheels, 4 DC motors, speed encoders for odometry and some wires.



Buy Now:
https://www.aliexpress.com/item/32807838716.html?spm=a2g0s.9042311.0.0.27424c4d6uYMge

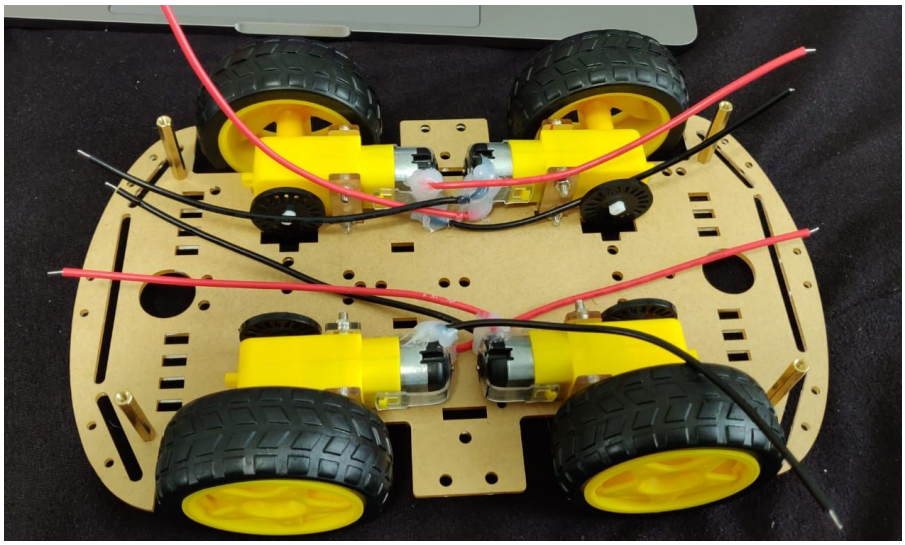
Assembly Problem #1: The first problem that I encountered when creating this robot was that as shown in the image above motors included in the bundle did not have wires attached to them. I was supposed to solder the wires to the motors but I neither had soldering iron nor had any soldering experience.

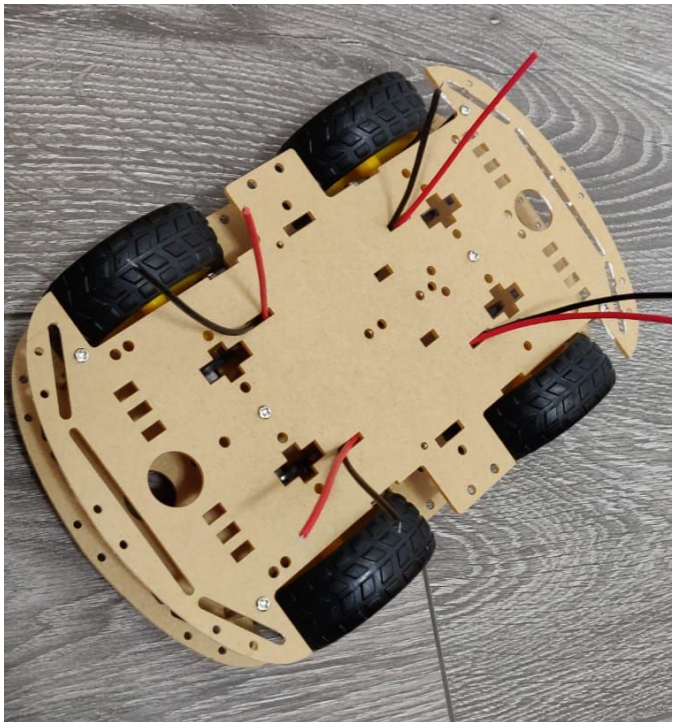
Solution: Hot Glue! (Solution to all my problems :P). I twisted the ends of wires on to the terminals of the motors to ensure that the connection is reliable and then covered the entire thing with hot glue. Not an ideal way but it proved to be pretty reliable :P



Attaching Motors to the Chassis: https://youtu.be/QqaeyPw-wNU

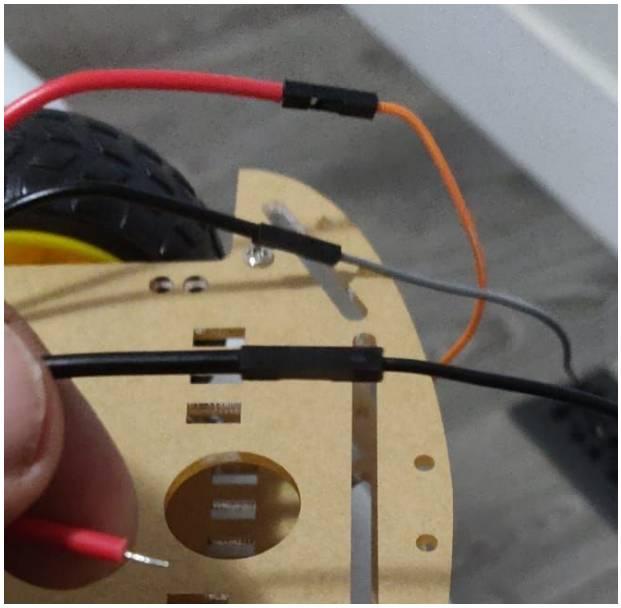
Final Result:





Assembly Problem #2:
I wanted to use a breadboard to connect motors to the motor driver but the wires connected to the motor terminal did not end in pins :(These wires ended in ends that were pointed but a bit too flexible to be used with the breadboard.

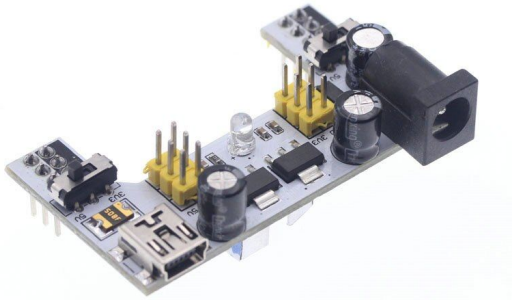
Solution:
I used a Male to Female pin connector wire! I carefully push the end of each wire into the female connector side of a male to female connector wire that I had bought from AliExpress. The connection was firm but not durable enough. So I used some black electric tape to secure the connections and it worked perfectly!



Adding Arduino, Breadboard, Motor Drivers and a Power source

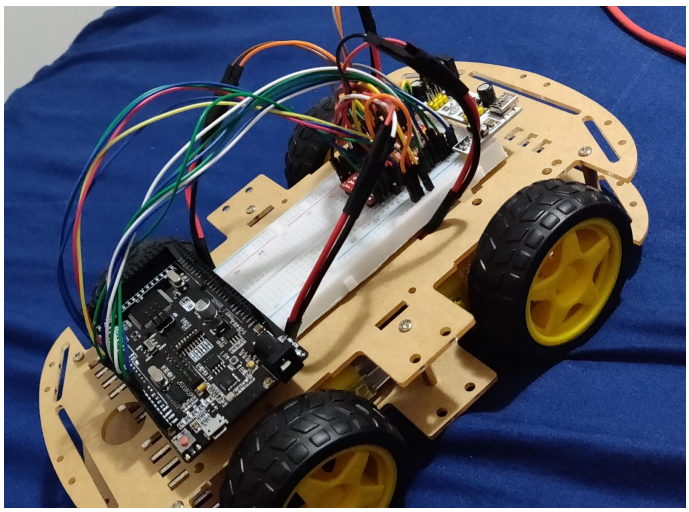
Since I want my robot to connect to the Internet, I am using an Arduino Mega that has an ESP8266 on board so I can connect to the WiFi. Also since I am planning to add a lot of sensors and other components to my project, I decided on using a large breadboard.

4 motors and all other components that I am planning to add will consume a lot of power, therefore, I also decided to use an MB-102 power supply. The good thing about this power supply was that I can use the power banks that I have to power the robot.



Power Supply

Then I installed the two motor drivers from Sparkfun(the exact same ones which we got as part of the SIK). I hooked up the connections to the motors and the Arduino. I used MB-102 power supply to provide the power to the motors and motor drivers. This is what my robot looked like at this stage:

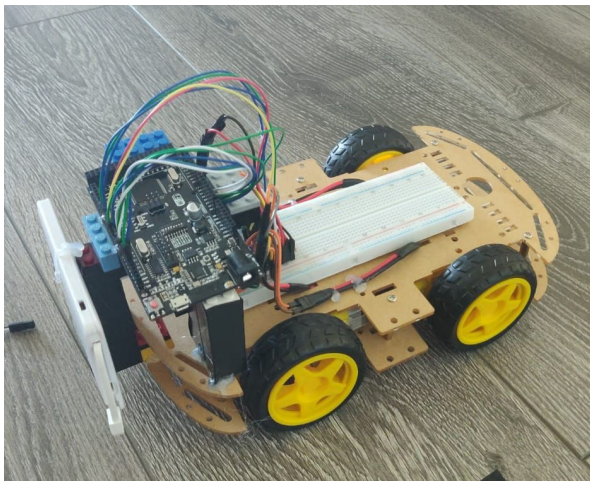


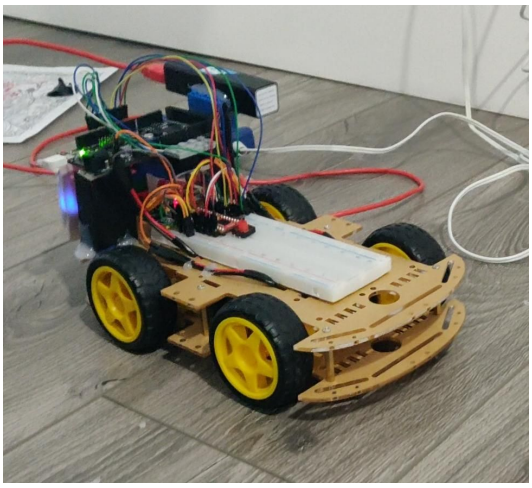
Assembly Problem #3:
I did not have enough space on the robot. As visible from the image above the breadboard takes up the majority of the space on the chassis leaving no room for the Arduino.

Solution:
Lego and Hot glue! Yes, hot glue comes to the rescue again :P

I used lego and hot glue to create a raised platform for Arduino so that I can slide the breadboard under the lego platform. I used hot glue for extra strength in holding lego together.

This is what Aroid looked like after this:





I also added 2 power banks to the design to ensure that the robot can operate wirelessly. The white power bank in the left image above is used to power Arduino Mega whereas the black power bank in the right image is used to power other components via MB-102 power supply on the breadboard.

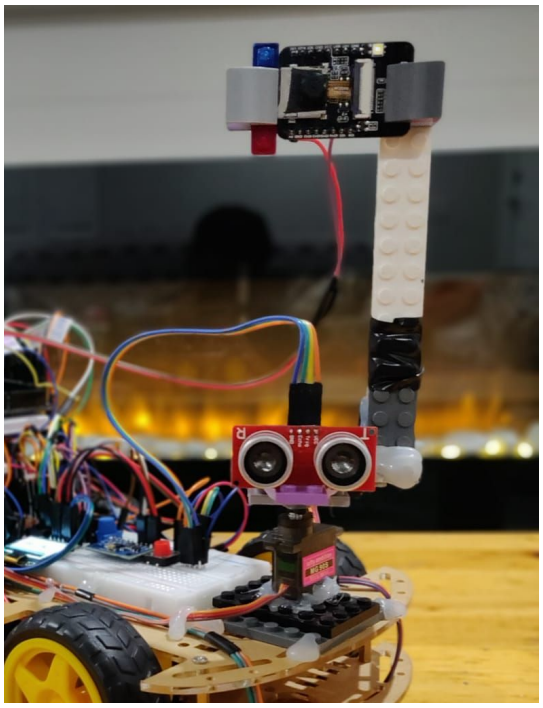
Adding Camera and Ultrasonic Sensor

The next step was to install the camera and ultrasonic sensors. I wanted to make these both sensors movable so that they can be pointed in any direction that I wanted them to be in.

An obvious solution to this problem was to create a head with 2 degrees of freedom powered by servos. What did I create it out of? Lego, Black electric tape and hot glue :p

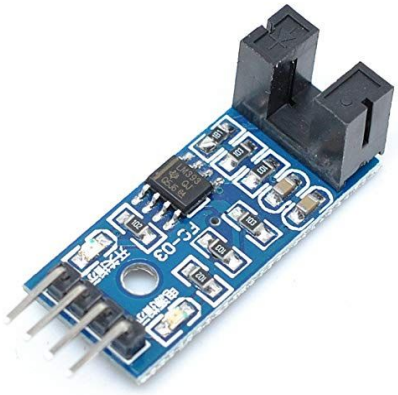
This is what I created:
https://youtu.be/LoQII3-DI1w

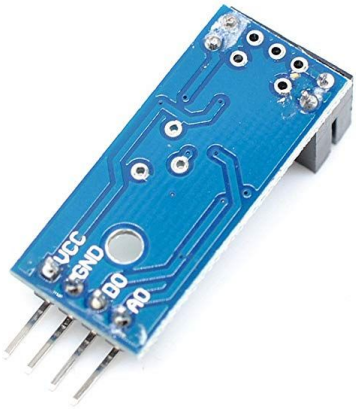
For making some improvements in the machine vision part, I had to change the camera mount a bit to ensure that the camera is positioned in the middle of the robot, not towards one side. This was fairly straightforward, all I had to do was add a couple more Lego blocks.



New Camera and Ultrasonic Sensor mount

Adding LM-393 Break-beam Opto Sensors

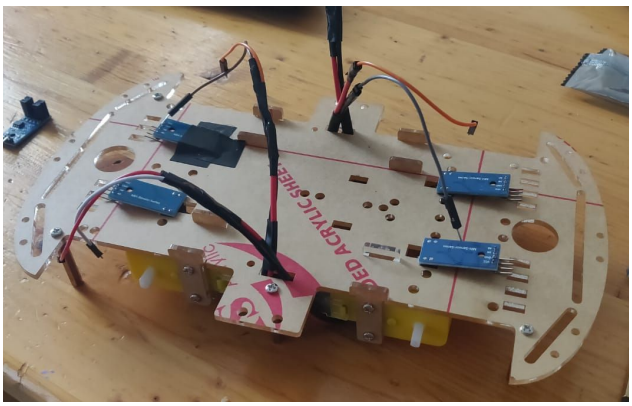


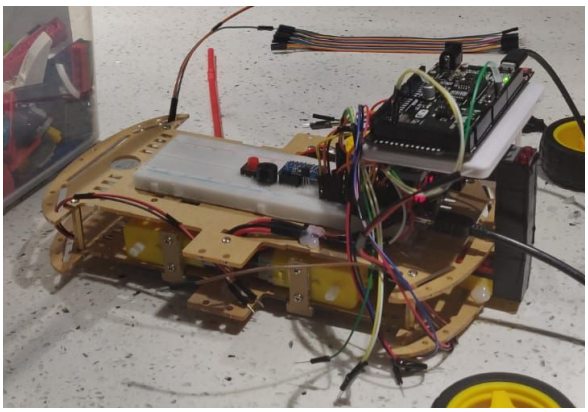


Installing these sensors was the most challenging part. By the time, I found these sensors on amazon and ordered them I had already assembled the entire chassis. The good thing about the chassis was that it had slots in it for these sensors.

Assembly Problem #4: In order to install these sensors I had to take apart the entire chassis and reassemble everything.

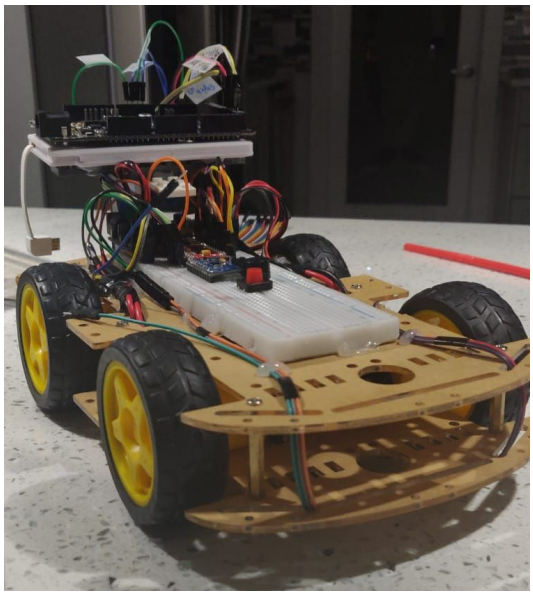
Solution: There is nothing much that could have been done at this point except taking the robot apart :(So…. I did it and it took a lots and lots of time.

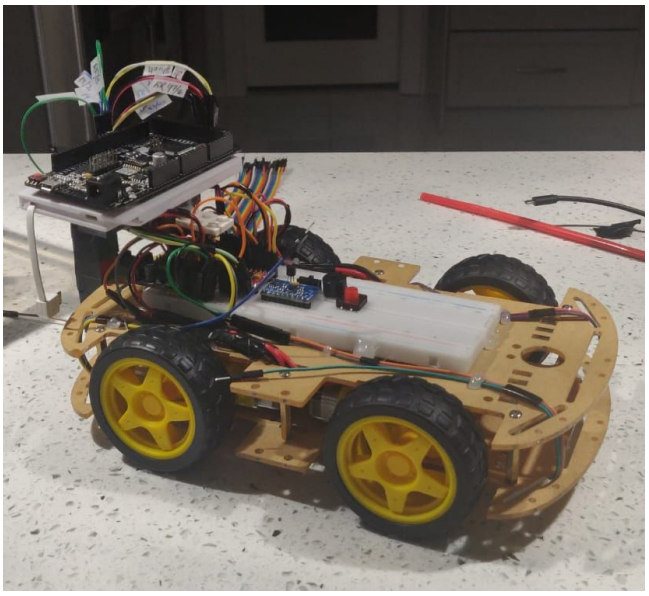




It gave me a chance to redesign the entire robot, so I changed a few things. Some of the things that I changed are: the position of the battery banks, reduced the size of the platform that Arduino is placed on etc. I also ensured to label all the wires using sticky tape to make them easier to manage.

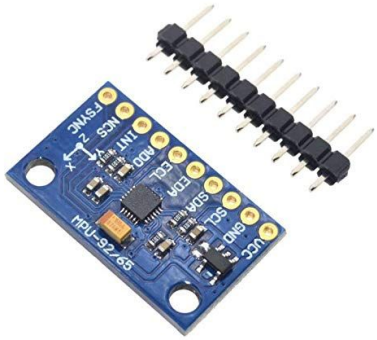
This is the redesigned Aroid:





IMPORTANT NOTE: In order to make these sensors work I had to connect the D0 pin of these sensors to the interrupt pins of the Arduino. For Arduino Mega D2, D3, D18, D19, D20 and D21 pins can be used as interrupt pins. To learn more about interrupts see the **Appendix: Arduino Interrupts.**

Adding MPU-9250



MPU-9250 is accelerometer, gyroscope and magnetometer all in the same module.

Assembly Problem #5: The module shipped with no header attached and needed to be soldered. **Solution:** I tried using the same hot glue approach but it was not reliable enough. So the only option was to solder the header on to the module. I borrowed a soldering iron from a friend of mine and learned how to solder. 15-20 mins worth of practice later I was able to solder the header to the module perfectly!

I am using this to sense the orientation of Aroid. This module uses the I2C protocol for communication (Read **Appendix: I2C Protocol** for more details on Arduino). Due to I2C hooking it up was fairly straight forward all I had to do is connect:

VCC pin ←→ 5V
GND pin ←→ GND
SCL pin ←→ SCL Arduino
SDA pin ←→ SDA Arduino

Adding Buzzer

Adding buzzer was fairly straightforward it required the same setup as suggested by the SIK Guide's Circuit 2A. It was the easiest to install the component.

Adding Photoresistor and Temperature Sensor

Adding photoresistor and temperature sensor was fairly straightforward as I had previously done it for several circuits in Sparkfun's Inventor's Kit.

Assembly Problem #6: I was initially powering these sensors using an external power source (although power supply was running in 5V mode the same as I would have had with Arduino).

When I performed the analogRead to take a reading from the sensor I was receiving values that jumped up and down erratically. I thought there was something wrong with my circuit but after a few google searches, I finally realized what was wrong.

Solution: The power supply adapter from the power bank was outputting voltage of ~5-5.1V (it fluctuated, even more, when the motors were running) whereas the Arduino was output somewhere around ~4.8-4.95V. This is what was causing the error in Analog Read as in order to perform analogRead on a pin Arduino measures the voltage on that pin which should be in the range of 0-3.3/5V (less than the voltage difference between the power and ground pin on the board). https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

So I decided to ditch the idea of powering these sensors from the external power source and used the Arduino 5V pin to provide the power to these sensors.

Adding SSD1306 OLED Screen

I was initially planning to use the LCD screen that shipped with the SIK to display information about the network such as IP of the robot, subnet etc. but stupid me accidentally connected two wires of the LCD incorrectly and…. the smell… and then…. the fumes… By this point, I had realized that I had successfully killed my LCD.

My intuition was proven to be correct when I tried making the LCD work again and half of the LCD would just not display anything.

So I ordered an OLED from AliExpress. I was really excited about the new OLED as I can do more stuff with it like displaying logos, drawing graphics etc. and the biggest relief was the fact that this screen used I2C. So a lot fewer wires to handle.

Due to I2C hooking it up was fairly straight forward as well all I had to do is connect:

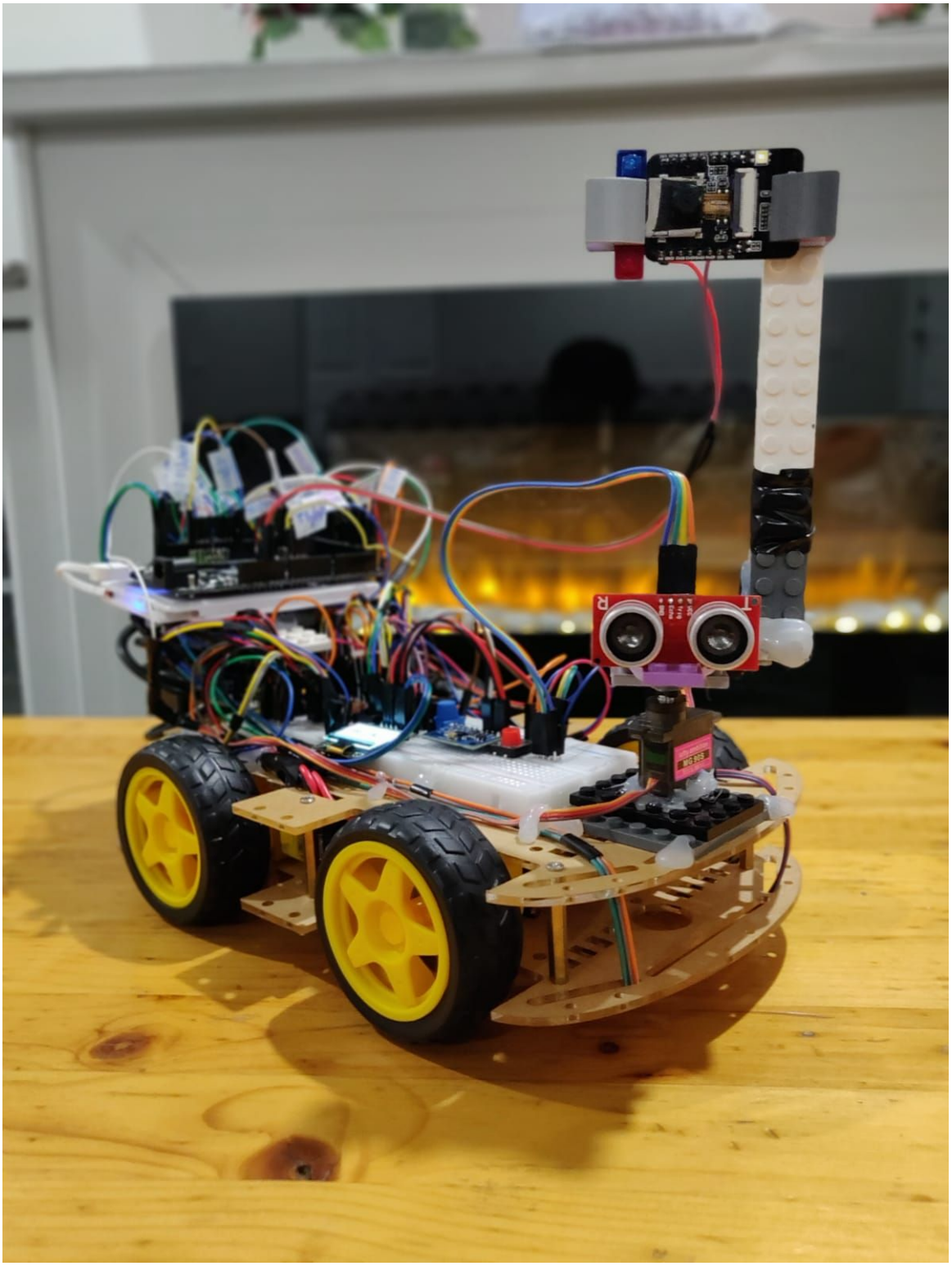
VCC pin ←→ 5V
GND pin ←→ GND
SCL pin ←→ SCL Arduino
SDA pin ←→ SDA Arduino

(Read **Appendix: I2C Protocol** for more details on Arduino)

Wiring and Connections

I tried my best to have good cable management for Aroid. I did label each wire using sticky tape that saved me a lot of time and helped me debug and checking for bad connections. I have also created a circuit diagram for Aroid. Please refer to **Appendix: Circuit Diagram** for details on how each component is connected to the Arduino.

MEET AROID 26.01: AFTER ASSEMBLY



MAKING IT ALL WORK WITH CODE

Programming Arduino

Making the Motors move

Making the motors move was the easiest part of this project. Since I was using 2 of Sparkfun's motor Drivers exactly the same to that shipped with the SIK, making the motors was fairly straight forward. The only thing that changed is that I was powering the motors using an external power supply.

All the code for motor control is located here:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/MotorControl.ino

Receiving commands over WiFi

The ability for Aroid to connect to WiFi and receive commands wirelessly was an important piece. In order to do this, I used an Arduino Mega with onboard ESP8266. How I got it to work is well documented in this landing post:
https://landing.athabascau.ca/discussion/view/4599242/arduino-mega-with-wifi-connection-capabilities
Post also contains links to code snippets that can help you to better understand how this works.

For Aroid this is the code that I used for the ESP8266 module to handle the requests:
https://github.com/ArshSekhon/Aroid26.01/blob/master/ESPServerIO/ESPServerIO.ino
This sketch allows both sending and receiving data to and from Arduino. How this works is ESP8266 accepts the commands over the WiFi. If it is a properly structured request then the ESP module writes the command to Arduino's Serial3.

Arduino reads the request from the Serial3 and performs the required action.

For Arduino, the logic for handling the incoming requests from ESP8266 is located in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/IO.ino

Sensing Light Levels and Temperature

Sensing light levels using a photoresistor is something that I had already done for circuits in the SIK Guide so it should be easy to do the same for this project right? Unfortunately, it was not as easy.

Problem: The readings from the photoresistor and temperature sensor were fluctuating a lot, way beyond the negligible point.

Solution: I tried almost everything and nothing worked. I tried to eliminate each variable one at a time. I used Arduino Mega for this project so I tried switching to Arduino Uno and testing these sensors there. The probability of having bad sensors was fairly low as 2 known good sensors cannot go bad at the same time. Then I realized that the last variable the power source was the cause of the problem. I was powering these sensors using another power source than my Arduino (although power supply was running in 5V mode the same as I would have had with Arduino).

When I measured using a multimeter, the power supply adapter from the power bank was outputting voltage of ~5-5.1V (it fluctuated even more when the motors were running) whereas the Arduino was output somewhere around ~4.8-4.95V. This is what was causing the error in Analog Read as in order to perform analogRead on a pin Arduino measures the voltage on that pin which should be in the ranging of 0-3.3/5V (less than the voltage difference between the power and ground pin on the board).
https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

I then decided to provide power to these sensors using Arduino and the same code worked flawlessly.

The code for getting the readings from these sensors is located in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/SensorsAndDevices.ino

Displaying Content on SSD 1306 OLED

Using SSD 1306 was my first time working with a component that uses the I2C protocol. I was able to find this video tutorial on youtube that explained how to use it: https://youtu.be/A9EwJ7M7OsI

I followed the video step by step but unfortunately, the display did not work. I kept scratching my head and then after some research, I realized that I was using the wrong I2C address for the screen. Using the I2C Scanner Sketch I was able to determine the actual I2C address of the screen and it worked! The example sketch that came with Adafruit's library for SSD1306 was really helpful to learn how to use the OLED for different tasks like displaying logos, drawing figures etc.

In Aroid there are two files that are related to displaying content on OLED:
This file contains the logic for displaying content on the OLED screen:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/OLED.ino

This header file contains the bitmap for Aroid's logo that is displayed on the screen:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/Aroid_Logo.h

Using MPU-9250 to sense orientation in space

Using the MPU-9250 sensor was one of the most challenging parts for Aroid. I wanted to use this module to sense the direction of the heading of Aroid. I used the magnetometer in the module to measure the earth's magnetic field and used the value of the magnetic field in both x and y-axis to estimate the angle of the heading of Aroid.

Problem: The biggest problem that I faced using this module was that I was unable to calibrate the module properly. I tried using almost every single library available online for the MPU-9250 module but nothing worked.

Solution: After spending an entire day on getting this module to work and not making any progress at all. I decided to do the calibration process by hand. How did I do it? I installed a magnetometer app on my mobile phone and used the values it gave me for both x and y-axis to calculate the values for scaling and bias factor for values for both axes. Then I manually applied this scaling and adjusted the bias in code. It worked perfectly! But unfortunately, this is not the ideal way of doing this.

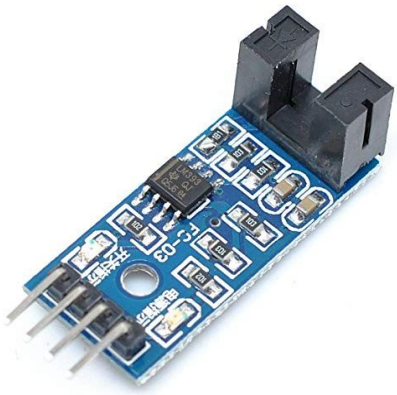
I have not calibrated the accelerometer and gyroscope as I did not really use them in my project. Also, the scaling and bias factors for the magnetometer are hardcoded into the code as of now. The code for this sensor is present in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/MPU92_65.ino

Odometer with Break Beam Sensor (LM393)

The break beam sensor uses external interrupts to sense the wheel rotations. This is how it works:

- The motors are fitted with disc speed encoders that came with the robot chassis.

- These discs then go inside the slot of break beam sensors.



- When disc rotates the slots in the disc allow the infrared light to pass through. So anytime the infrared light passes through it is evident that one slot of the disc has passed through the infrared beam.

How this works in code is when the infrared light passes through it causes an interrupt. This interrupts increments the counter that holds the number of slots that have passed through. This happens for each of the four wheels.

There is also a scheduled timer that uses the counter value for each number and calculates the RPM for the wheel. It also calculates the distance moved by multiplying the number of rotation of each wheel by the circumference of the wheel.

The code for this is present in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/Odometry.ino

Note: Since opto sensors use the external interrupts so D0 pin of each of these sensors has to be connected to the interrupt pins of Arduino (D2, D3, D18, D19).

Using Ultrasonic Sensor

Again, measuring distance using an ultrasonic sensor is something that we had done in circuits listed in the Sparkfun Inventor's Kit but implementing this was not as easy as it was for SIK Guide's Circuit.

Problem:
When I tried using the ultrasonic sensors and it was returning me super low values like 0.29 every once in a while and also the values of readings fluctuate a lot. This made no sense to me. I was really confused.

Solution:
Then after some time debugging, I started thinking about how the ultrasonic sensor worked and I found the cause of the problem!

It was the LM393 Opto Break Beam sensors that I was using for odometry. In order for a break beam sensor to perform odometry, it is necessary to make use of interrupts but this caused problems when I tried to take readings from the ultrasonic sensor because the interrupts caused by odometry made the echo time from the ultrasonic sensor to be inaccurate. Why? Think of how ultrasonic sensor works it sends out a sound wave and then measures the time taken by the echo to reach back to the sensor and uses it to calculate the distance of an object in front of it. So far so good. But what if while the sonar sensor is waiting for the response an interrupt happens and changes the flow of execution. The ultrasonic sensor can fail to sense the echo. This would cause the sensor to fail.

So my first thought was what if I disable interrupts while the ultrasonic sensor is taking a reading and then enable them again after it is done. It sounded pretty reasonable to me but it did not work as expected it caused servos to move randomly and trust me it looked a lot more horrible than what it sounds like. I suspect that the Servo library might have something to do with interrupts.

Then my next thought was what if we treat venom with venom, I mean what if we fix problems caused by interrupts using interrupts so I started doing some research and after some time googling I was able to find this Instructable post.
https://www.instructables.com/id/Non-blocking-Ultrasonic-Sensor-for-Arduino/

It discusses how to use ultrasonic sensors using interrupts. But now I had another problem I did not have enough external interrupt pins :(. This was because I2C and LM393 sensors for odometry were using all of my interrupt pins. After a lot of research and trying out different things I found that there are different kinds of interrupts called PinChangeInterrupts. I was also able to find a library called "EnableInterrupts" that allowed me to easily use Pin Change Interrupts. I used this to make my ultrasonic sensor work using interrupts and everything worked perfectly.

So how it worked was instead of using **pulseIn()** function to busy-wait until the echo is detected it used interrupts to detect when echo is received. So the time measurement in which echo is received is accurate and therefore the distance reading is accurate as well.

Code for taking readings from Ultrasonic sensor using interrupts is present in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/UltrasonicSensor.ino

Also, an important thing to note here is this line of code should be placed inside the setup function in order to attach the interrupt and make Ultrasonic sensor work.

```
enableInterrupt(SONAR_ECHO_PIN, ultrasonicSensorInterruptHandler, CHANGE);
```

Responding to requests for sensor readings over WiFi

Note: Before reading this section please read the section “Receiving commands over WiFi” first.

Making Arduino respond to requests was fairly straight forward. Just like for receiving commands I created another HTTP API endpoint on ESP8266 for read requests. How this thing worked is whenever a request for sensory reading is received on this endpoint the ESP module forwards this request to Arduino by writing it to Serial3 of Arduino. After then it waits until it receives a response from Arduino via Serial3. If it receives a response within 5 seconds of initial request then the ESP module forwards the same to the client that made the request and if it does not receive a response from Arduino within 5 seconds then it sends an error code of 403 to the client.

Refer to the code for ESP8266 for details :
https://github.com/ArshSekhon/Aroid26.01/blob/master/ESPServerIO/ESPServerIO.ino

The code for how Arduino handles requests for sensory readings is present in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/Bot/IO.ino

Streaming from ESP-32 over WiFi

Getting the ESP32 Camera to work was a challenge on its own, this was due to the fact that I had absolutely no idea about how things thing worked when I purchased it. My story of getting it to work is well documented in this post on the landing:
https://landing.athabascau.ca/discussion/view/4513474/connect-your-robot-to-a-wireless-network-to-control-it-and-stream-video-from-it-esp-32-camera-module

This tutorial explains really well how to connect ESP32 to the WiFi network and start streaming from it: https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/

I am using a modified version of the ESP32 Camera Web server started example for this project as the only thing this module does is stream video over WiFi and example code did this flawlessly. So why reinvent the wheel. I did modify the ESP32 Starter code to have it acquire a fixed IP of 192.168.0.100 that made using it much more convenient.

The code that I used for the ESP 32 Camera module is present here:
https://github.com/ArshSekhon/Aroid26.01/tree/master/ESP32CameraServer

Programming Python Controller: Self Driving

Implementing a controller for the Self Driving part was a challenge on its own but I was able to overcome it successfully. Due to a lack of background in the field of machine vision, it took me a while to figure this part out.

So what I ended up implementing is a closed-loop controller that uses PID control. Python controller is in charge of making the robot stay and travel on the path. It is also responsible for creating a trajectory history as the robot moves which is the same as creating a rough map of the path it has travelled. This is a general description of how the python controller is implemented:

- Python controller requests a frame of the video stream from the ESP32 Camera module.
- Then analyzes the video frame to calculate an estimate of the direction in which it should head to.
  (**Note:** This heading is represented as a pixel on the frame. So if the path ahead is straight then this heading should correspond to the center of the frame. If there is a turn to the left the heading would shift to the left of the frame. Watch the video to get a better idea).
- Then using this heading it calculates the error and then uses PID control to calculate how much the robot should rotate to get the heading to be roughly in the center of the screen. If the heading is roughly in the center of the screen then the robot just starts moving straight.

I tried using the following approaches for analyzing the video frame to calculate the desired heading:

Finding the right direction to head to: Approach #1

In my first approach, I tried using a lane detection process to find the lines and programmed the robot to stay within those lines. This is step by step process of how I did it:

- I converted the RGB frame to grayscale.
- Performed a canny edge detection on the grayscale frame.
- Applied a Gaussian blur to the frame obtained after canny edge detection.
- Applied a mask to hide all the edges that are not in the region of interest.
- Calculated the hough lines for the region of interest.
- Then I calculated the average for the slopes and intercepts for the hough lines in both the left and right side of the frame. This gave me 2 lines(blue lines in the demo) that were an estimate of the lane boundaries in front of the Aroid.
- Then I used these two lines to calculate the desired direction of heading for Aroid.

This approach worked fine for the straight or slightly curved path but did not do very well on sharp turns. Also sometimes the camera was picking up lines from the floor or other objects in the frame. This approach would have worked perfectly when navigating empty corridors but it did not do very well when handling paths with sharp curves in rooms packed with all kinds of things like couches,

wall hangings, floor with patterns on it etc. This was because during a sharp curve region of interest processed by the controller also picked up lines from other objects.

The source code for this approach can be found in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/controller_approach1.py

Finding the right direction to head to: Approach #2

So I definitely need something better than Approach #1 as I wanted my robot to handle sharp curves. So I decided to make use of a bright green coloured path. So that Aroid can also use colour information, combine it with the lane detection from Approach #1 and determine the correct direction of heading.

New things that I added in this approach were:

- **I added camera calibration:** While I was trying to learn more machine vision approaches, I also found out that cameras also cause the images to distort and it is always a good idea to get rid of these distortions before we try to extract any information from these images. This web post does a great job explaining how camera calibration works and why it is necessary: https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html
- **Utilized Color Information:** I also decided to use a mask that only selects the portion of the image that has pixels with hue value in the region corresponding to the green colour. Using this I was able to detect all the greenish pixels. Then I plotted the number of green pixels against the x-coordinate. This allowed me to detect what region of the screen has the most number of greenish pixels. This information really helped Arduino detect sharp turns. This in combination with lane detection did a perfect job making the robot determine its angle of heading.

The source code for this approach can be found in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/controller.py

Approach 2 worked perfectly. It helped Aroid determine the correct heading even when navigating sharp turns.

P.S. These are the two major approaches that I tried. There were a ton of iterations of each approach that involved things like tweaking parameters to trying different filters etc.

PID Control

Now after processing the image since we had the desired heading direction, I was able to calculate the error in heading. Then I used a PID control mechanism to determine how much Aroid should turn.

Implementing the PID control was relatively easier but finding the right value for Kp, Kd and Ki parameters required a lot of patience and time. After several iterations, the new parameters started doing quite an impressive job as you guys can tell from the test videos. I have implemented a class that encapsulates the logic for PID control:
https://github.com/ArshSekhon/Aroid26.01/blob/master/PID.py

This is then used by the main controller script.

Keeping a travel history

Since we had all the sensors in-place and working perfectly with Arduino. Implementing this was not hard. All I did in code was whenever the robot moved I stored its direction of heading (represented as an angle from the north) and after it was finished moving I recorded the distance it has moved. I did it for every single move. This information was obtained from Arduino via HTTP requests.

Since the Aroid only rotates or travels in straight lines, this information is enough to re-create the trajectory that Aroid has followed. After controller script is exited this data is saved to a file called travel_history.txt

The logic for recording the trajectory is present in:
https://github.com/ArshSekhon/Aroid26.01/blob/master/travel_mapping.py

This is then used by the controller script to perform and save the readings as the robot moves.

ANSWERING THE GUIDING QUESTIONS

Design Decisions

How did you approach the overall design of the project?
I have discussed the overall approach I have taken for this project in detail in **Motivation behind building Aroid** and the **Initial Design and Planning** Section of this document.

I would also recommend reading the **Concepts Used** section for a better understanding of a higher level design of Aroid.

What resources did you employ at this stage to assist you in creating an overall design?
MIT Robotics Primer was a great resource when it came to understanding higher-level design considerations. The content covered in the textbook did help me a lot to understand how to approach this project.

This phase also required some creativity as I was searching through hardware on AliExpress and was thinking about how I can use them to create a robot that can get the job done while not being heavy on my pocket.

General Programming Considerations

Were any special programming considerations required to implement your design?
Since Aroid was clearly underpowered to do the video processing, I had to create a distributed controller architecture with a lower level controller (Arduino) and a higher-level controller (a python script that would run on a computer on the same network as Arduino). As mentioned previously in this document:

- A lower-level controller would directly interact with the hardware. It would handle tasks like spinning motors at a certain speed, rotating robot by given angle etc. It would also accept action requests from remote on the same WiFi network and perform the requested task.
- The second controlling module would be a higher-level controller which would handle the task of analyzing the video stream from the ESP32, determine the best action and then send an action request to the Arduino which would execute them.

Also, when programming Arduino I had to be careful regarding the timing. This is because I had to use interrupts for break beam sensors. These interrupts could happen any time so I had to be careful when dealing with components that are time-sensitive. For example, interrupts really caused an issue with the ultrasonic sensor as timing is critical when reading distance from the ultrasonic sensor.

Where did you seek resource information to assist in this process?
I think most helpful was my previous experience working with HTTP APIs and the knowledge regarding timing and related topics in computers that I had gained from an Operating systems course.

Program implementation

Discuss your robot's programming.
I have discussed the programming for Aroid in-depth in **Making it all Work with Code** section of this document.

What major categories of robotic control did you implement?
Aroid uses behaviour-based control overall. It uses PID control for steering and positioning it properly on the path. For more details please refer to **Concepts Used** and **Making it all Work with Code** section of this document.

I also wanted to implement deliberative control to make Aroid do things like solving a maze. Aroid has all the hardware and software required to be the starting point for this: Aroid can create a trajectory map, detect obstacles and stay on the path. All of this can be employed to create a maze solving robot but this requires a huge investment of time so I decided to keep it out of the scope for this project.

Testing

How did you test your robot and program to ensure you had met your design goals?
Please refer to the next section **"Aroid in Action: Testing and Results"** for more details on this question.

Discuss any deviations from your expected test plan output and what you did about this.
As discussed in the testing videos in the next section, there were slight errors when it came to making Aroid move by a given distance and also when rotating it by a certain angle but I think these were natural and small enough to be ignored. Aroid did a great job for self-driving test and other tests.

Outcome

Discuss the outcome of your project. Did your robot meet all design expectations?
Aroid received a pass for all of its tests! Refer to the next section **"Aroid in Action: Testing and Results"** for more details.

Now that you are done, what would you do differently were you to undertake the same project again?

First and foremost, I would like to have more time to invest into this project. Also, I think It would be amazing to gain some experience in the field of Machine Vision first as it would really open a lot of new possibilities. Experience in the field of Deep Learning would also be a huge asset as well. It would allow me to do more fun things like detecting stop signs, toy traffic lights etc. and it would be super cool to have Aroid do things like that.

Also, another thing that can make Aroid's trajectory much smoother and better is if it uses a chassis that allows steering the front wheels. I think this would be the only thing that I would like to change in the robot itself.

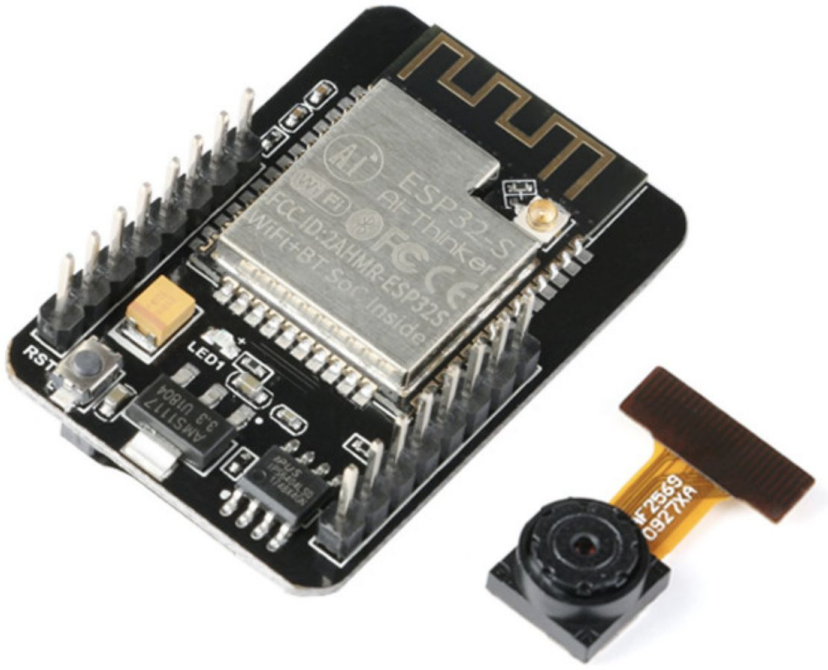
APPENDIX: HARDWARE USED

Robot Chassis

Description:
Chassis Kit consisted of the following elements: 2xCar chassis plates, 4xDC gear motor, 4xYellow wheels, 1x4-AA-L Battery box, 4x20 line tachometer encoder, 8xFastener, Several pillars screw nut.

Price: USD 14.49
Buy Now:
https://www.aliexpress.com/item/32807838716.html?spm=a2g0s.9042311.0.0.27424c4d6uYMge

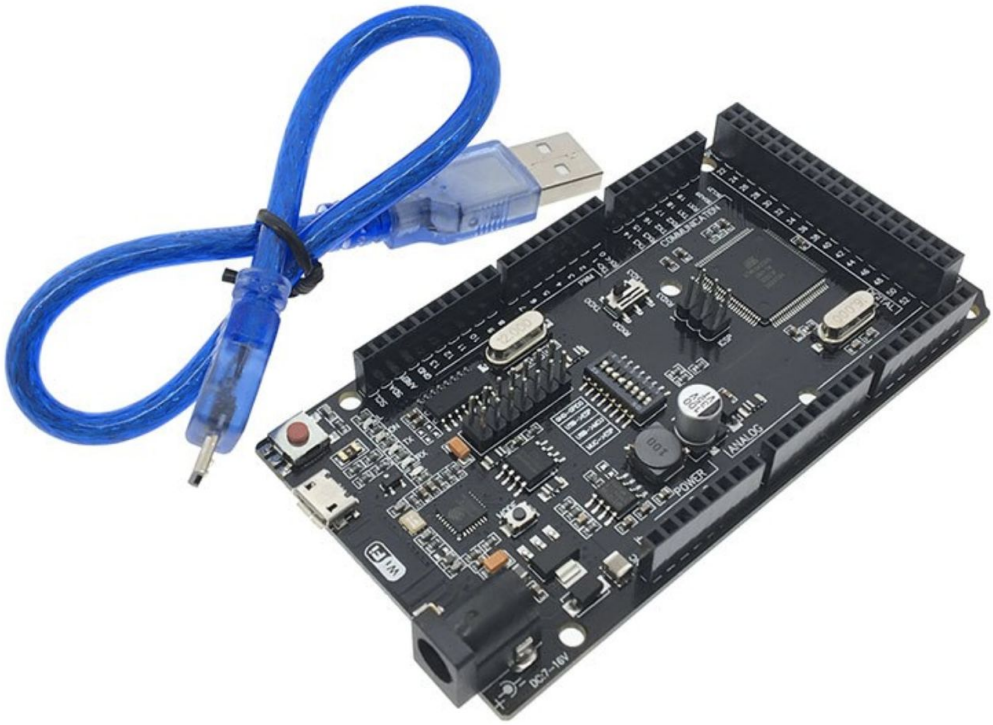
Camera: ESP32-S Ai Thinker Module



Description: This the camera module used in the head of the robot. It works independently of the Arduino and is programmed separately using an FTDI interface. It is capable of connecting to the WiFi and the video from the camera can be streamed over WiFi. In this project, it is programmed to be used as an IP camera to allows the robot to see.

Price: USD 7.45
Buy Now:
https://www.aliexpress.com/item/32947577882.html?spm=a2g0s.9042311.0.0.27424c4dml9pld

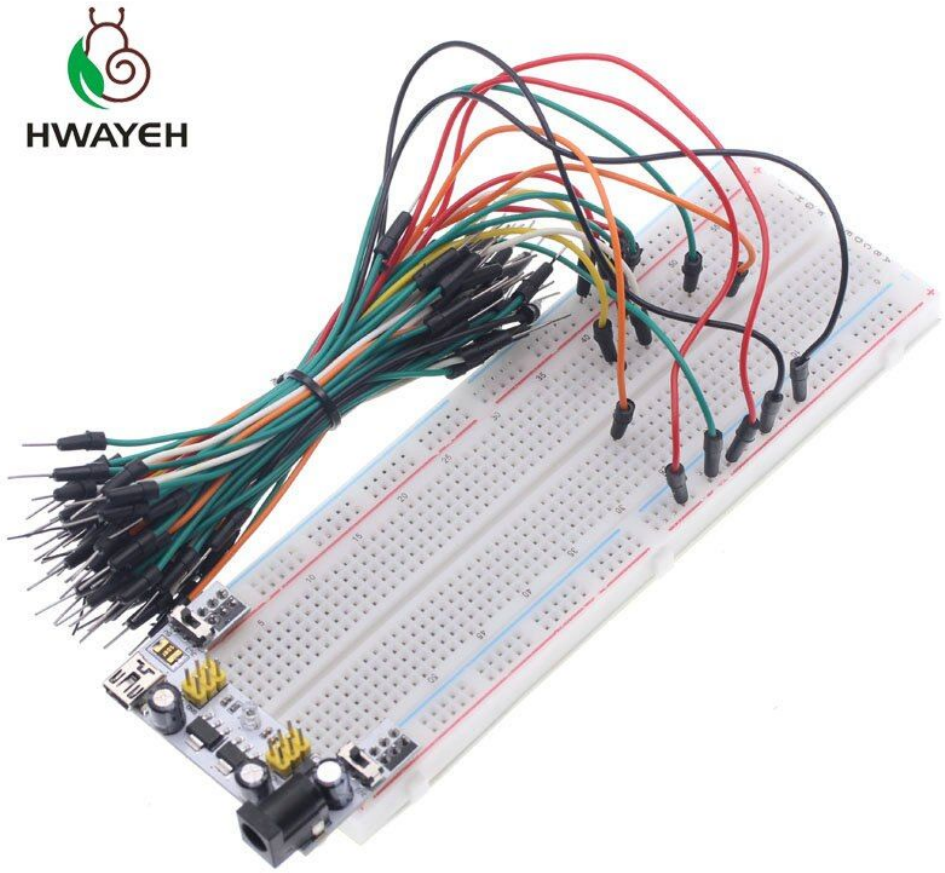
Arduino: Mega2560 + WiFi R3 ATmega2560+ESP8266



Description:
This is the microcontroller that I used for this project best part about this is that it has an ESP8266 on board. Using different settings of DIP switch on the controller both Arduino Mega and ESP8266 can be programmed separately. ESP8266 and Arduino Mega Can communicate via Serial3. This allows the robot to be able to connect to the WiFi Router. This can be programmed to accept commands over the WiFi Network.

Price: USD 11.94
Buy Now:
https://www.aliexpress.com/item/32967155571.html?spm=a2g0s.9042311.0.0.27424c4dBMQoIU

Breadboard + MB102 White Power Module + Wires



Description:
This is the breadboard that I used for the robot. The power module allowed the robot to be powered by power banks and I needed more wires than what was provided in SIK.

Price: USD 3.45
Buy Now:
https://www.aliexpress.com/item/32697390495.html?spm=a2g0s.9042311.0.0.27424c4dBMQoIU

SSD1306 OLED Display Module 128X64 I2C

Description:
Had burnt the LCD with SIK so had to buy a new one to display info related to the robot like, robot's IP, ESP's IP etc.

Price: USD 1.76
Buy Now:

https://www.aliexpress.com/item/32962785036.html?spm=a2g0s.9042311.0.0.27424c4dBMQoIU

Dupont Jumper wire 10CM Male to Male + Female to Male + Female to Female

Description:
Some more shorter wires for better cable management.

Price: USD 1.78
Buy Now:
https://www.aliexpress.com/item/32962785036.html?spm=a2g0s.9042311.0.0.27424c4dBMQoIU

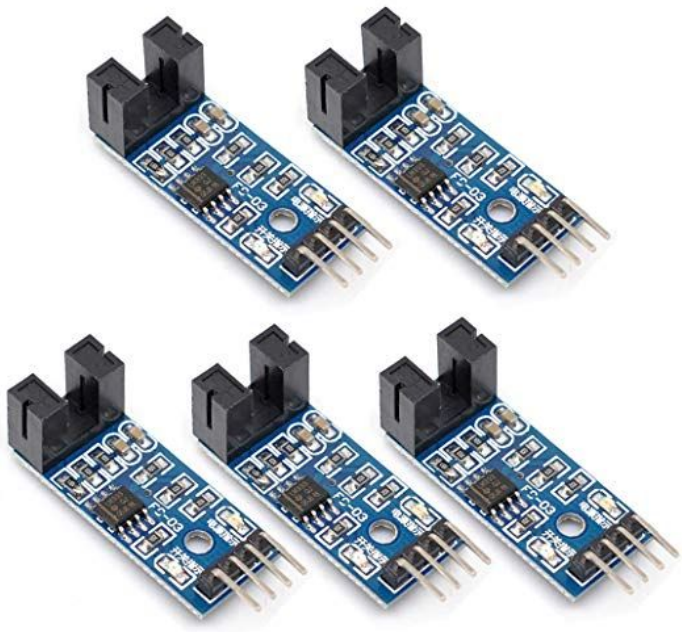
Elegoo 120pcs Multicolored Dupont Wire 40pin Male to Female, 40pin Male to Male, 40pin Female to Female Breadboard Jumper Wires

Description:
Some longer wires….. :P

Price: USD 9.00

Buy Now: https://www.amazon.ca/gp/product/B01EV70C78/

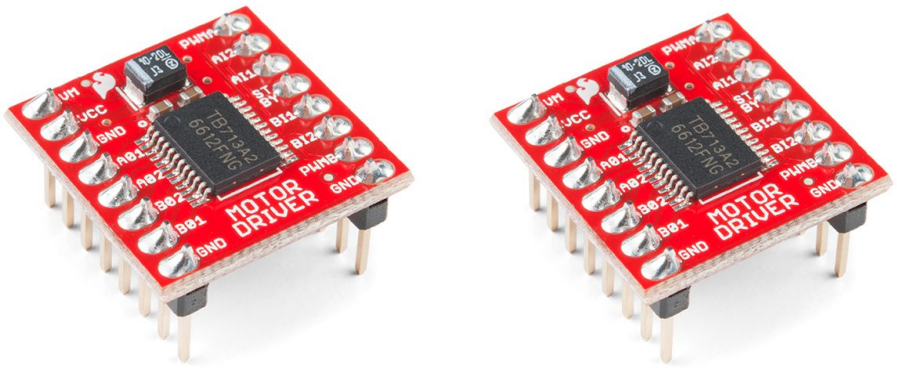
Willwin 5pcs LM393 Chip Motor Measuring Comparator Speed Sensor Module Slot Type IR Optocoupler for MCU Arduino



Description:
Break beam opto-sensors used for odometry with slotted disc encoders that came with the robot chassis.

Price: USD 9.39
Buy Now: https://www.amazon.ca/gp/product/B0776RHKB1/

2pc SparkFun Motor Driver - Dual TB6612FNG (with Headers)



Description: Motor driver are used to control the 4 motors on Aroid. Each motor driver is capable of handling 2 motors.

Price: 2 x USD 5.45

Buy Now: https://www.sparkfun.com/products/14450

2pcs Lofty Ambition MG90S 9g Metal Gear Servos

Description:
2 Servos used to create the head of the robot with a camera and an Ultrasonic sensor.

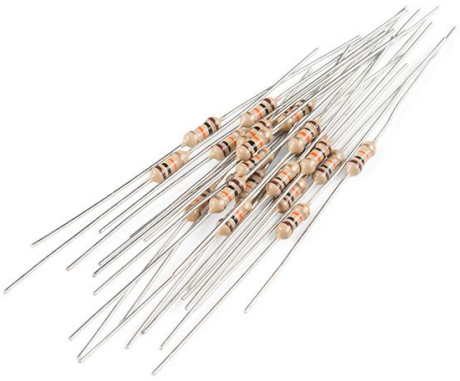
Price: 2 x USD 2.94
Buy Now: https://www.amazon.ca/gp/product/B0776RHKB1/

Ultrasonic Distance Sensor - HC-SR04

Description:
The ultrasonic sensor is used to detect obstacles by measure the distance of the obstacle in front of it

Price: USD 3.95 (Was part of SIK)
Buy Now: https://www.sparkfun.com/products/13959

Resistor 10K Ohm 1/4 Watt PTH - 20 pack



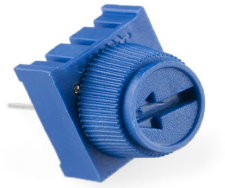
Price: USD 1.20 (Was part of SIK)
Buy Now: https://www.sparkfun.com/products/14491

Mini Photocell

Description:
Photoresistor is used to measure the light levels in the environment in which Aroid is located.

Price: USD 1.50(Was part of SIK)
Buy Now: https://www.sparkfun.com/products/9088

Trimpot 10K Ohm with Knob



Description:
This potentiometer is used to control the volume of the buzzer

Price: USD 0.95(Was part of SIK)
Buy Now: https://www.sparkfun.com/products/9806

Mini Speaker - PC Mount 12mm 2.048kHz

Description:
Aroid uses the Buzzer to play sounds when it encounters an obstacle.

Price: USD 1.95
Buy Now: https://www.sparkfun.com/products/7950

APPENDIX: ARDUINO INTERRUPTS

This is a great article that I found online that discusses interrupts for Arduino:
https://www.allaboutcircuits.com/technical-articles/using-interrupts-on-arduino/

I would also recommend you to watch this video to understand interrupts better.
https://www.youtube.com/watch?v=QtyOiTw0oQc

APPENDIX: I2C PROTOCOL

I2C protocol is a communication protocol used by low-speed devices. A large number of devices compatible with Arduino support I2C for example servo controllers, OLEDs, LCDs etc. It can be also used to make two Arduino talk to each other. I2C simplifies the communication process a lot.

I2C solves a lot of problems if you are short on pins for example for using the LCD that came along SIK we had to use a lot of wires for communication that significantly reduced the number of pins on the board that can be used for other components. Whereas the OLED screen that I used in my project used I2C , therefore, it just has 4 pins: 2 for communication and two for power. I2C can also allow using multiple Arduinos for the same project in a master-slave architecture. This is sometimes necessary in some cases.

I2C only uses 2 pins (SCL and SDA) for communication and multiple I2C devices can share the same pins. Each device has its own I2C address that is used to interact with it in code. Manufacturers usually list the most likely I2C address of the components but if you have multiple components of the same type each of them gets assigned a different address that necessarily might not be the same as the one provided by the manufacturer. I2C Scanner sketch (https://gist.github.com/tfeldmann/5411375) comes in really handy in such situations.

In order to connect a component that uses I2C protocol is the following:

1. Supply power using the VCC and GND pins/ as required.
2. Connect SDA and SCL pins of the component to SDA and SCL pins of the Arduino.
3. Determine the address of the device using the I2C Scanner sketch.
4. Use this address and library provided by the manufacturer(if any) to communicate with the component.

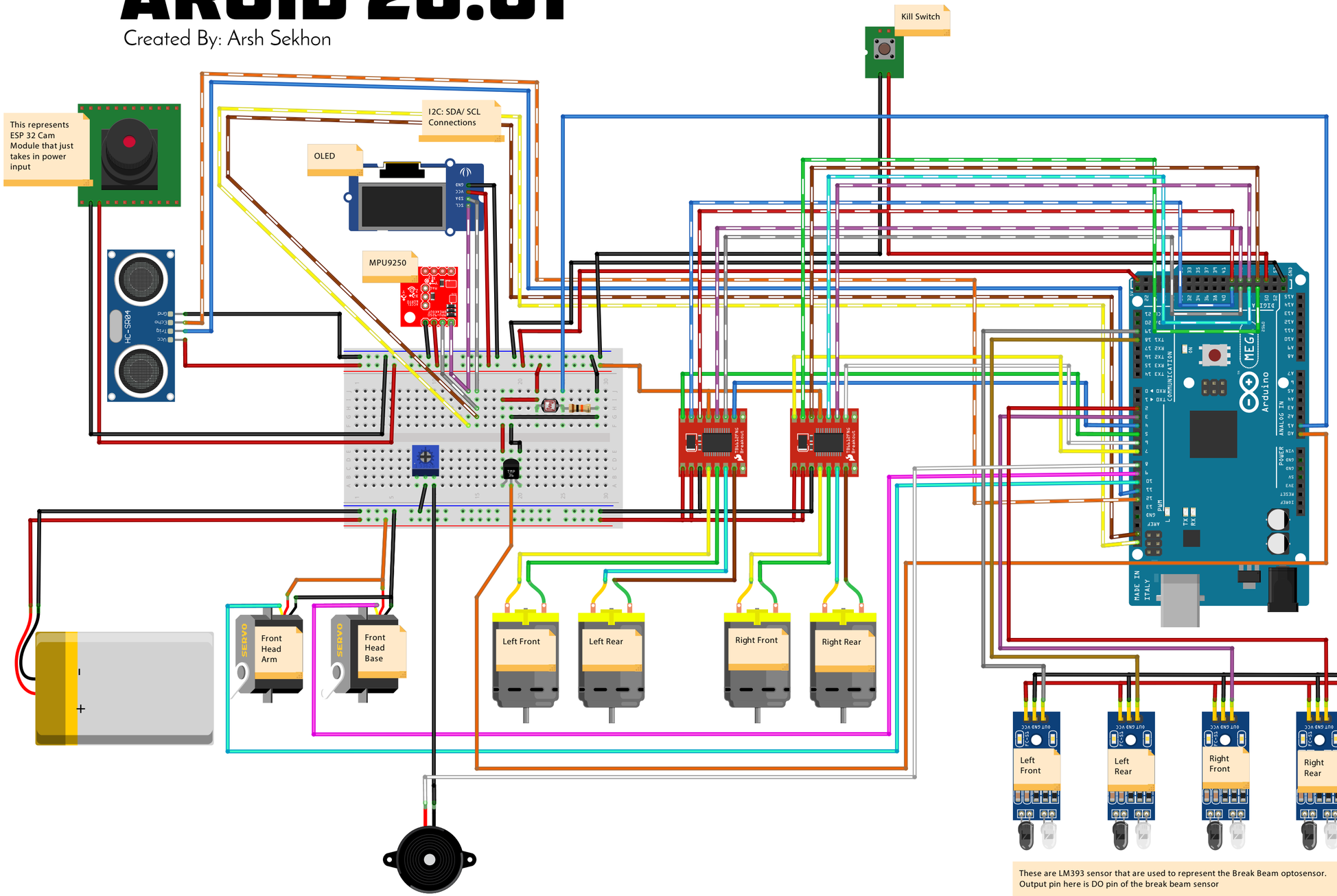
For more information on how I2C works please visit: https://i2c.info/

APPENDIX: CIRCUIT DIAGRAM

Refer to the next page for the circuit diagram.

AROID 26.01

Created By: Arsh Sekhon



PINS SUMMARY

TEMPERATURE_SENSOR_PIN A0
PHOTORESITOR_READ_PIN A1
BUZZER_PIN D8

LEFTSIDE_MOTOR_DRIVER_AIN1 D43
LEFTSIDE_MOTOR_DRIVER_AIN2 D42
LEFTSIDE_MOTOR_DRIVER_PWMA D5
LEFTSIDE_MOTOR_DRIVER_BIN1 D45
LEFTSIDE_MOTOR_DRIVER_BIN2 D44
LEFTSIDE_MOTOR_DRIVER_PWMB D4

RIGHTSIDE_MOTOR_DRIVER_AIN1 D48
RIGHTSIDE_MOTOR_DRIVER_AIN2 D49
RIGHTSIDE_MOTOR_DRIVER_PWMA D7
RIGHTSIDE_MOTOR_DRIVER_BIN1 D46
RIGHTSIDE_MOTOR_DRIVER_BIN2 D47
RIGHTSIDE_MOTOR_DRIVER_PWMB D6

ODOMETER_RIGHT_FRONT_PIN D3
ODOMETER_RIGHT_REAR_PIN D2
ODOMETER_LEFT_FRONT_PIN D19
ODOMETER_LEFT_REAR_PIN D18

HEAD_SERVO_BASE_PIN D9
HEAD_SERVO_ARM_PIN D10

ULTRASONIC_SENSOR_TRIG_PIN D11
ULTRASONIC_SENSOR_ECHO_PIN D12

KILL_SWITCH_PIN D51