Automatic Floor Cleaner Navigation Algorithm

- CSE3001 Project
- Madhukar Temba
- 19BAI1161

Introduction

The aim is to make an algorithm for a 3D printed floor cleaner robot which will make the robot navigate any room while avoiding obstacles and also know where it currently is with respect to the starting position.

Issues after literature survey

- **The robots don’t have a way of knowing their position in the room.**
- **They don’t have any mechanism to actively control their speed.**
- **Their speed may vary with the battery voltage.**
- **They need to be charged manually after every cleaning cycle.**
- Absolute position lost after being displaced by external factors.
- Ability to detect obstacles of all colours, shapes and sizes.

Issues in robot version 1

- **The current robot cleaner does clean well but repeatability of its steps is low. That means the robot cannot precisely execute an action every time.**

- The gyroscope sensor being tested suddenly stops responding due to unknown reason. Connected to main controller by I2C bus.

- **The robot tends to get stuck at platforms elevated at 2.25cm.**

Objective

We need to develop a way through which the robot can know the position of the charging station with high accuracy and also know if any obstacle is on its path so that it can avoid it.

Proposed Method

- INPUT: Some kind of sensor so that the robot could know what is in front of it. For that I first used an ultrasonic distance sensor with a combination of bump sensors to determine any obstacle ahead of the robot’s path. But after testing it was found that the ultrasonic sensor worked at a very specific angle due to which the robot kept bumping into stuff. To solve this another robot was built which uses 5 infrared proximity sensors arranged in the front side along with the bump sensors.

Proposed Method

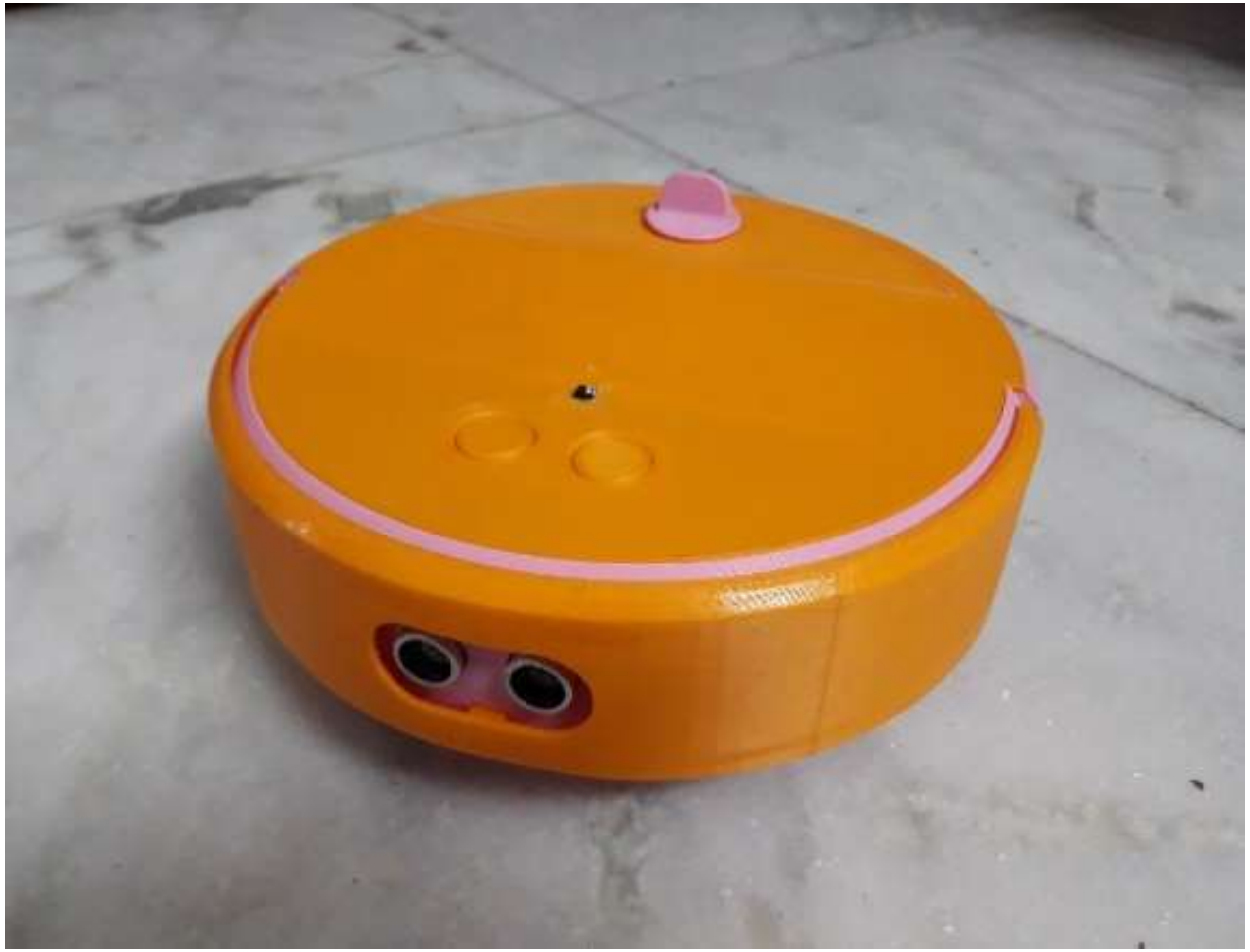
- METHOD: After considering all the inputs the robot will make a decision on how much to turn. The robot will follow a random navigation algorithm. It will also count the steps moved using which it will maintain x-coordinate, y-coordinate and angle data relative to the start position.
-

Proposed Methods

- OUTPUT: In the first version the robot had DC gear motors with an encoder which it used to move around. But as the encoder was having only 20 steps it was difficult to maintain accurate position. Also, the robot got stuck very often as the tires were fixed and would lift off the ground if the robot accidently got stuck on a slightly elevated surface.

- To address this the new robot uses stepper motors with 4096 steps and spring-loaded tires to always make good contact with the ground even on uneven surfaces.

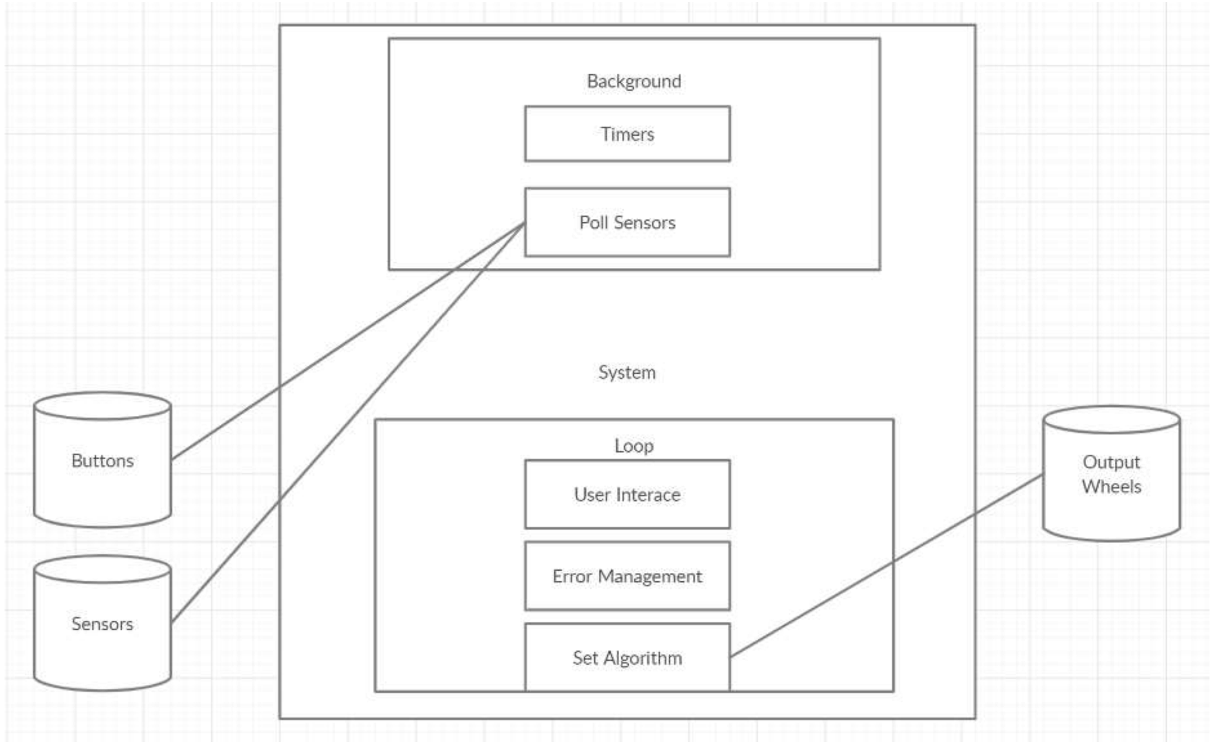
Version 1



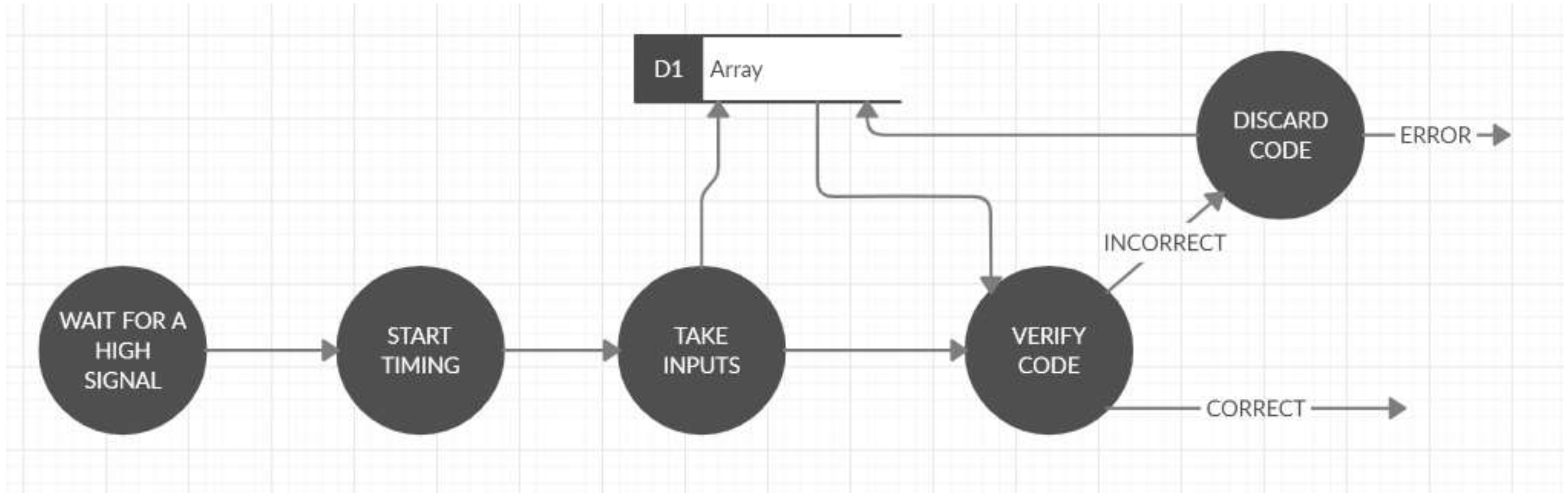
Version 2

Proposed Design

- The algorithm takes all the inputs from the front sensors and then gives an angle to turn to avoid the coming obstacle:



Proposed Design



Modules

1. Speed management module
2. Rotate module
3. Move forward module
4. Check sensors module
5. User interface module

Modules

6. Sound module
7. IR sensor module
8. Power management module
9. Charging module
10. Deep sleep module

As there are 2 versions of the robot both using different hardware some of the modules have 2 versions.

Module 1: Speed management

For Version 1: As the DC motors spin at high speed there needs to be module to control the power given to each one to make the robot go in a straight line and also maintain a set speed.

INPUT: The module will receive the pulses from the two encoders each attached to one motor. It also knows that the encoder has 20 steps.

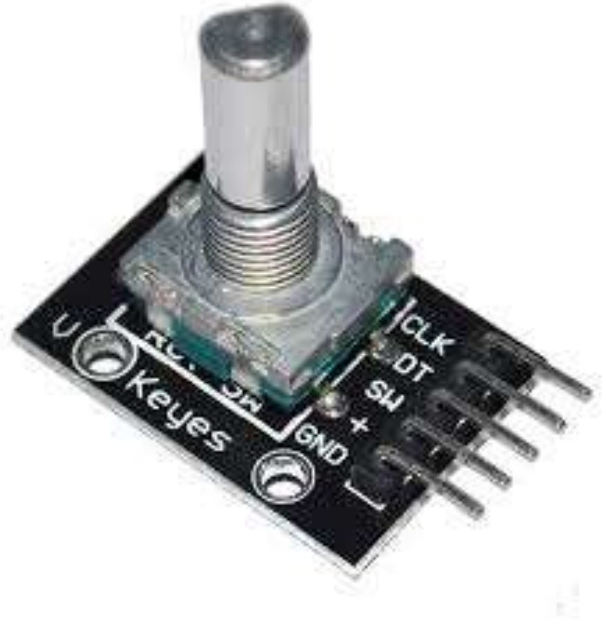
PROCESSING: Now as the encoder will give pulses, we will check the delay between the 2 consequent pulses.

Now if the delay is greater than set time that means the wheel is rotating slower than required. So, we increase the power given to the motor by a tiny amount.

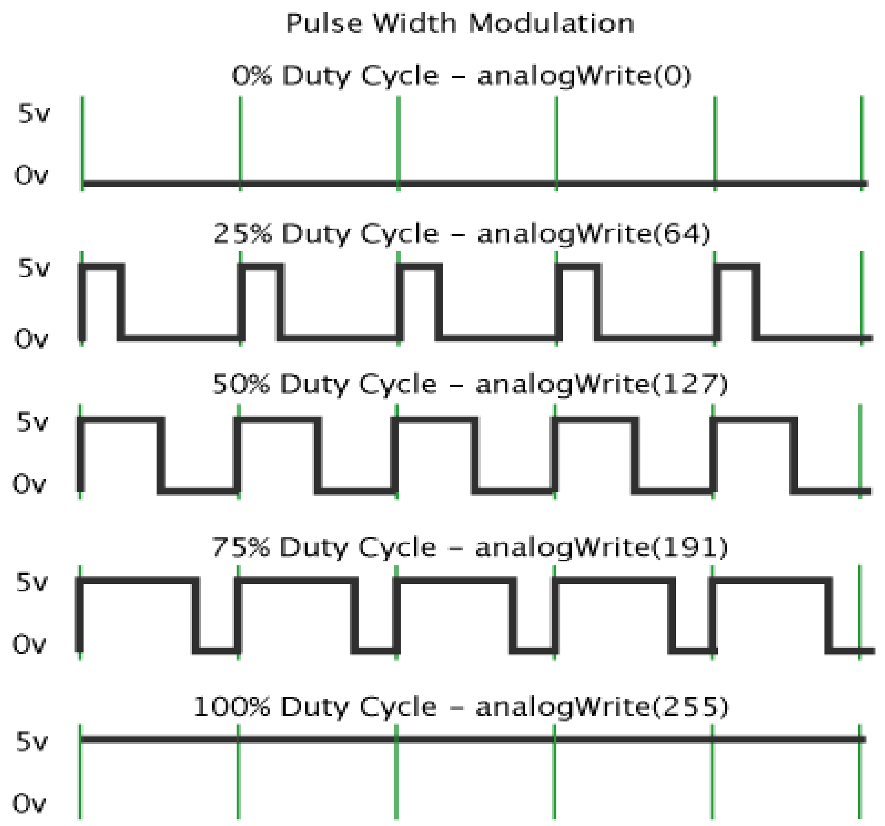
If the delay is shorter than the set time then it means that the wheel has sped up. So, we decrease the power given to the motor by a tiny amount. Now as this is done many times a second the motor speed is very stable.

The microcontroller has the encoder pins attached as an interrupt so that it can respond to the pulses immediately no matter where it is in the code.

Now as the encoders used are mechanical, they have a problem of debouncing. That means when the switch metal plates contact each other they bounce a bit giving incorrect pulses and too many of them halting the processor. So, a timer is used to solve that issue in the interrupt code which will only take the input after a specific amount of time. Here is a picture of the encoder used:



Pulse width modulation is used to vary the power given to the motors, meaning that we turn the control pin on and off very rapidly (at a frequency of 1000Hz) and change the on time to give a specific amount of power over a period of time.

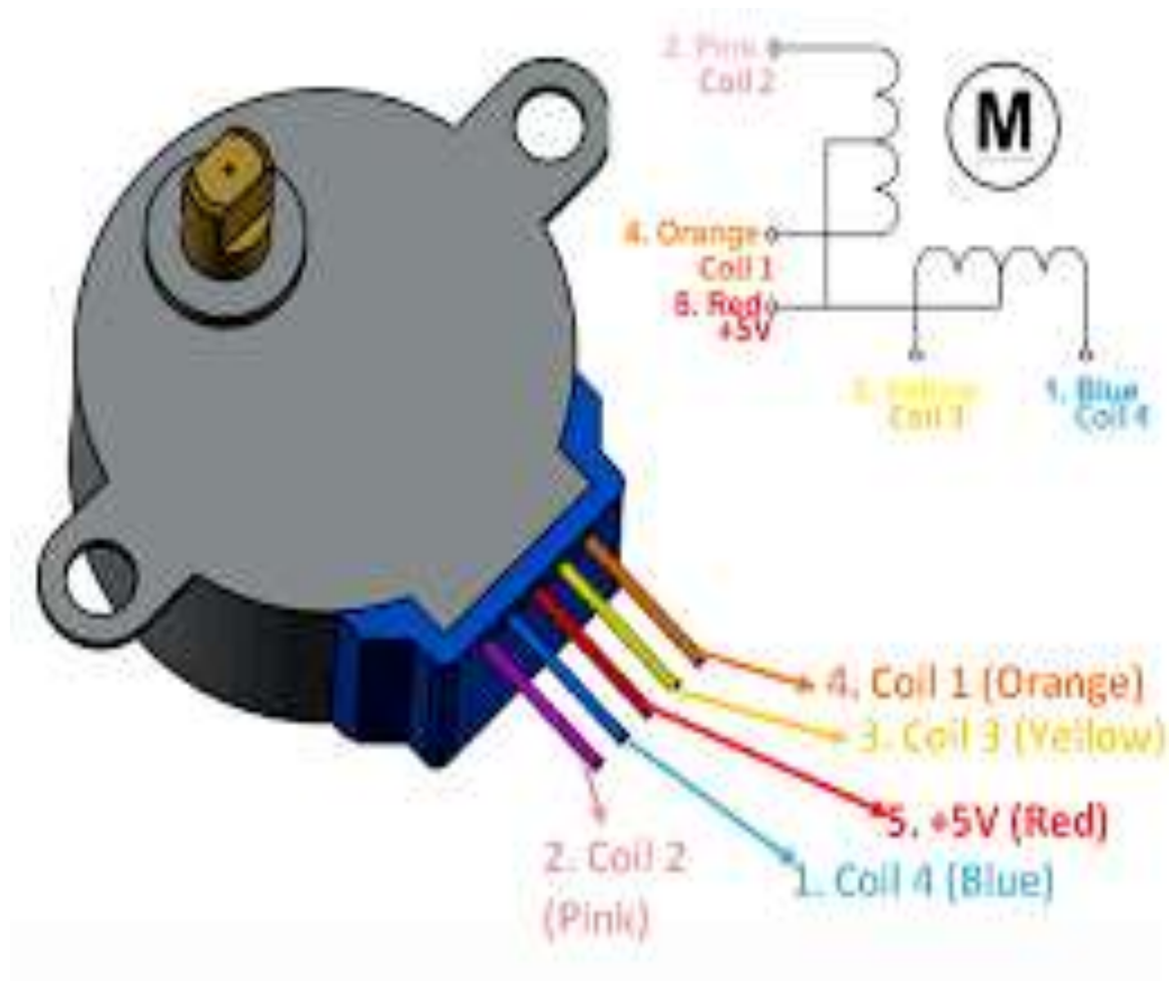

Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
25% Duty Cycle – analogWrite(64)
50% Duty Cycle – analogWrite(127)
75% Duty Cycle – analogWrite(191)
100% Duty Cycle – analogWrite(255)
5v
0v

The speed management module also has another function which will run continuously to check the encoder is giving pulses or not meaning the wheel is stuck or not. If a wheel is stuck for over 5 seconds the robot will beep and stop. Then it will increase the power to that specific wheel and try again until the wheel starts rotating. Now if maximum power is given to the motor and if it still does not spin that means the robot is stuck and it shuts down after giving a final turn off beep.

OUTPUT: Now the signal is given to the motor driver which controls the motor speed.

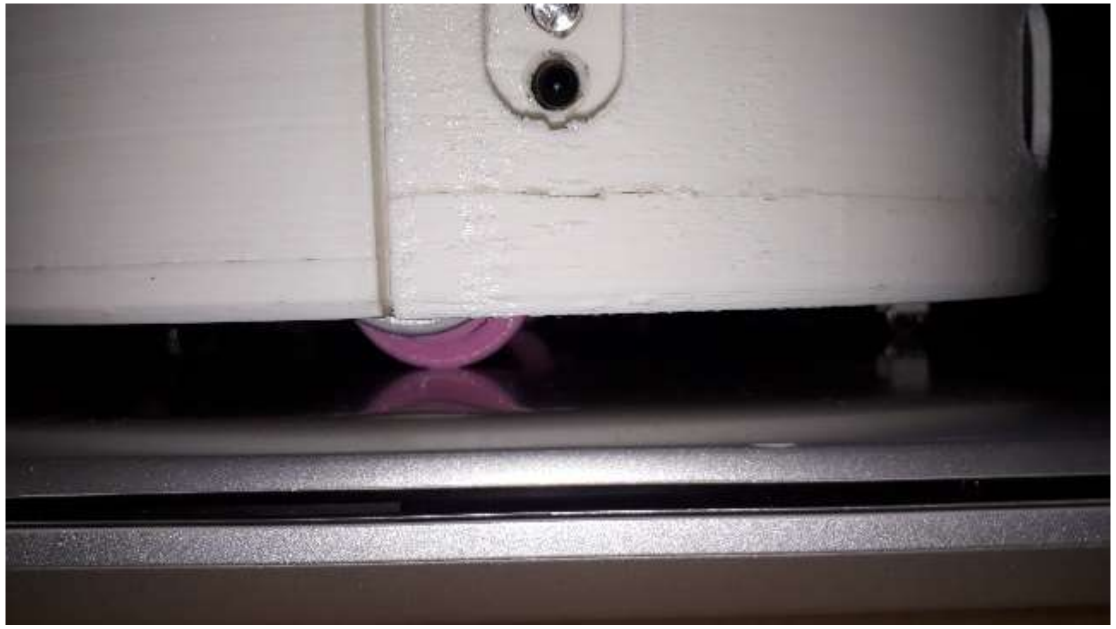
For Version 2: As the second version uses stepper motors the speed management is not a big issue as the speed of the motors is very slow and stable.

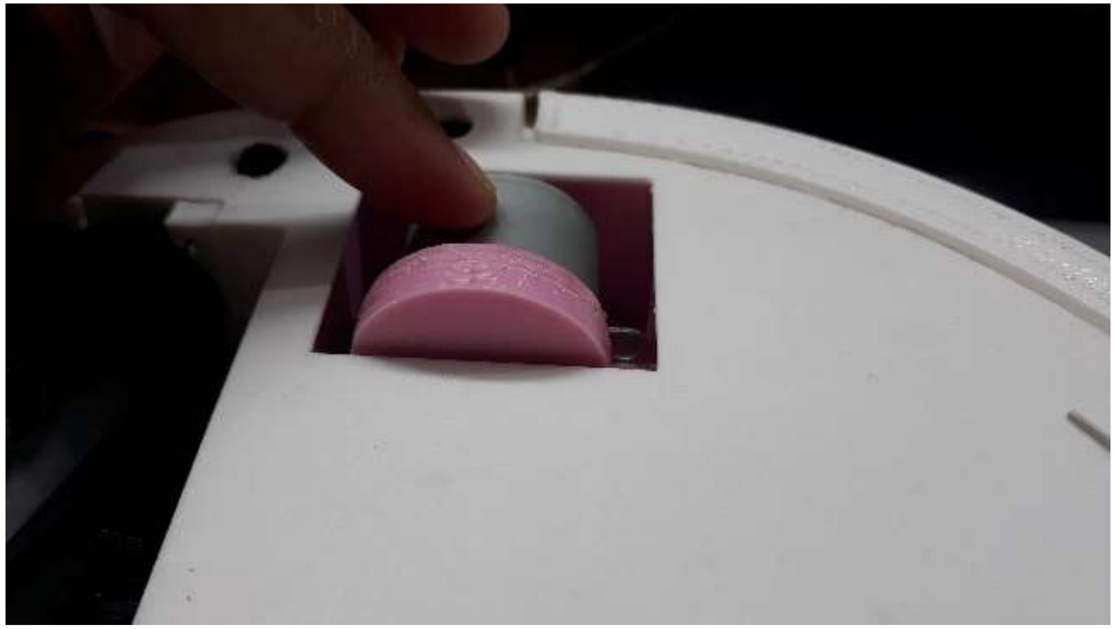
To set a speed value we only vary the delay between two consequent steps of the motor.


2. Pink
Coil 2
M
4. Orange
Coil 1
5. Red
+5V
3. Yellow
Coil 3
1. Blue
Coil 4
4. Coil 1 (Orange)
3. Coil 3 (Yellow)
5. +5V (Red)
2. Coil 2
(Pink)
1. Coil 4 (Blue)

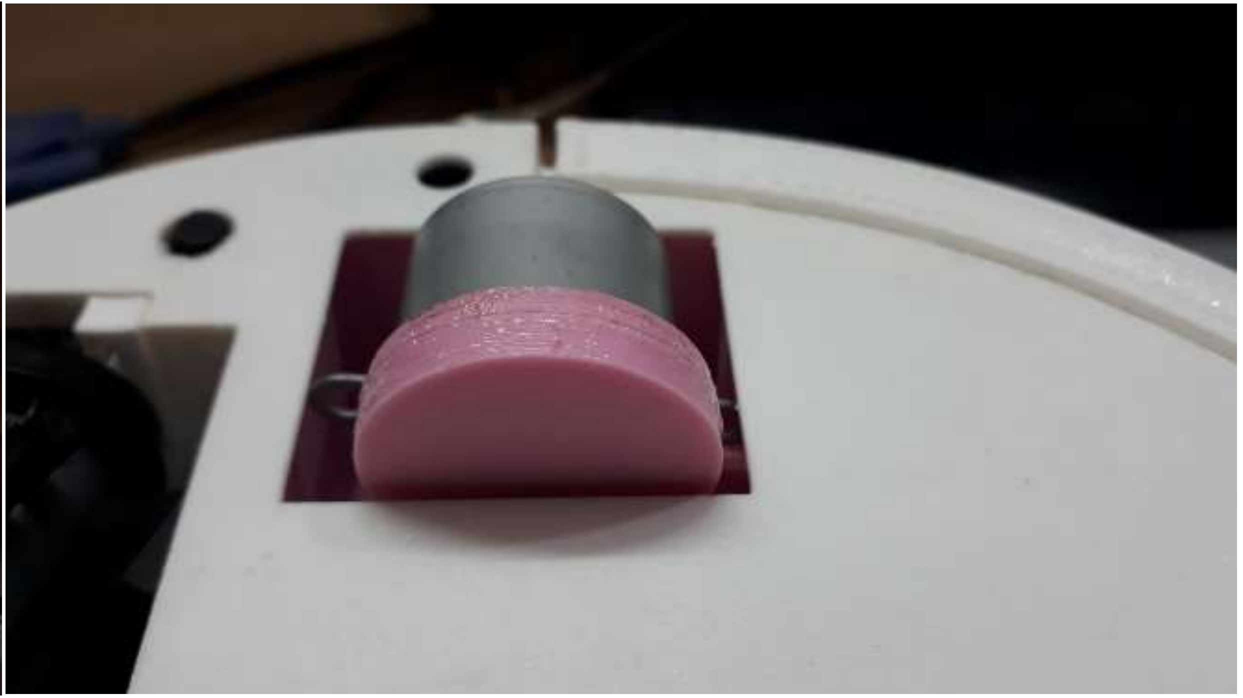
Also, to solve the issue of maintaining good contact with the ground the tire mechanism is spring loaded with the tires always wanting to stick out of the body maintaining good contact with the ground.

When on level surface:





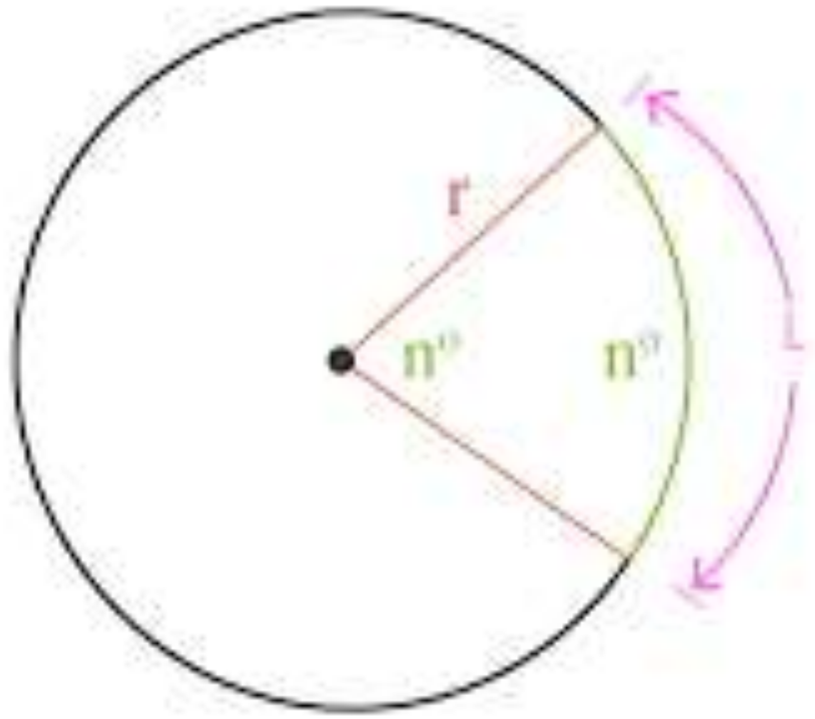
When on elevated surface:



Module 2: Rotate

- This module is used to rotate the robot by a specific angle.
- **INPUT:** It requires the angle you want to rotate, distance between the two tires, the diameter of the tire and the number of steps in the encoder or stepper motor.

- **PROCESSING:** It first calculates how many steps are required to complete the rotation. To calculate this, we know that:
- Distance between tires = tiredist;
- Diameter of the tire = tiredia;
- Length of an arc of a circle given the angle:



$$L = \frac{n^\circ}{360^\circ}(2\pi r)$$

Number of steps = nsteps;

Given angle = deg ;

Steps required = x; (We have to find this)

As we know all of this, we can use this information to find the number of steps (Let it be x) required:

$$\pi \times tiredia \times \frac{x}{nsteps} = 2\pi \times tiredist \times \frac{deg}{360}$$

So, the value of x is:

x = $deg \times tiredist \times \frac{nsteps}{360 \times tiredia}$

Now, the sign of the given angle is checked and according to the sign the robot rotates clockwise or anti-clockwise. Note that steps cannot be negative so if it is negative then it is first converted into positive. The current angle is noted after completion of the rotation by summing previous angle and this angle.

OUTPUT: This module turns the robot by a specific angle by rotating the tires in opposite directions and counting the steps.

Module 3: Move forward

This module is very similar to the rotate module just this time the formula and direction of wheels change.

INPUT: This module requires the distance to be moved, diameter of the tire and number of steps in the encoder or stepper motor.

PROCESSING: It first calculates how many steps are required to move the distance. To calculate this, we know that:

Diameter of the tire = tiredia;

Number of steps = nsteps;

Distance to move = dist;

Steps required = x;

So,

$$dist = \frac{x}{nsteps} \times \pi \times tiredia$$

So, the value of x is,

$$x = \frac{nsteps \times dist}{\pi \times tiredia}$$

Now, the sign of the given distance is checked and according to the sign the robot moves forward or backward. Note that steps cannot be negative so if it is negative then it is first converted into positive.

This value is used to update the current x and y co-ordinates.

The new x co-ordinate is given by:

$$x = x_0 + number\ of\ steps\ moved \times \cos(current\ angle)$$

The new y co-ordinate is given by:

$$y = y_0 + number\ of\ steps\ moved \times \sin(current\ angle)$$

OUTPUT: This module moves the robot forward or backward a specific distance by spinning both wheels in the same direction. It also notes the new x and y positions.

Module 4: Check sensors

This module is used to get all the inputs from the sensors and buttons.

INPUT: It requires all the pin numbers to measure their values.

PROCESSING: As the processor is very fast, we take all the inputs in one row and store their state in specific variables to be accessed later in the code.

For version 1: This module first takes the input from the ultrasonic distance sensor:

The distance sensor is connected to the microcontroller via 2 pins. One is the transmit pin and the other one is the receive pin.

First, we send a signal to the transmitter and after that we measure the time taken for the receive pin to go high. Now,

Speed of sound = Speed = 343m/s;

Time taken = Time;

Distance = x;

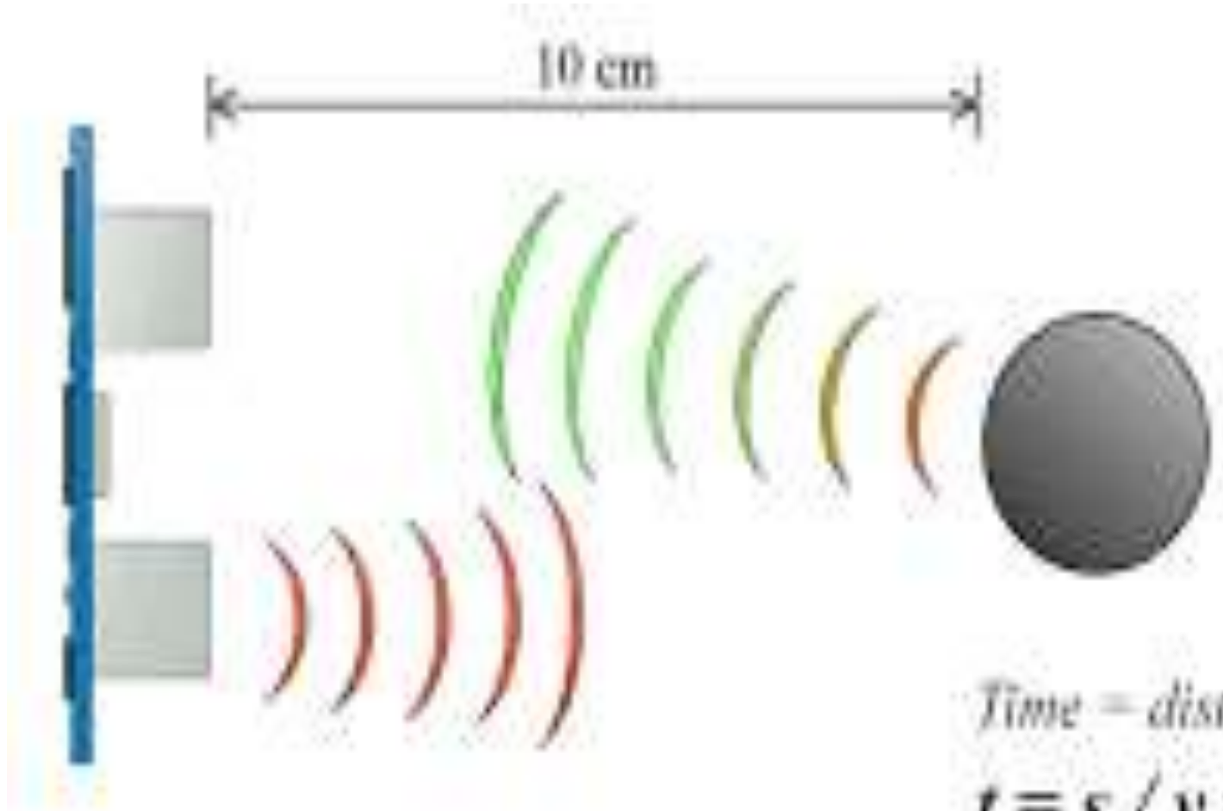
As we know,

$$Speed = \frac{Distance}{Time}$$

$$Distance = Speed \times Time$$

But the distance calculated will be twice the original distance so we have to divide it by 2,

$$Distance = Speed \times \frac{Time}{2}$$



speed of sound:

$v = 340\ m/s$

$v = 0{,}034\ cm/\mu s$

Time = distance / speed:

$t = s / v = 10 / 0{,}034 = 294\ \mu s$

Distance:

$s = t \cdot 0{,}034 / 2$

By using this we get the value of the distance.

Then we see if the distance is less than the given threshold. If yes then the distance variable gets changed.

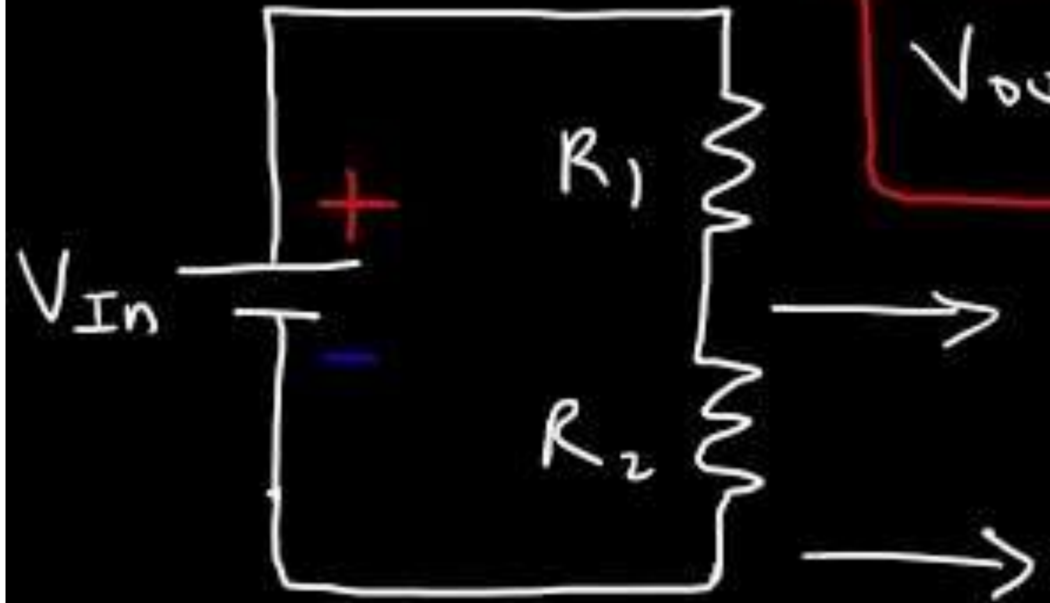
After that we take input from the bump switches and the top switch for and change their variables accordingly.

To check the voltage, it uses the Analog to Digital Converter (ADC) in the micro controller to calculate the voltage. The ADC can be of 10-bit or 12-bit resolution.

It also checks the voltage of the battery connected to the microcontroller with a voltage divider as the microcontroller works on 5V or 3.3V and the battery voltage is higher.

For this we need to use a voltage divider,

Voltage Dividers



$$\boxed{V_{out} = V_{in}\left[\frac{R_2}{R_1+R_2}\right]}$$

We need to select specific R1 and R2 values to get Vout under 5V or 3.3V.

For version 2: Most of it is the same as it does not have an ultrasonic sensor that part is not there. Also, the top switch is a capacitive touch switch which is triggered only when there is a sudden change in the value of the pin to avoid false triggers.

To detect the touches, the touch pin gives a high frequency signal to that pin and measures the signal. Now if a finger touches it then the capacitance changes and becomes:

$$New\ capacitance = Old\ capacitance + Finger\ capacitance$$

So, the signal gets affected due to this and a touch is detected.

Now as the version 2 is having a dual core processor this task will be handled by the second core and core 1 will run the main program.

OUTPUT: This module refreshes the state of the sensors and stores them in some variables which will be accessed later in the program.

Module 5: User interface

The user interface has 2 buttons which are used to control the robot.

INPUT: The button variable refreshed by check sensors module, LED pin number and buzzer pin number.

PROCESSING: These are the various functions which are done by the buttons,

For version 1:

1. Long press button 1 to turn on the robot. This works by using a combination of PNP and NPN transistors, when the button is pressed, it temporarily closes the connection and the microcontroller turns on. After a few seconds the microcontroller pulls a pin high which will activate the transistors. This will bypass the switch and the microcontroller turns on the LED and gives a beep to let you know it is powered on.
2. Press button 2 once to start cleaning.
3. Press button 2 again to pause cleaning.
4. Press and hold button 2 to turn off the robot. When this command is given then the microcontroller pulls the pin latching the circuit low there by breaking the circuit and turning off the robot.

The LED and the buzzer are also a part of the user interface.

The LED is turned on and off smoothly with the help of PWM and using for loops to gradually increase the brightness.

The pin the buzzer is connected to turns on and off very rapidly at from 1000Hz to 3000Hz creating sound. The buzzer is driven by a transistor as it draws more current than the pin can supply. The buzzer code is part of the Arduino library and only one line of code is required to use it.

For version 2: It has an WS2812B led which requires its own library to control. It also has a DAC with an amplifier to playback WAV files. This will be discussed in the sound module.

OUTPUT: This module lets the user control and know the status of the robot.

Module 6: Sound

This module is only present in robot version 2. This module is used to play WAV audio files. This module uses the ESP32 DAC Audio library made by XTronical. https://www.xtronical.com/dac-audio-4-2-1-released/

INPUT: It requires the ESP 32 microcontroller DAC pin 25, amplifier power pin number and the library.

PROCESSING: The ESP 32 has two 8-bit DACs (Digital to Analog converters) on GPIO (General Purpose Input Output) pins 25 and 26. The pin 25 is attached to the PAM8403 5V amplifier.

1. The file to be played first needs to be translated into a WAV file.
2. Then the file gets converted into unsigned 8-bit PCM code by the software Audacity.
3. After that another software is used to translate that code into C language code.
4. Now we have the hexadecimal values in the WAV file which are stored in an array.
5. To play the file seamlessly, the library first copies some of it in a buffer then start playing the file.
6. We need to repeatedly fill the buffer to keep the file playing.

That is how the sound is played through the DAC.

As the sound module is not used constantly, we also have a transistor which controls the power to the amplifier to turn it off when not in use.

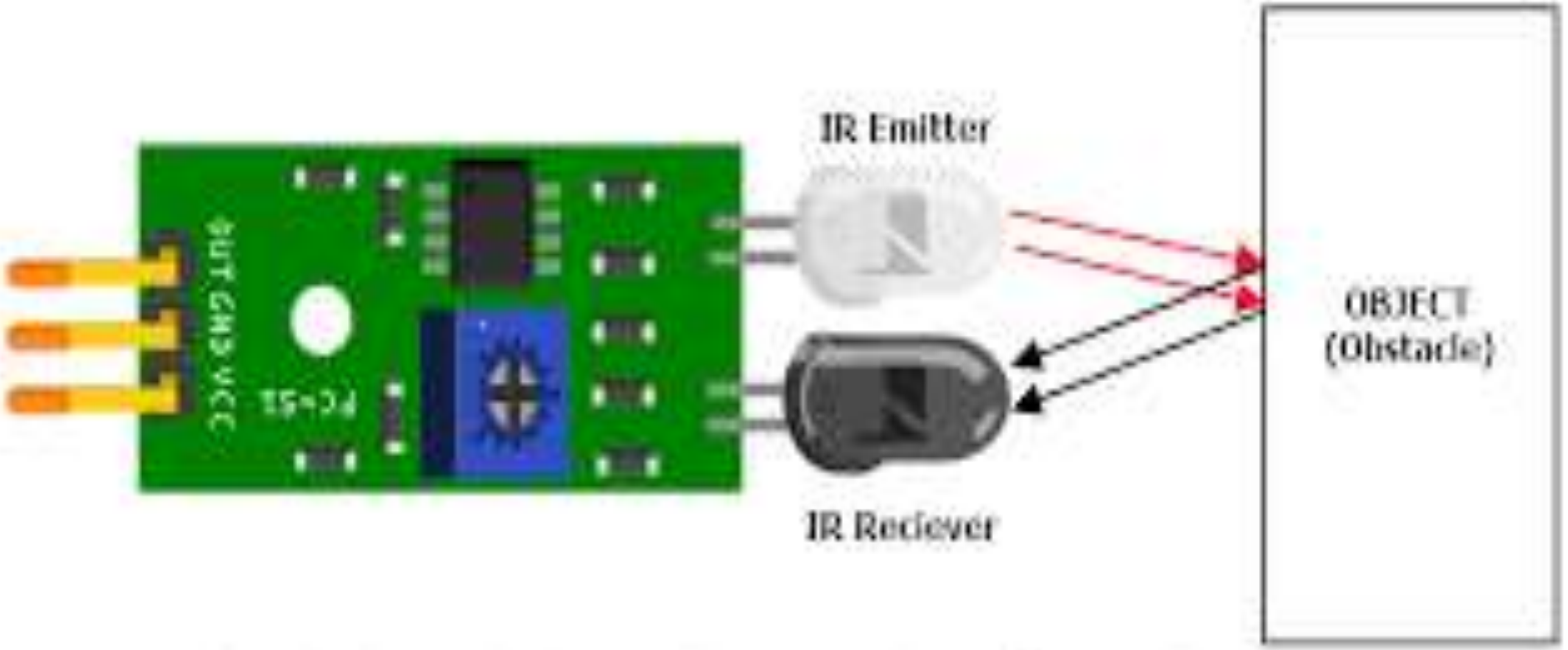
OUTPUT: The DAC pin is connected to the amplifier which is connected to the speaker. This module is used to play sounds.

Module 7: IR sensor

This module is only used in robot version 2. This module is used to get reliable readings from the 5 IR sensors mounted in front of the robot.

INPUT: The individual pin numbers connected to the 5 sensors and the IR led control pin.

PROCESSING: Each of the 5 IR sensors have a transmitter LED and a receiver LED, the receiver LED appears black as it is opaque in the visible spectrum but transparent in the infrared spectrum.



Note: Black surfaces absorbs light naturally. So it will not reflects much light on IR Reciever.
This concept is used in Line follower robot.

The emitted IR light reflects from an obstacle and comes back to the receiver LED. When enough IR light is received by the sensor it pulls the output pin low indicating an obstacle.

Now the problem is that a lot infrared light can come from other sources like IR TV remotes, light bulbs and most importantly the sun.

VectorStock®

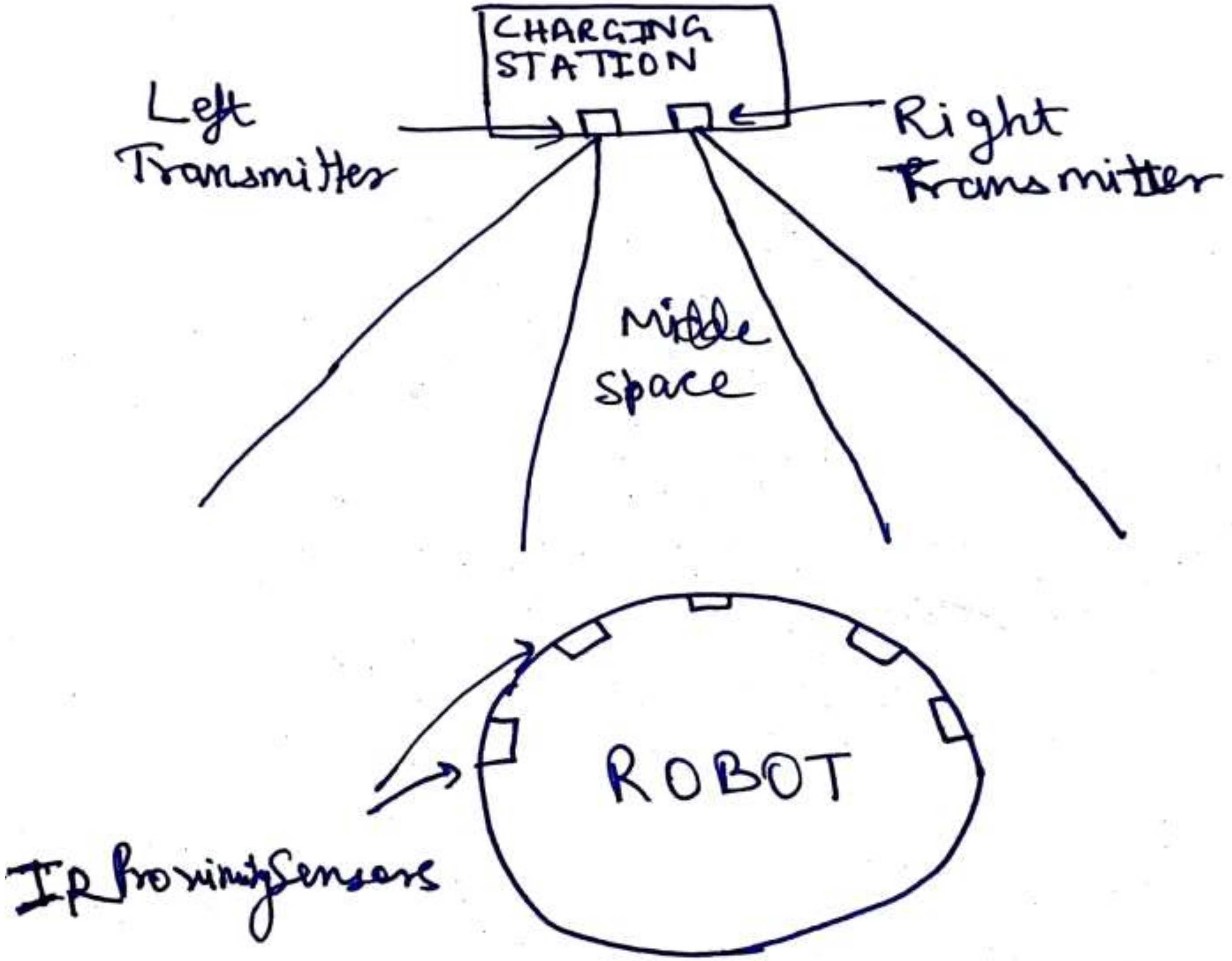
To avoid this, we control the IR transmitting LED.

1. We turn off the IR transmitter LEDs.
2. Then we take input from all 5 sensors.
3. If any sensor is still detecting an obstacle that means there is external IR light. We mark this sensor
4. We store the input from all the sensors in an array.
5. Then we turn on the IR transmitter LEDs and take the input again but we do not consider the input from the marked sensors this time.

6. If all the sensors are marked, we don’t consider any sensor and rely on the bump sensors.

This is how the IR sensor inputs are processed.

There is also a special mode where the IR sensors accept a binary code of fixed length from an external transmitter which is used for navigation to charging base.


CHARGING STATION
Left Transmitter
Right Transmitter
Middle space
ROBOT
IR Proximity Sensors

- When the robot reaches near the base station by backtracking it will turn off its IR LEDs and checks each sensor for the transmitted code.
- If the robot receives the left transmitter's code only then it will turn right a bit. It will turn left if it receives the right transmitters code.
- If it receives both transmitter's code or nothing then it will move forward.
- The robot will stop when it detects that it is charging.

As soon as the robot receives a high signal it starts its timing and stores the 8-bit code in an array.

Then it checks if the trailer is correct or not. If not then it rejects the code.

CUSTOM CODE:

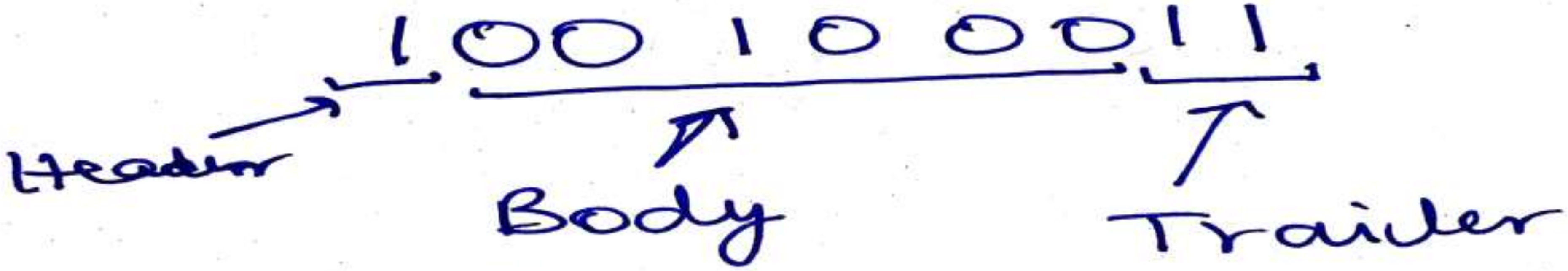
- The left and right transmitter will send a 9-bit code. The code will include a header and trailer.

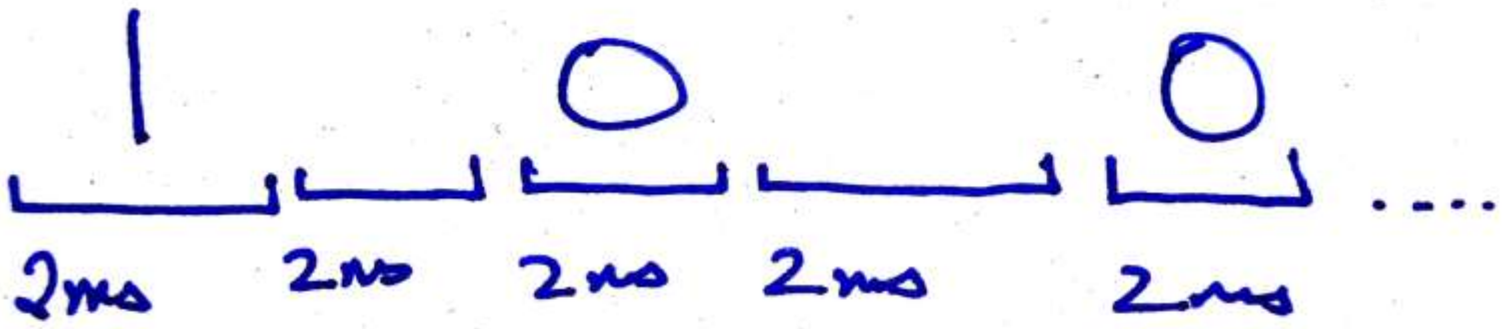
LEFT Transmitter will send: 100100011

RIGHT Transmitter will send: 100010011

The robot when in the centre will receive 100110011 or nothing.

CODE INCLUDES





This is the way by which the robot can get to the charging station.

OUTPUT: This module processes the IR sensor data for navigation.

Module 8: Power management

This module is used for controlling the power given to stepper motors, impeller fan, brush motor, LEDs and DAC.

INPUT: The pin numbers of stepper motors, impeller motor, brush motor, LEDs and DAC.

PROCESSING: The stepper motors, impeller motor, brush motor, LEDs and DAC are controllable via transistors and can be turned off by pulling the control pins low.

This module turns the robot off if the robot is idle for more than a minute.

Also, this module controls the power given to each component to prevent any damage.

OUTPUT: This module manages the energy being consumed by each component.

Module 9: Charging

This module is used to detect if the robot is charging properly or not.

INPUT: The charging pin number and battery pin number.

PROCESSING: It checks the voltage of the battery connected to the microcontroller and the supply voltage with a voltage divider as the microcontroller works on 5V or 3.3V and the input voltage and battery voltage are higher.

To check the voltage, it uses the Analog to Digital converter in the micro controller to calculate the voltage. The ADC can be of 10-bit or 12-bit resolution.

For this we need to use a voltage divider,

We need to select specific R1 and R2 values to get Vout under 5V or 3.3V.

For example:

Charging Voltage = 12V

ADC reference voltage = 5V

Then,

$$5 = 12 \times \frac{R2}{R1 + R2}$$

So, we get the relation

$$5 \times R1 = 7 \times R2$$

We can choose R1 = 100kΩ and R2 = 71.42kΩ

The robot when powered on checks if the charger is plugged in or not. If yes then it checks the voltage being received is high enough to charge the battery properly.

- If the voltage is correct then it beeps thrice to let the user know it is charging properly.
- If the voltage is incorrect then it beeps 2 twice with different tone to let the user know that the robot is not placed correctly over the charging station.

Using this the robot also knows that it is at the charging station and will know how to exit the charging station if it had to start cleaning.

OUTPUT: This module figures out that the robot is charging correctly or not also it is used to tell if the robot is at the charging station or not.

Module 10: Deep sleep

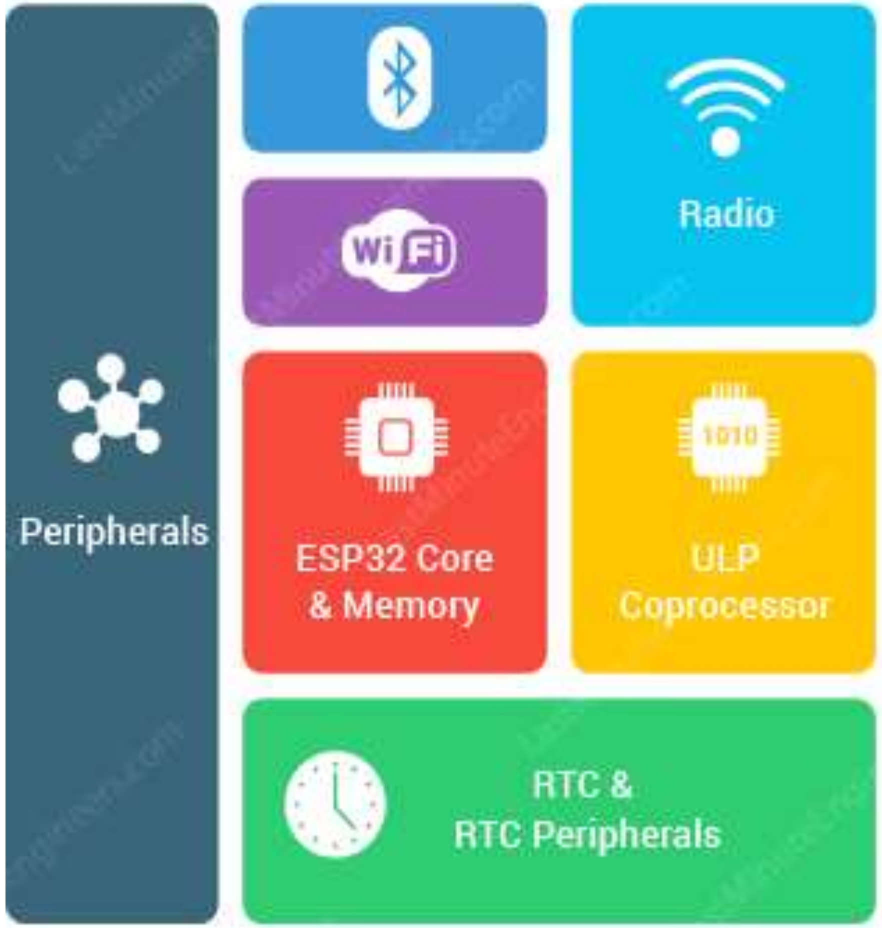
This module is only used in robot version 2. This module puts the robot in a low power consumption mode.

INPUT: The touch sensor pin.

PROCESSING: There are a number of modes of operation of ESP 32:

Last Minute ENGINEERS.com

Active Mode

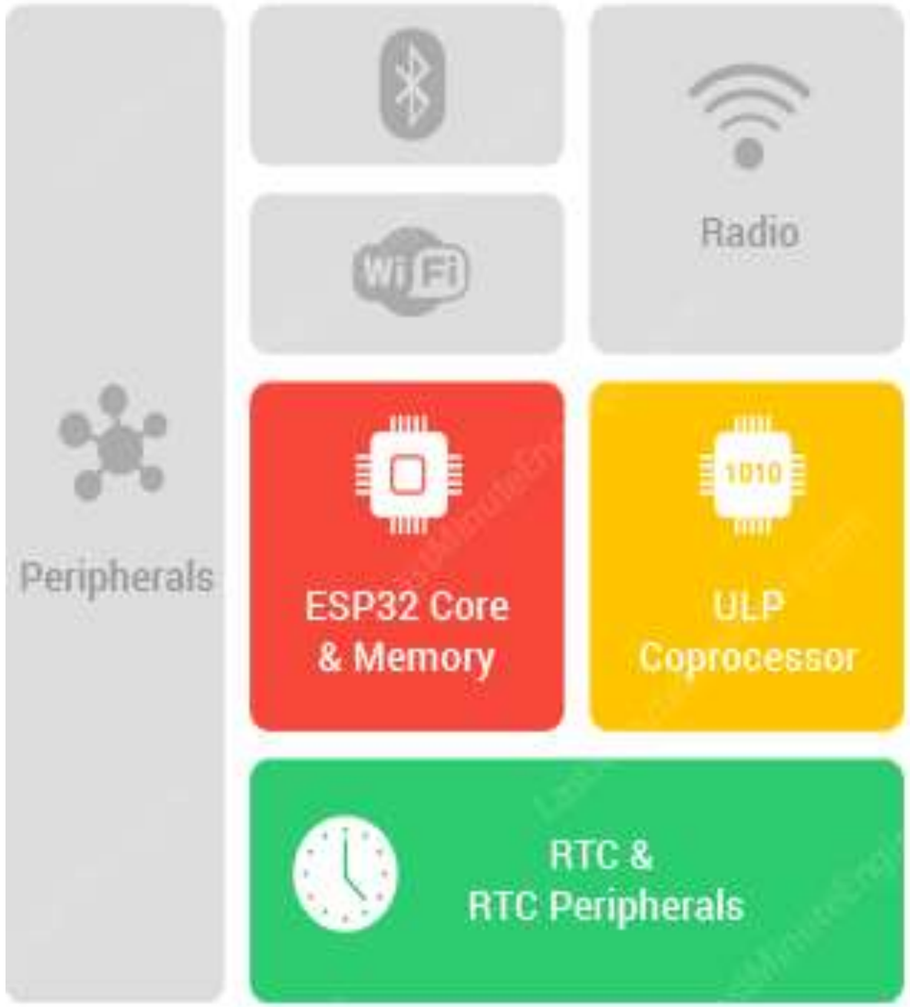


Active:	Inactive:
🟪 WiFi	
🟦 Bluetooth	
🟦 Radio	
🟥 ESP32 Core	
🟨 ULP Coprocessor	
⬛ Peripherals	
🟩 RTC	

Power Consumption:

160~260mA

Modem Sleep

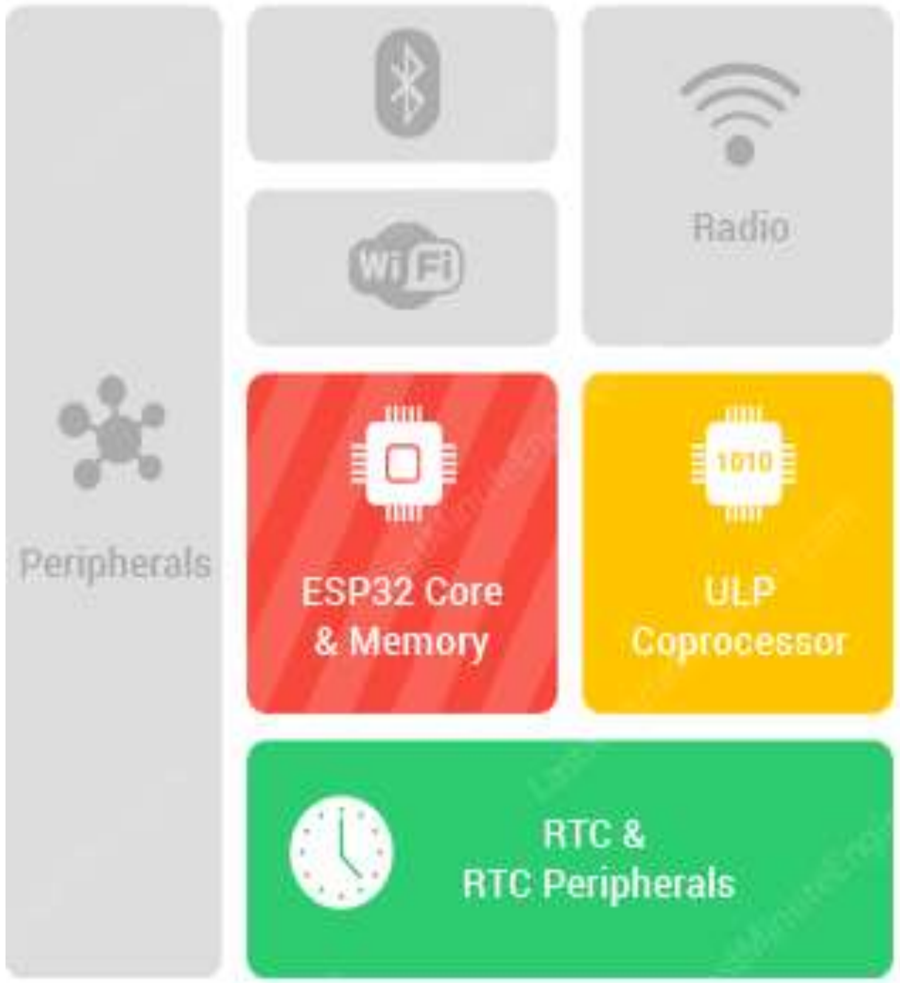


Active:	Inactive:
🟥 ESP32 Core	🟪 WiFi
🟨 ULP Coprocessor	🟦 Bluetooth
🟩 RTC	🟦 Radio
	⬛ Peripherals

Power Consumption:

3~20mA

Light Sleep



Last Minute ENGINEERS.com

Active:

- 🟨 ULP Coprocessor
- 🟩 RTC

Paused:

- 🟥 ESP32 Core

Inactive:

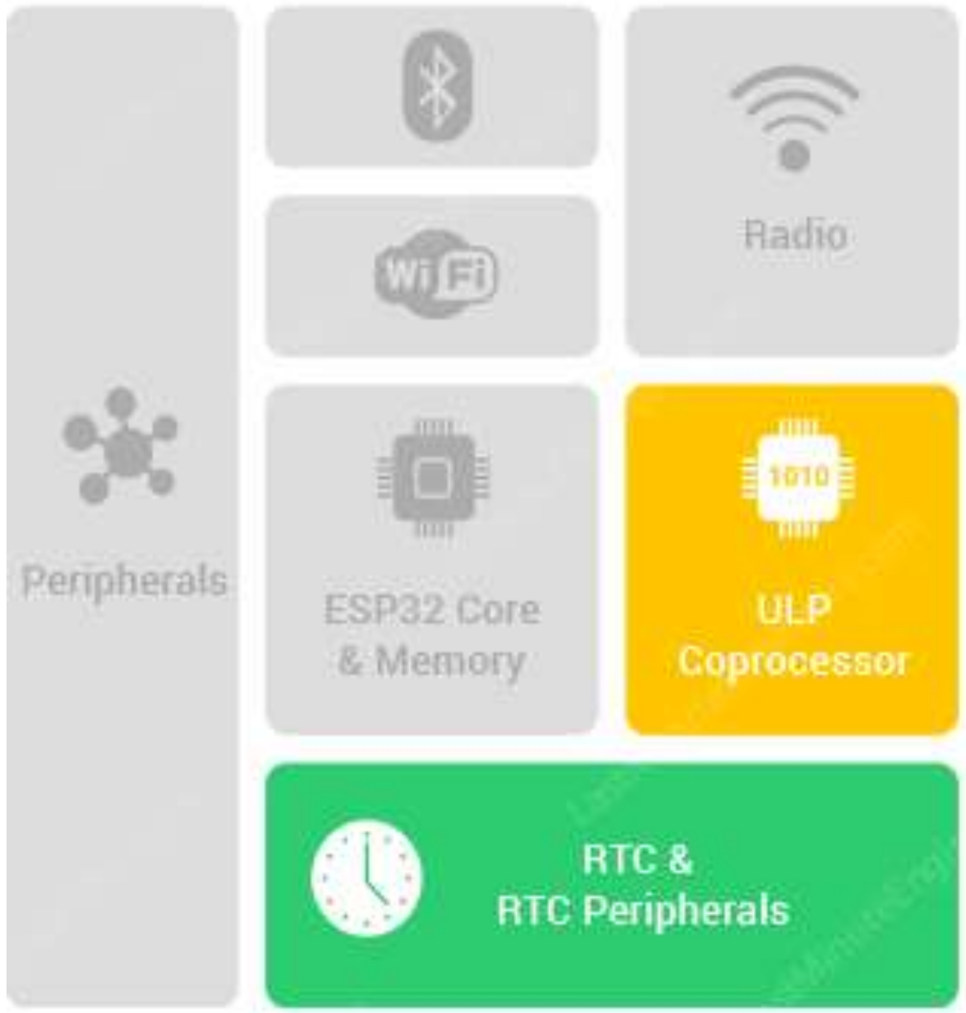
- 🟪 WiFi
- 🟦 Bluetooth
- 🟦 Radio
- ⬛ Peripherals

Power Consumption:

0.8mA

Last Minute ENGINEERS.com

Deep Sleeep



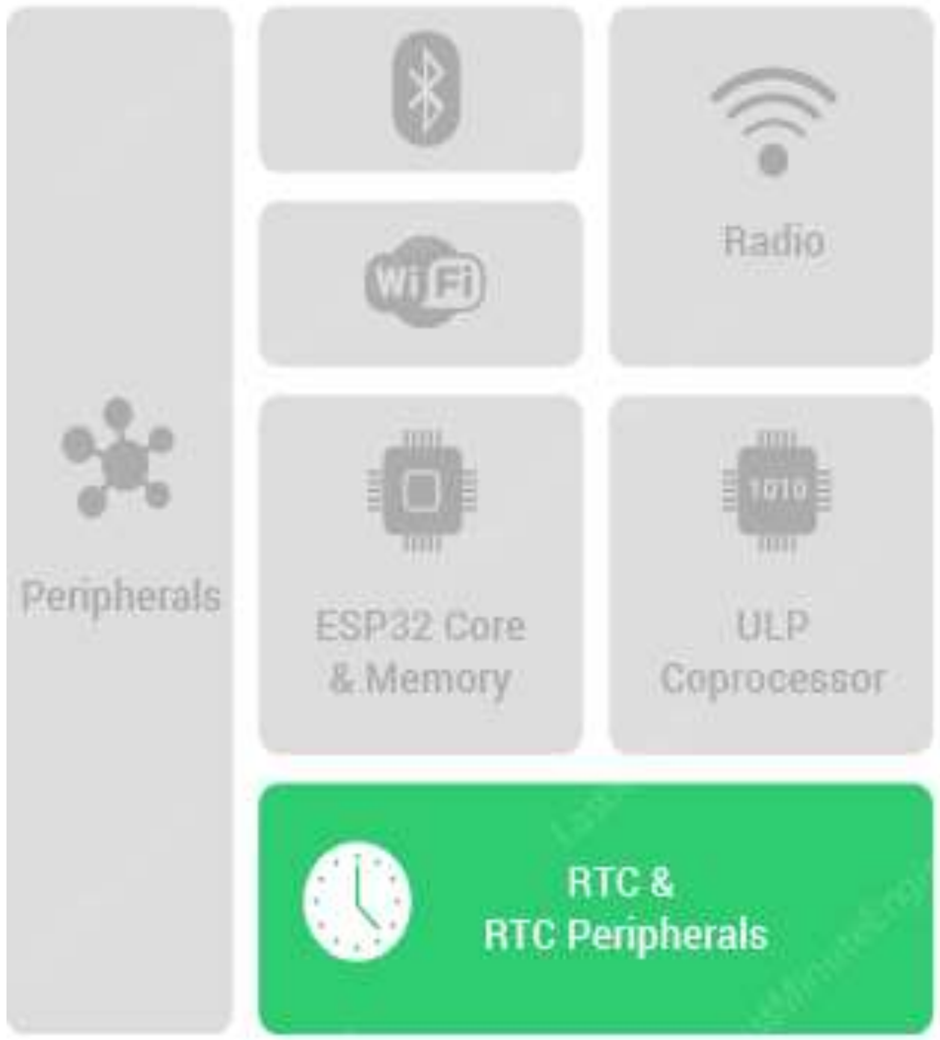
Active:	Inactive:
🟨 ULP Coprocessor	🟥 ESP32 Core
🟩 RTC	🟪 WiFi
	🟦 Bluetooth
	🟦 Radio
	🟦 Peripherals

Power Consumption:

10µA

Last Minute ENGINEERS.com

Hibernation



Active:

🟩 RTC

Inactive:

🟥 ESP32 Core

🟨 ULP Coprocessor

🟪 WiFi

🟦 Bluetooth

🟦 Radio

⬛ Peripherals

Power Consumption:

2.5μA

We put the microcontroller in deep sleep mode and use the touch sensor to wake it up when needed.

The ULP processor, RTC and RTC Peripherals are powered in this mode.

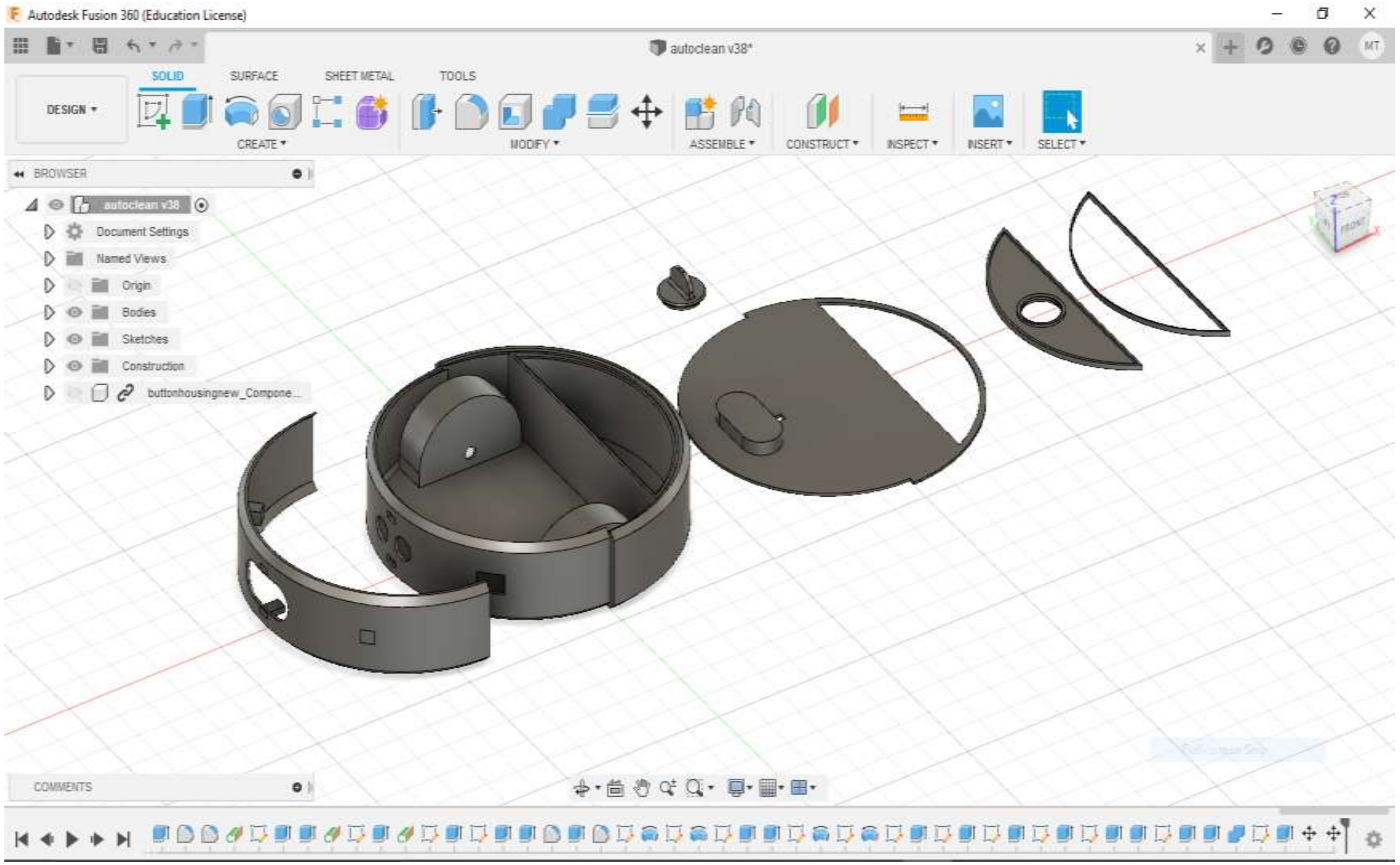
OUTPUT: Used to put and wake the microcontroller from deep sleep.

HARDWARE AND SOFTWARE REQUIREMENTS

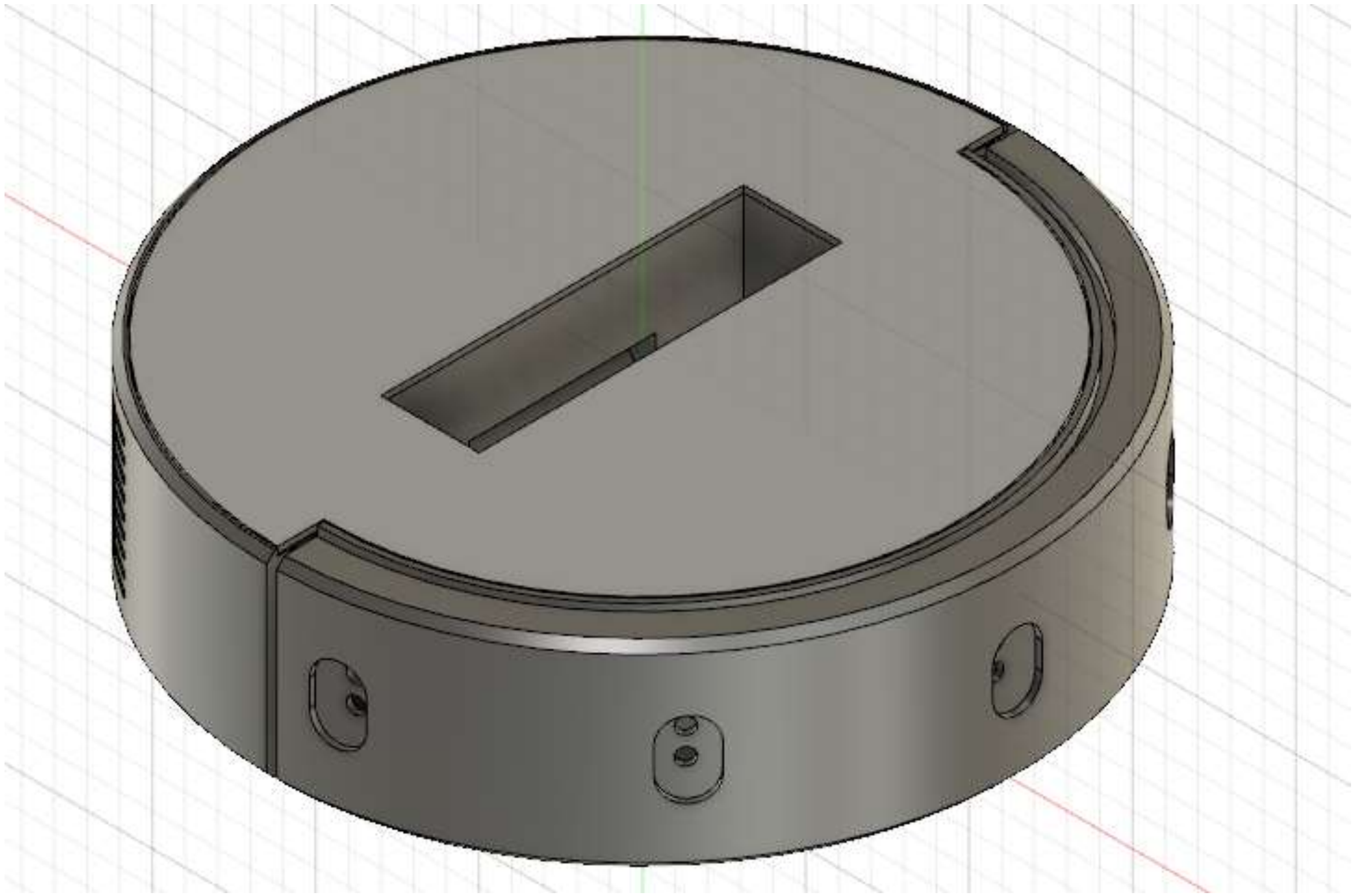
- Software was written in C language in Arduino IDE.

- All the parts for both the robots were designed in Fusion 360.

Version 1:


Autodesk Fusion 360 (Education License)
autoclean v38*
SOLID
SURFACE
SHEET METAL
TOOLS
DESIGN
CREATE
MODIFY
ASSEMBLE
CONSTRUCT
INSPECT
INSERT
SELECT
BROWSER
autoclean v38
Document Settings
Named Views
Origin
Bodies
Sketches
Construction
buttonhousingnew_Compone...
COMMENTS

Version 2:



After designing the robot parts were 3D printed with PLA plastic with 0.2 mm layer height.

The hardware components used for version 1 are:

1. Arduino Nano
2. Geared DC motors
3. Rotary Encoders
4. Battery management system(2S)
5. Two 18650 li-ion batteries
6. Green LED
7. Ultrasonic distance sensor
8. MPU6050
9. L298 motor driver
10. Limit switches
11. Push buttons
12. Buzzer
13. Resistors, transistors and diodes

The hardware components used for version 2 are:

1. ESP 32
2. 28-BYJ stepper motors
3. Stepper motor drivers
4. Connecting wire
5. Rotating brush
6. HEPA filter
7. Geared DC motors
8. Latching switch
9. 12V DC motor
10. Five infrared obstacle sensors
11. Two push buttons
12. Three 18650 li-ion batteries
13. Battery management system (3S)
14. Resistors, transistors and diodes
15. 5V buck converter
16. PAM8403 amplifier
17. Two speakers
18. RGB led

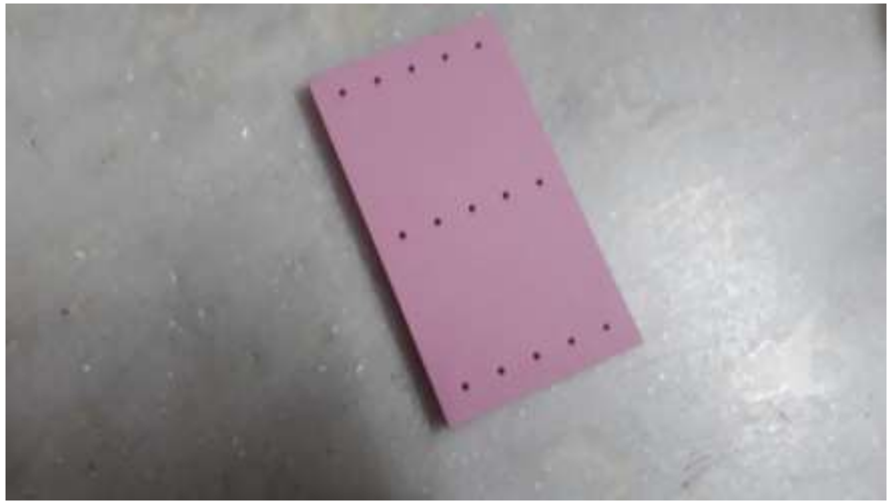
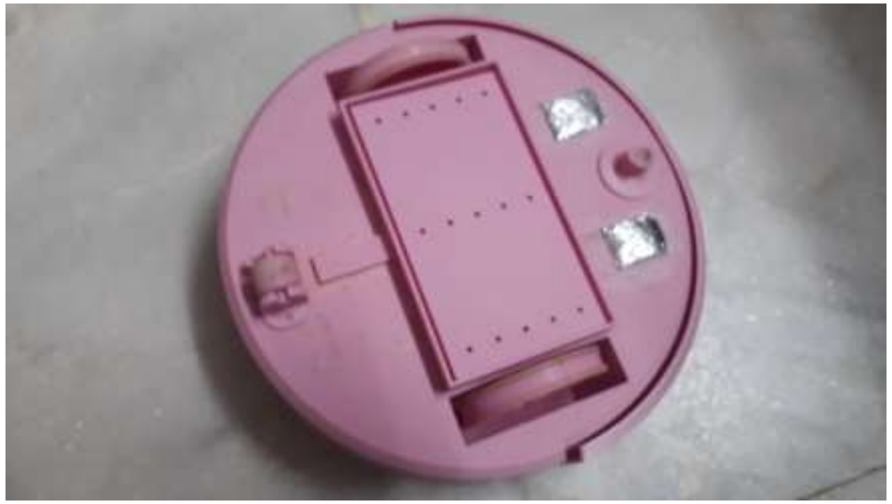
Cleaning mechanism of robot version 1:

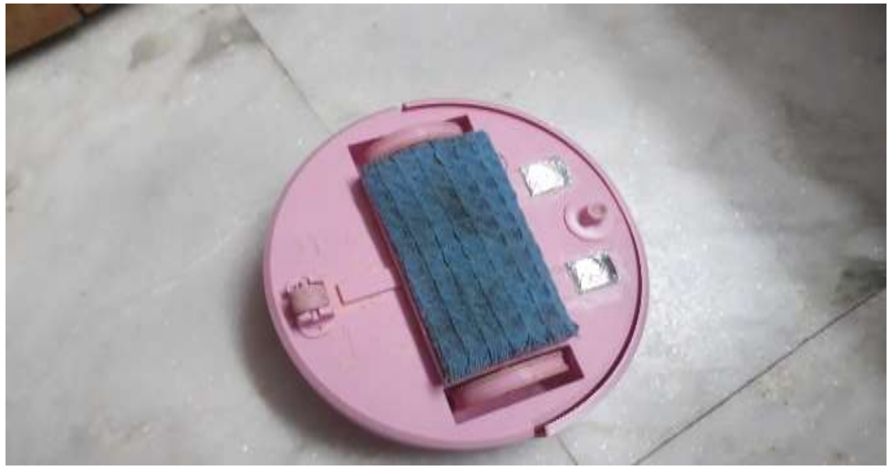
The robot version 1 has a 440ml reservoir where the user needs to add water and disinfectant.

After that it passes through a filter which removes any solid particles which may clog the system.

After that it goes into a channel which ends in two tiny holes where the solution drips slowly.

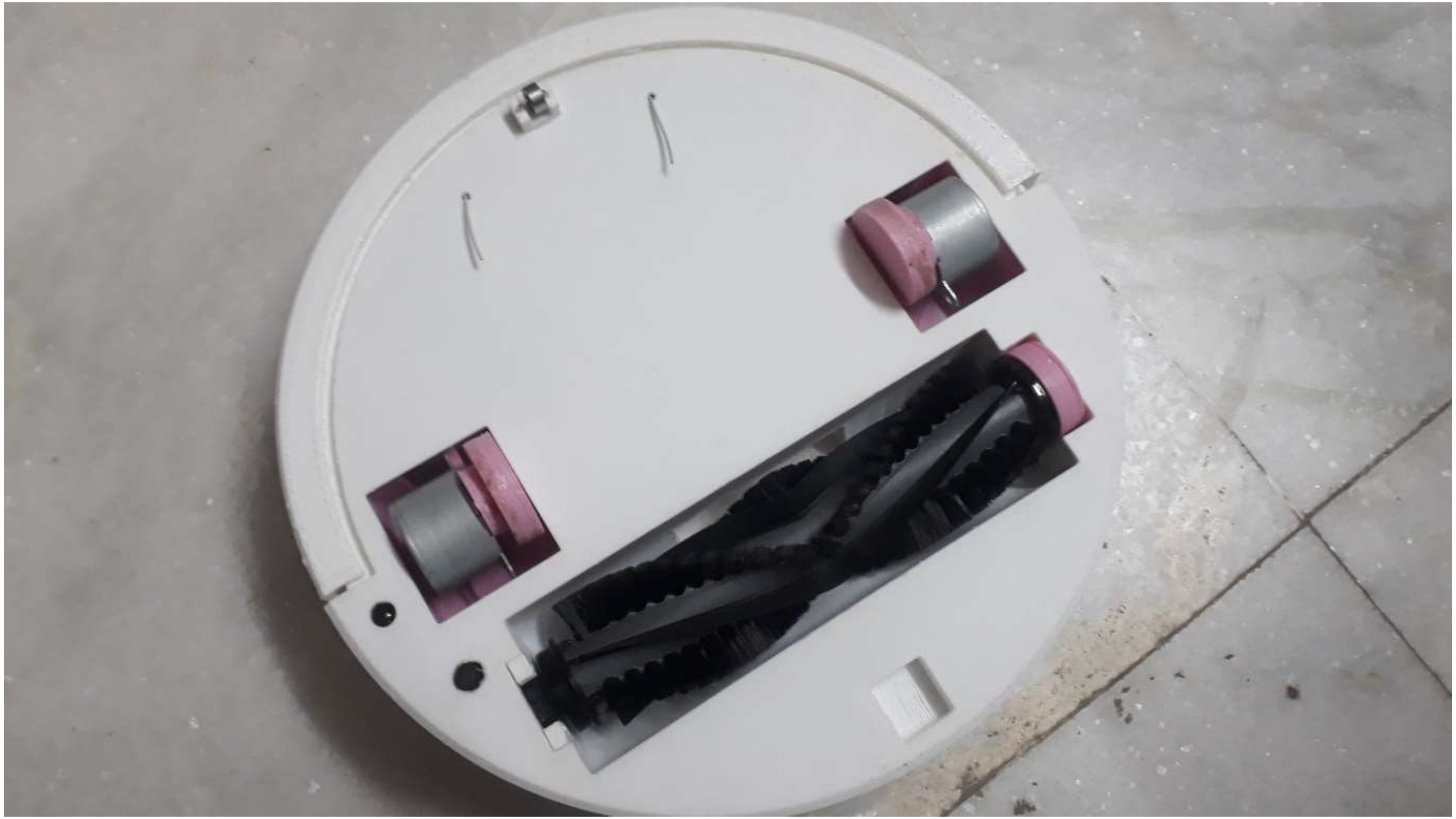
Then it soaks into the attached cloth.

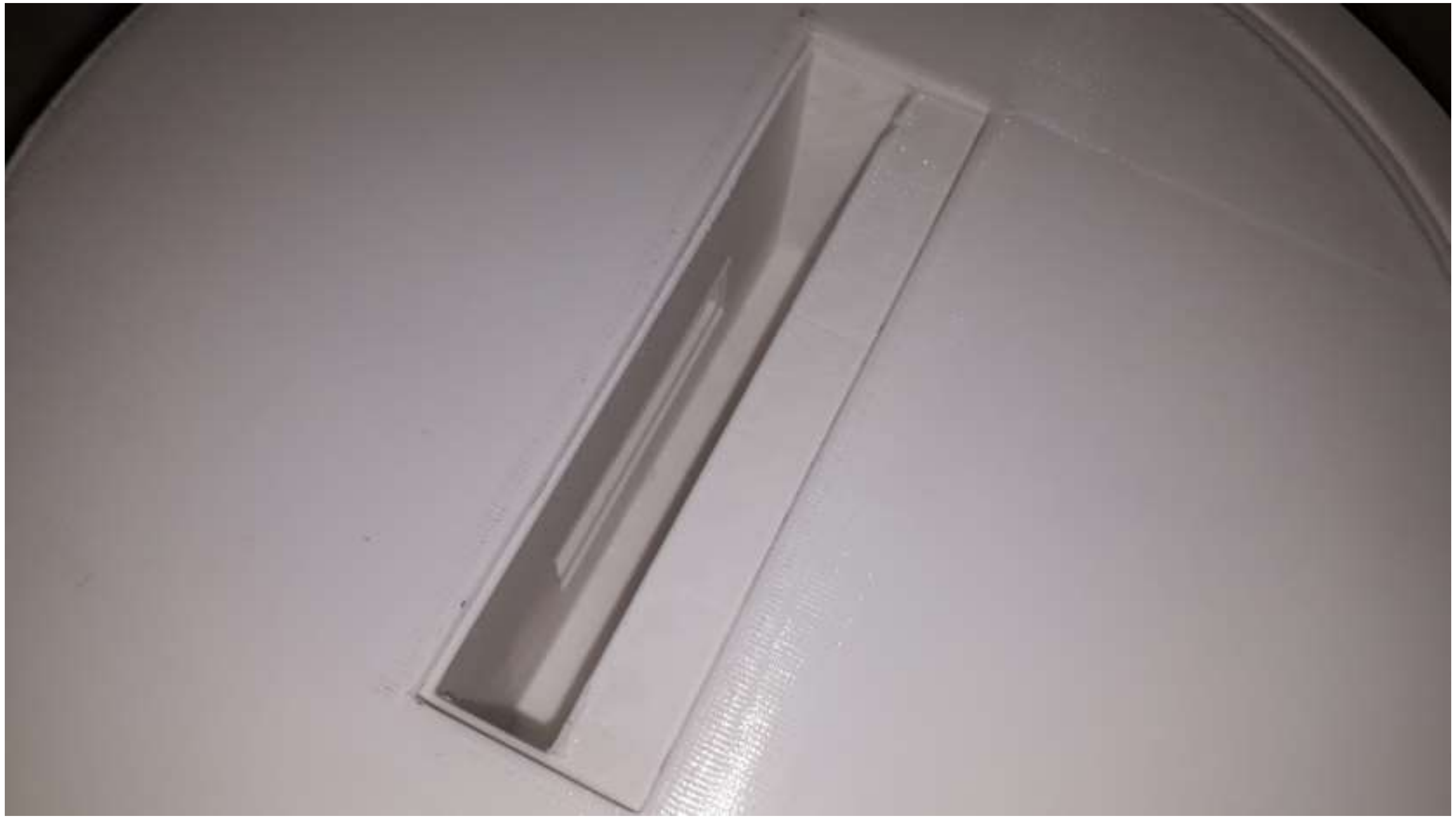


Cleaning mechanism of robot version 2:

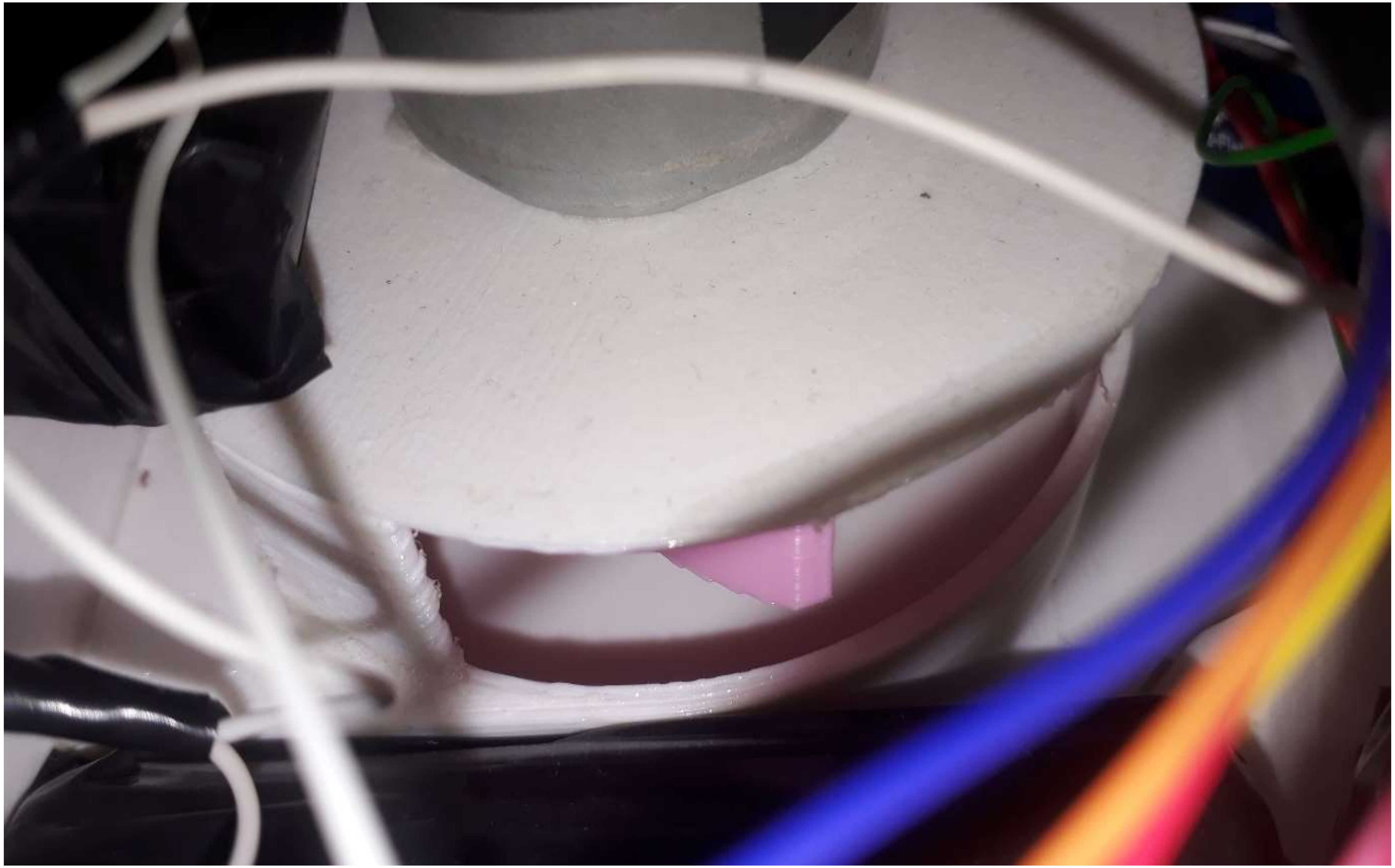
This robot uses a vacuum like mechanism for cleaning. First the dirt gets picked up by the rotating brush.



Then it enters the dust box and gets trapped by the HEPA filter.

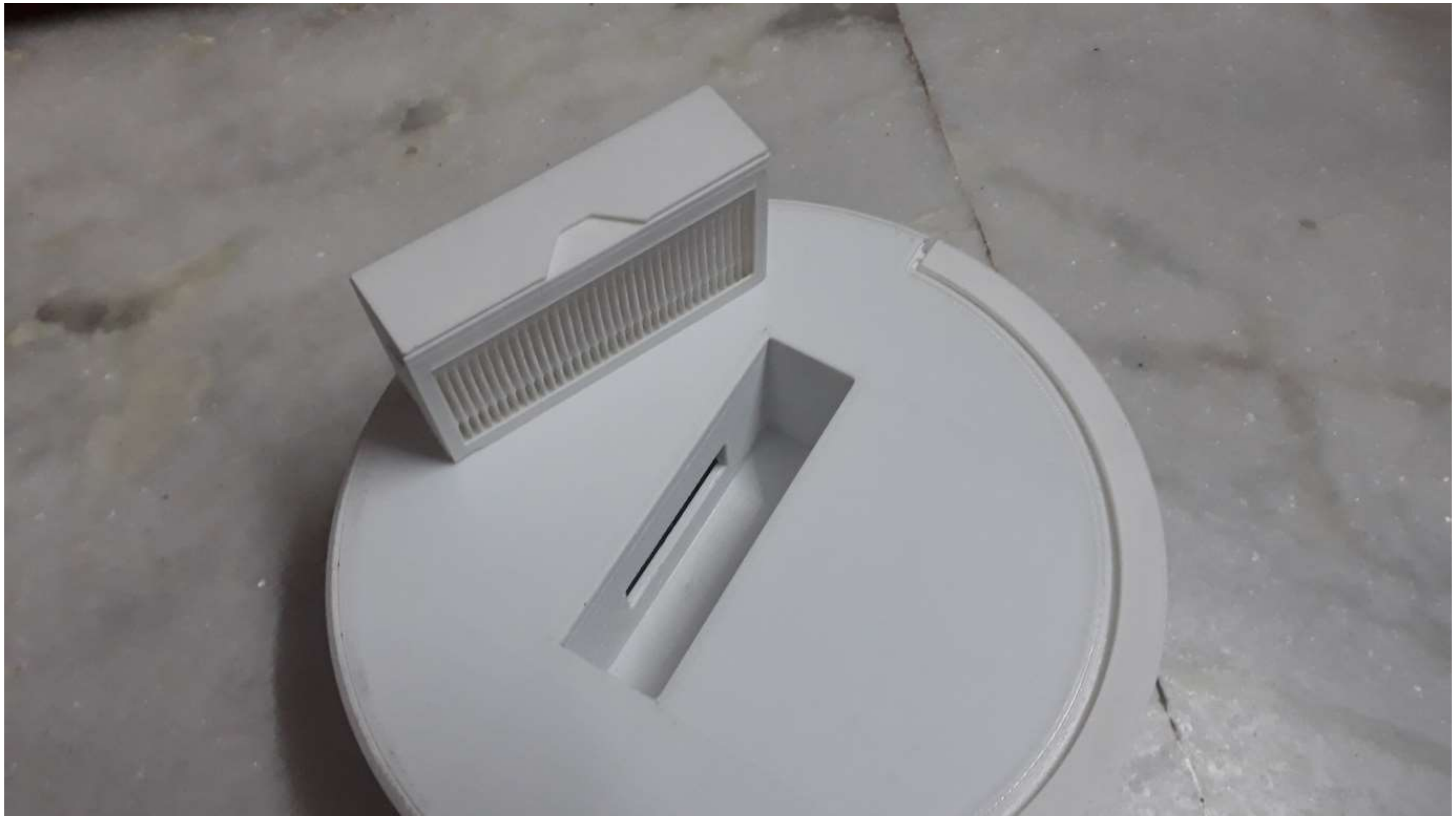


There is an impeller sucking the air through the HEPA filter. So, the air entering the robot is first purified by the HEPA filter.
Picture of impeller:



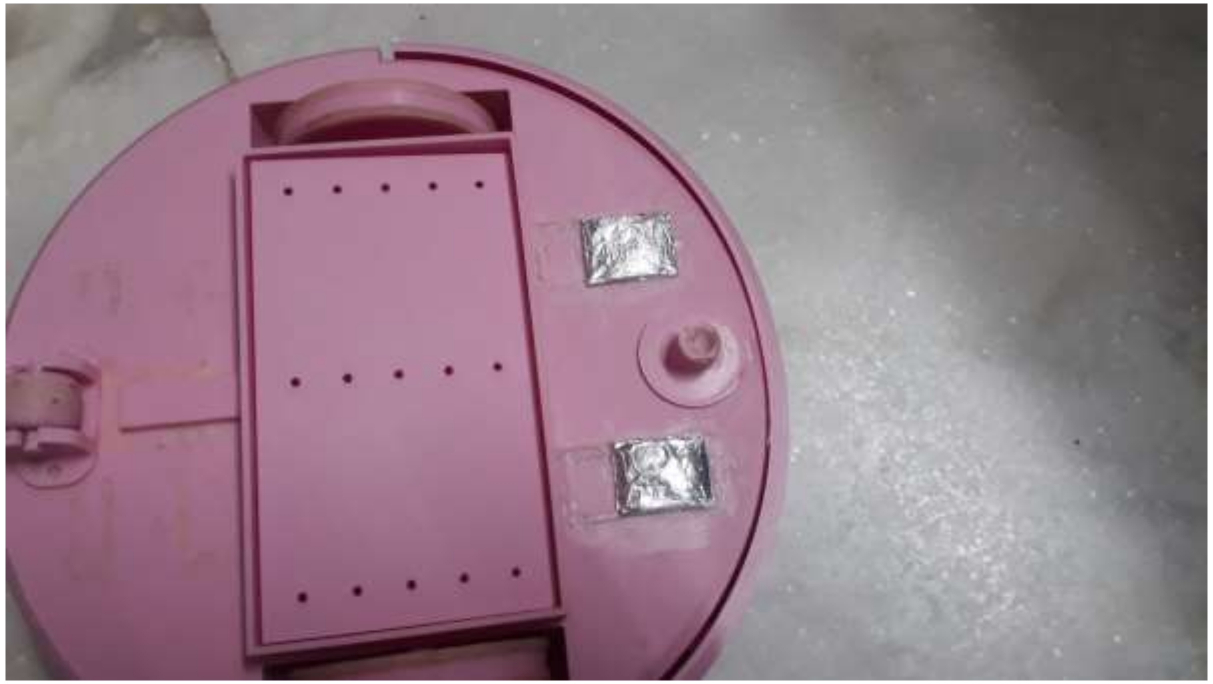
This purified air is used to cool the motors and other components of the robot. After that the air exits the robot through the back vents.

After cleaning the user can remove the dust box.



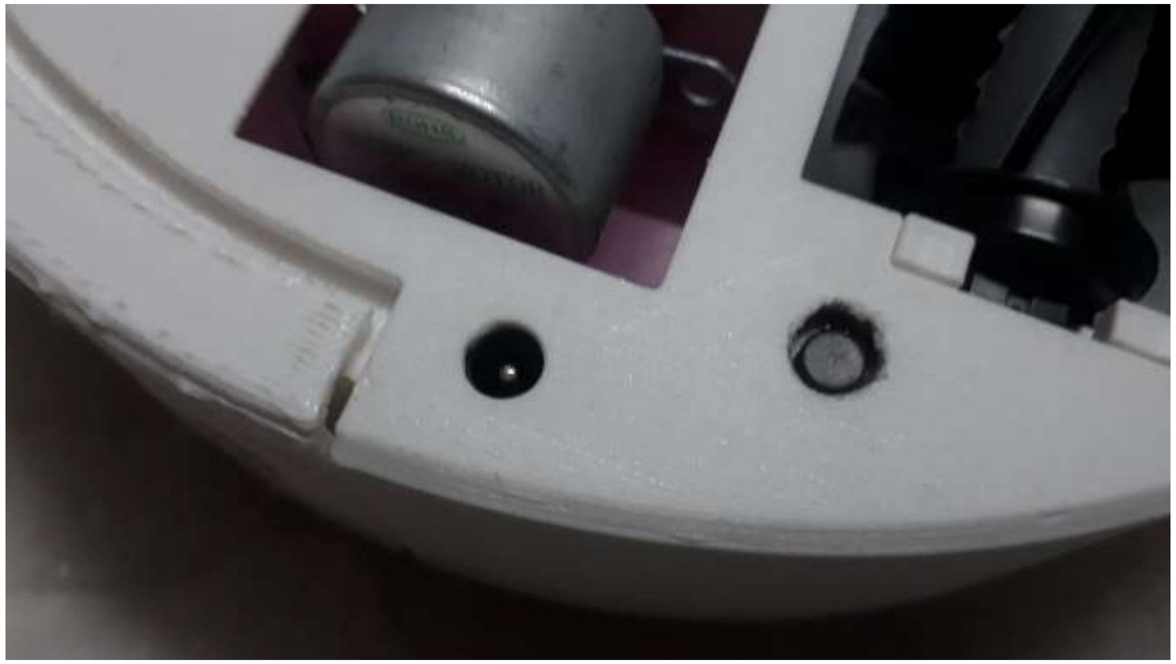
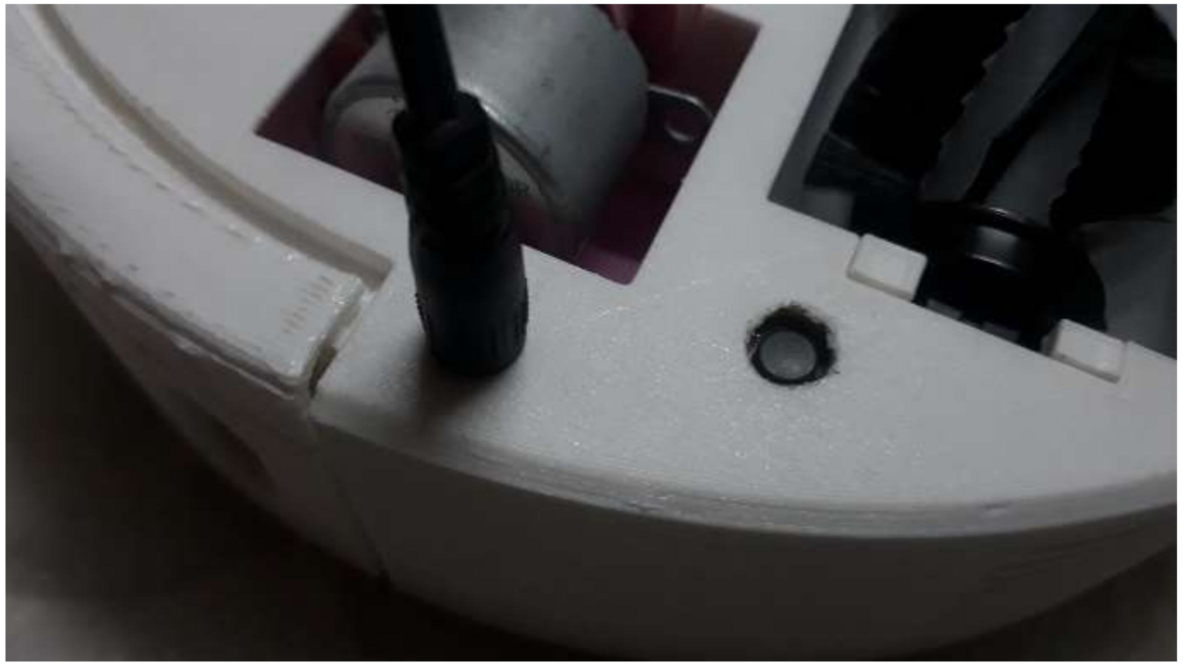
Ways of charging the robots:

Both the robots have charging contacts in the bottom to charge.



They have a charging station where they dock and charge through the charging contacts.

The robot version 2 also has a DC jack in the bottom to charge directly via a 5.5 mm DC jack.

This jack can also output power from the internal battery of the robot to power other things.

Battery capacity

Version 1: 7.4V, 14.8Wh battery

Version 2: 11.1V, 22.2Wh battery

Battery life

Version 1: Up to 3 hours in one charge

Version 2: Not tested

Forward step distance

Version 1: 1.099 cm

Version 2: 0.00153cm

EXPERIMENTAL ANALYSIS

Link to the videos:

https://drive.google.com/drive/folders/1YgutcwrSr0BbnQKVL41cM04cs_JBTSpl?usp=sharing

REFERENCES

- https://www.youtube.com/user/EEVblog
- https://www.youtube.com/channel/UCu7_D0o48KbfhpEohoP7YSQ
- https://lastminuteengineers.com/
- https://www.arduino.cc/
- https://www.youtube.com/
- https://www.youtube.com/user/msadaghd
- https://www.youtube.com/channel/UCu7_D0o48KbfhpEohoP7YSQ
- https://www.youtube.com/channel/UC8Ob-HnnmhlgSv5Vs_i32TQ
- https://www.youtube.com/user/EEVblog
- https://www.youtube.com/channel/UCzml9bXoEM0itbcE96CB03w
- https://www.youtube.com/user/mcwhorpj

Videos

- Videos:

#171 Arduino Guide to Infrared (IR) Communication also for ESP32 and ESP8266:
https://www.youtube.com/watch?v=gADIb1Xw8PE

EEVblog #980 - RoboMaid Automated Vacuum Cleaner Teardown:

- https://www.youtube.com/watch?v=NJvBQoIb5lg

- https://www.youtube.com/watch?v=1pwqS_NUG7Q

- and many more!