



Turtlebot

By:

1. Dhammapada
Mohapatra
2. Abhinand J Pai
3. Anubhav
Srivastava
4. Vodela Amith
Kumar

Mentors:

Prof.Nagarjuna G
Prof.Jude D'Souza
Mr.Ashish Kumar
Pardeshi

Mr.Ravi Sinha

Instructor: Rajib Ranjan
Maiti

Selection of Microcontroller:

- Microcontroller-Soul of Turtlebot (SoC (or) tiny computer).
- Designed to use almost all of its computational power.
- It executes the programs stored in its memory, manages other hardware components of the robot.
- Commonly used ones are **ESP32** and **ATMega328P**.
- In order to select one among these we are going to compare their architectures, computational modules, peripheral modules, power modes and costs of the microcontroller.

1.6 Block Diagram

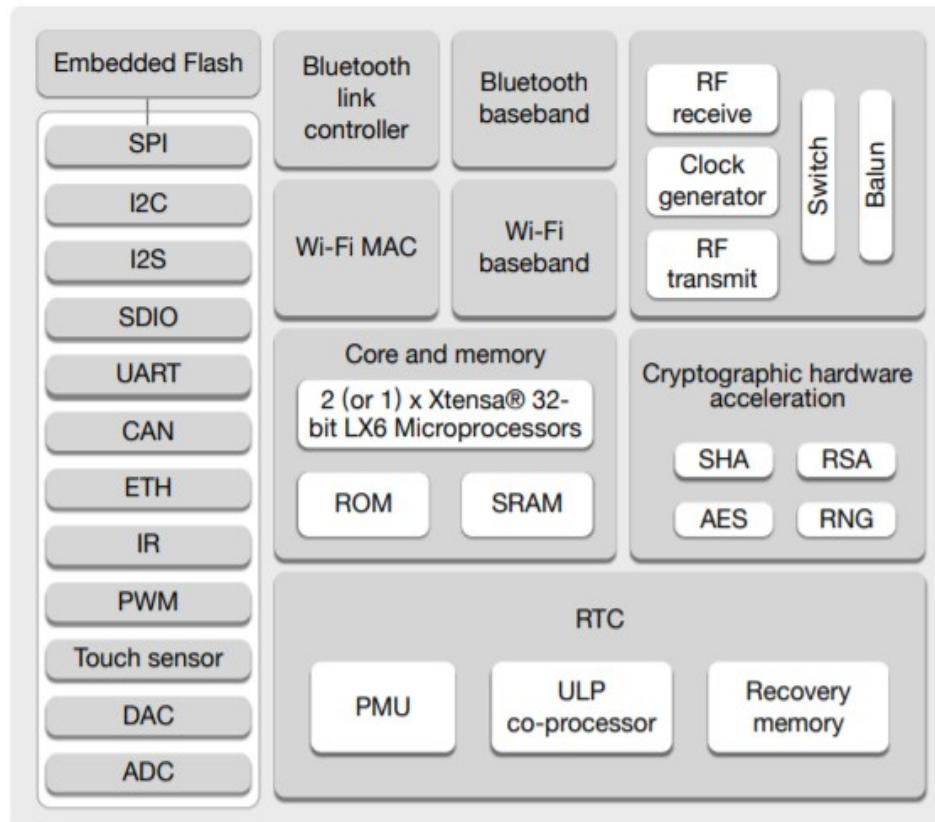
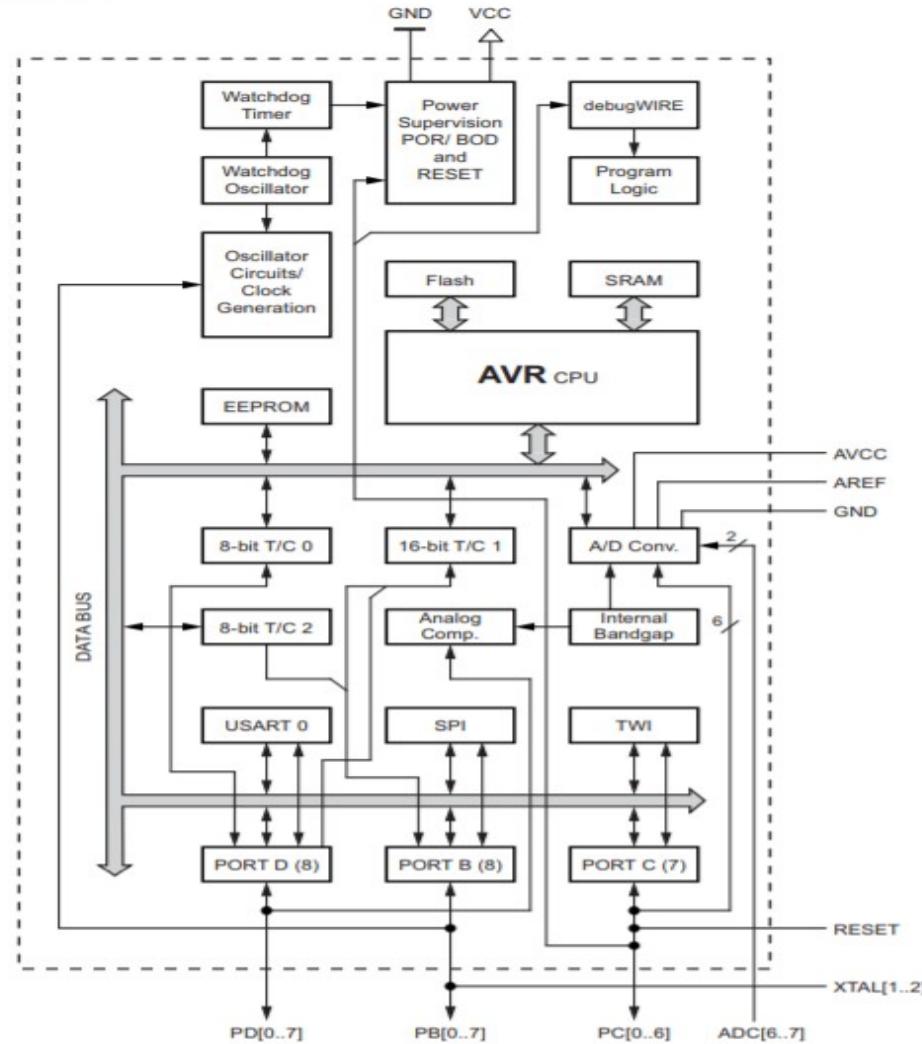


Figure 1: Functional Block Diagram

Figure 2-1. Block Diagram



Computational Features

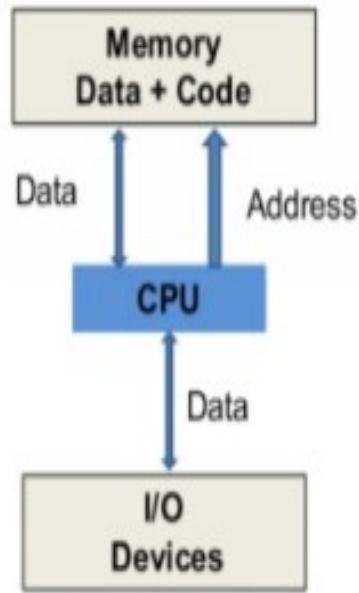
Feature	ESP32(32-bit MCU)	ATMega328P(8-bit MCU)
Architecture(Harvard/Von Neumann)	Harvard	Harvard
Architecture(CISC/RISC)	Enhanced RISC	Enhanced RISC
CPUs	Xtensa LX6(Dual Core)	AVR Core
SRAM(Internal)	520KB	2KB
Flash Memory	4MB	32KB
ROM	448KB	1KB
No of stages in pipelining	7	2
CPU speed	Operating at 160/240 MHz	Operating at 16 MHz
MIPS(Million Instructions Per Second)	Upto 600MIPS	Upto 16MIPS

Memory Architecture:

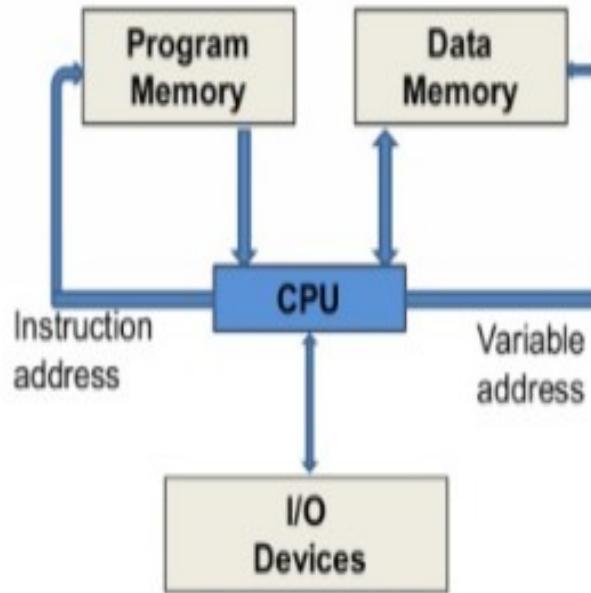
- **Harvard Architecture:** separate memories and buses for program and data.
- **Von Neumann architecture:** which uses same memory to store both instructions and data so the data bus can be used either to fetch instructions or to load data but both can't be done at the same time.

Instruction Set Architecture:

- **RISC Architecture:** RISC stands for Reduced Instruction Set Computer: This is the computer which has small and simple instruction set which enables processor to have fewer clock cycles per instruction thereby increasing the speed.
- **CISC Architecture:** which has complex instructions along with simple instructions which take more than one clock cycle for execution. Modern processors are Enhanced RISC processors which have very few CISC instructions.



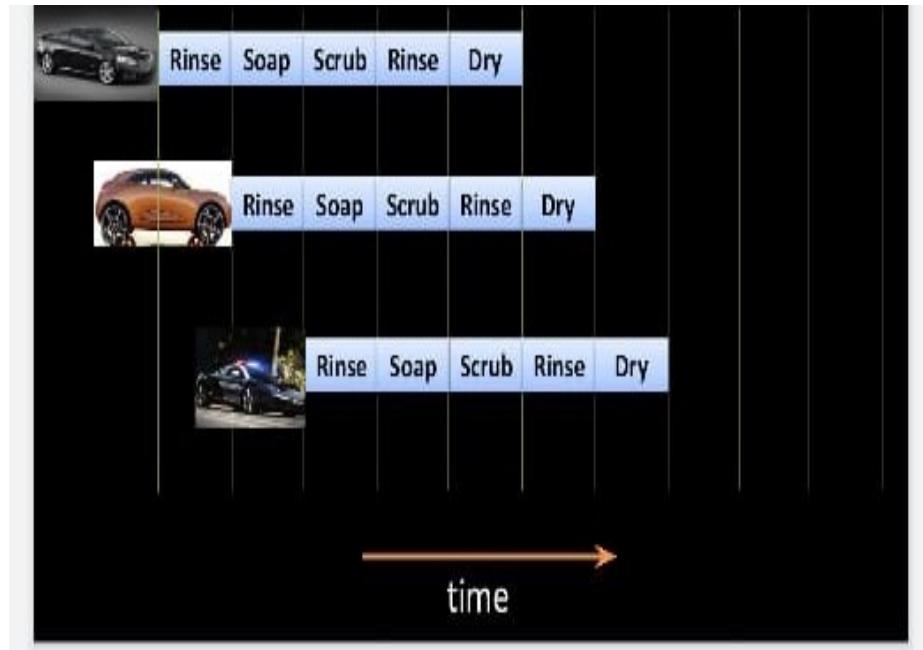
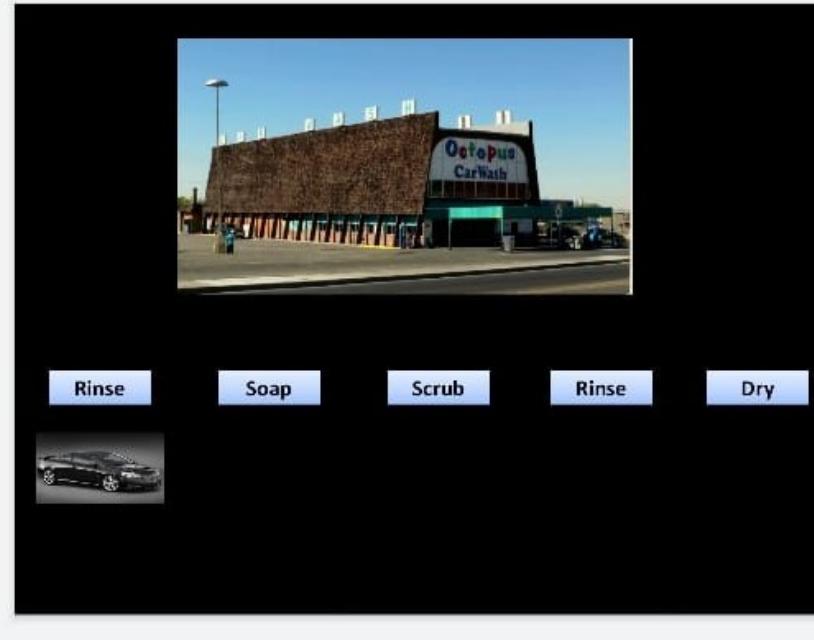
Von Neumann Machine



Harvard Machine

- Harvard and Enhanced RISC architectures are **common features** of both microcontrollers are both are intended to increase the computation speed.
- **ESP32 is Dual core whereas ATMega328P is Single cored:** Having two processors enhances the performance over having a single processor as multiple operations can be done at same time and Dual processors increase **MIPS(million instructions per second)**.
- **ESP32 has 7 stages of pipelining whereas ATMega328P has only 2 stages of pipelining:** **Pipelining** is a technique in which multiple instructions are **overlapped** in execution that is multiple instructions are executed simultaneously. Since ESP32 has more stages of pipelining the clock frequency is more than that of ATMega328P.
- The size of **internal memory** of ESP32 is larger(SRAM, Flash, ROM) than that of ATMega328P.

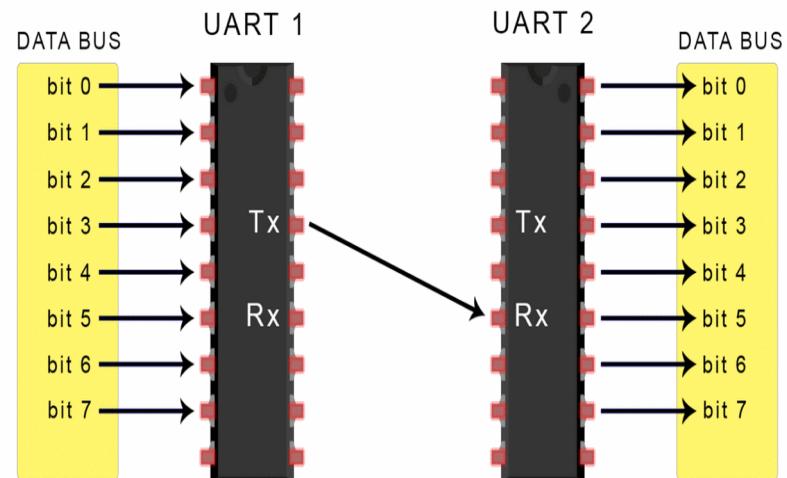
Pipelining Analogy



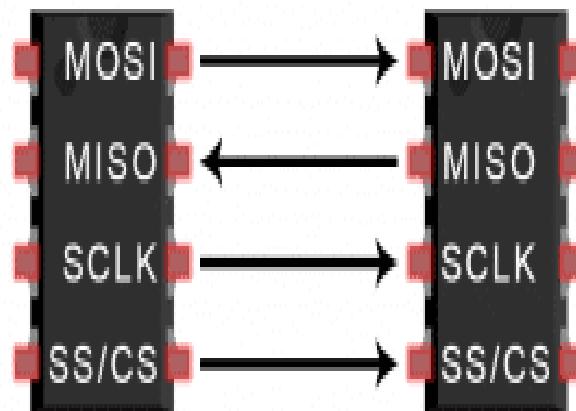
PERIPHERALS

Peripheral	ESP32	ATMega32P
Bluetooth Link Controller	Yes	No
Bluetooth Baseband	Yes	No
WiFi MAC	Yes	No
RF module	Yes	No
WiFi Baseband	Yes	No
SD/SDIO/MMC Host Controller	Yes	No
SDIO/SPI Slave Controller	Yes	No
ULP Co processor	Yes	No
IR Remote Controller	Yes	No
DAC	Yes(2)	No
CAN Controller	Yes	No
Capacitive Touch Sensors	Yes(10 Capacitive Sensing GPIOs)	No
Hall Sensor	Yes	No
GPIO pins	34	25

Peripheral/Feature	ESP32	ATmega328P
RTC Memory and RTC peripherals	Yes	No
I2S Controller	Yes	No
UART	Yes	Yes(USART)
SPI	Yes(4)	Yes
ADC	Yes	Yes
I2C Controller	Yes	Yes
Multiplier	32 bit multiplier	2 cycle Multiplier(8-bit)
Does it support Floating point unit?	Yes	No
Does it support DSP instructions?	Yes	No
Cryptographic Hardware Accelerators	Yes	No
LEDPWM	Yes	No
MCPWM	Yes	Yes



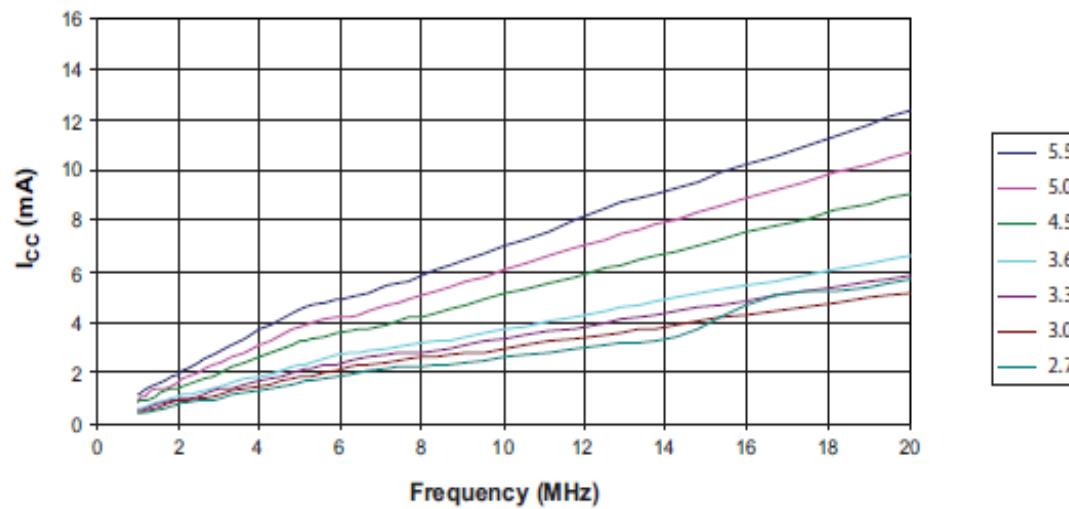
Master Slave



Mode	Range of current
Operating voltage	1.8V-3.6V(3.3V -Typical voltage)
Active mode	95mA-240mA(depending on mode of operation)
Modem sleep mode	20mA-68mA(depending on frequency and number of cores)
Light sleep mode	0.8mA
Deep sleep mode	10uA/100uA/150uA
Hibernation mode	5uA
Power off mode	1uA

Mode	Range of current
Operating Voltage	2.7V-5.5V
Active Mode	1.5mA(3V,4Mhz)=6mA(linear Interpolation to 16Mhz(operating frequency) as typical characteristics are linear)
Power down mode	1uA at 3V

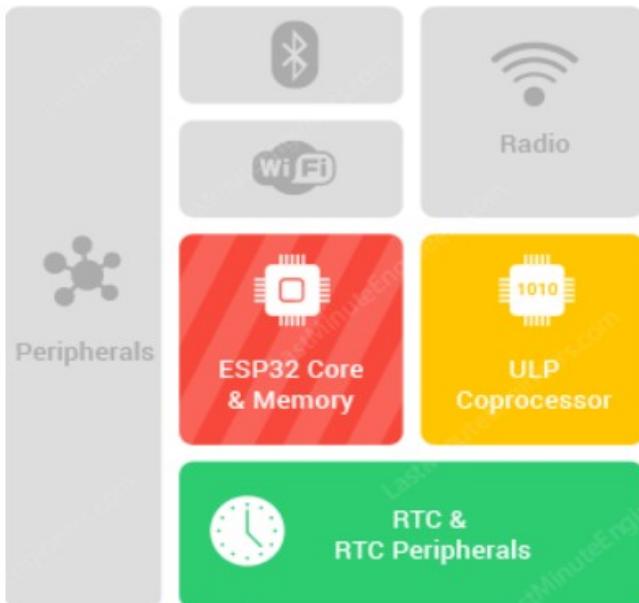
Figure 29-1. Active Supply Current versus Frequency



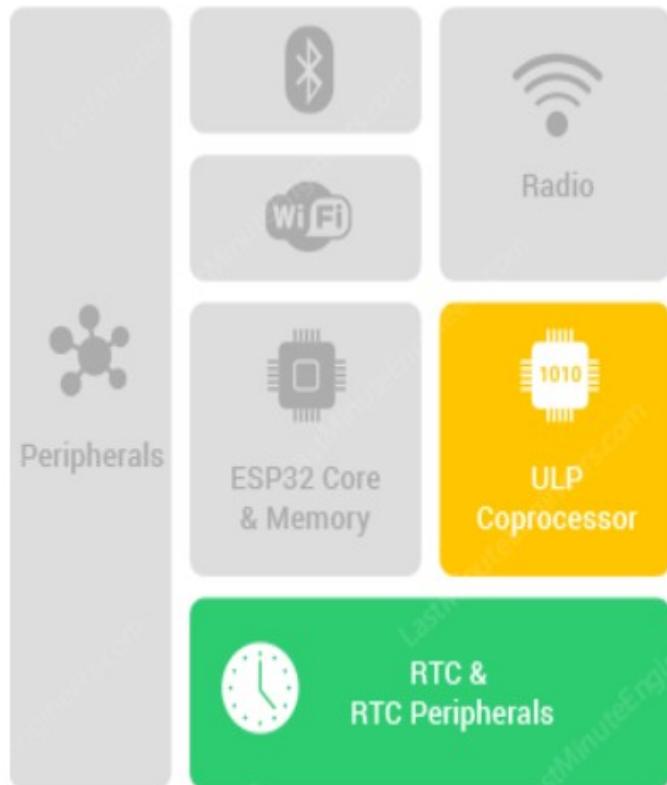
Active Mode



- **Modem Sleep Mode** : Communication modules are disabled
- **Light Sleep mode** is similar to modem sleep mode. In this during sleep time CPU and peripherals are paused by **Clock gating** so power consumption is reduced.(function should be used).

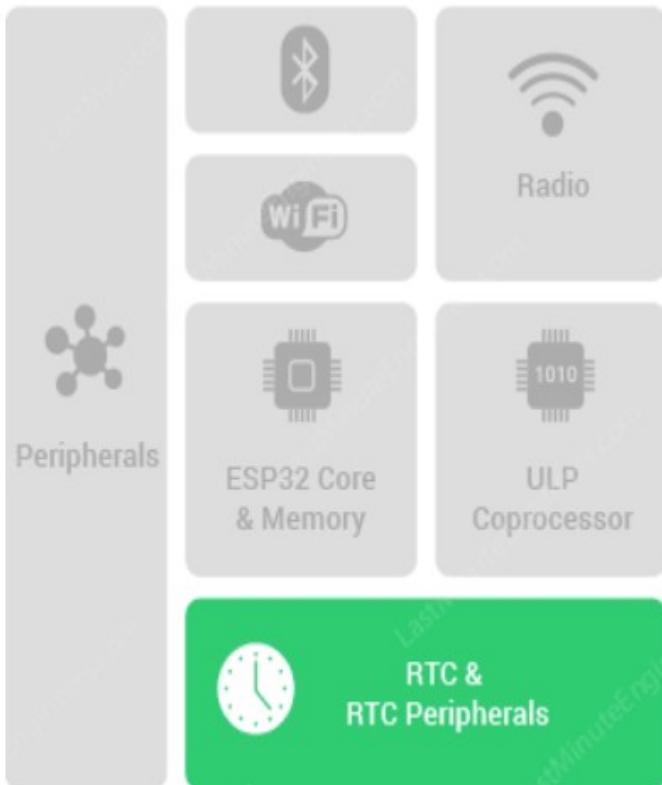


Deep Sleep mode:



- We need to use function to push into deep sleep mode.
- In this mode power supply is cut-off to Core(CPU and Memory) and peripherals which were clock gated.
- ULP co processor, RTC(real time clock) Memory and RTC Peripherals are active.
- The code required to be executed by processor & Data recorded during this mode should be stored in RTC memory.

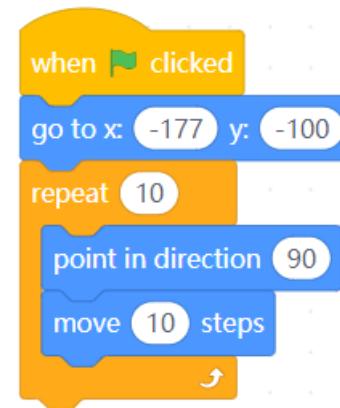
Hibernation

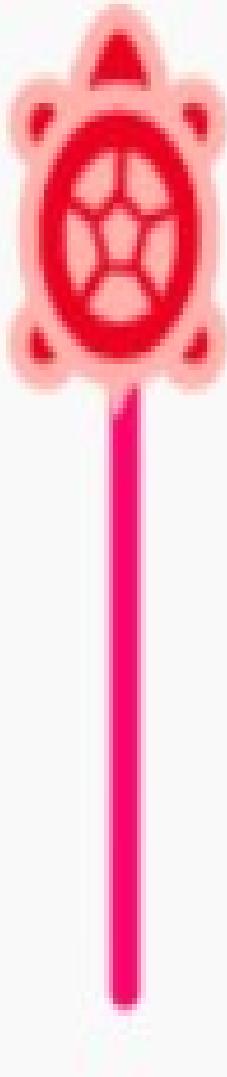


- **Hibernation mode:** In this mode power supply to ULP Coprocessor and RTC memory is cut-off.
- **Some peripherals and RTC timer** are the only active modules whose function is to wake up the ESP32 when required this reduces even more power consumption.

Block programming

```
@event.greenflag
def on_greenflag():
    sprite.x = -177
    sprite.y = -100
    for count in range(10):
        sprite.direction = 90
        sprite.forward(10)
```



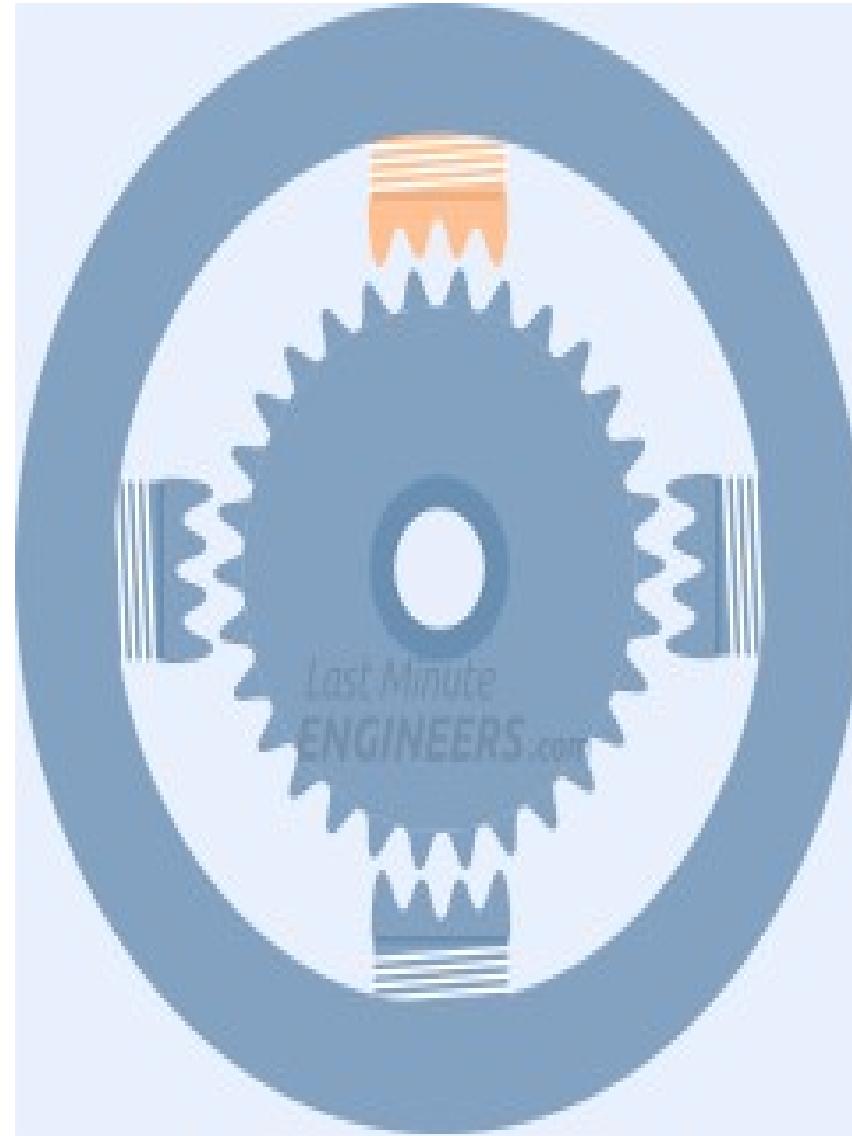


TURTLEBOT – ELECTROMECHANICA L COMPONENTS AND COMPLETE CODING USING ARDUINO IDE THROUGH BLUETOOTH

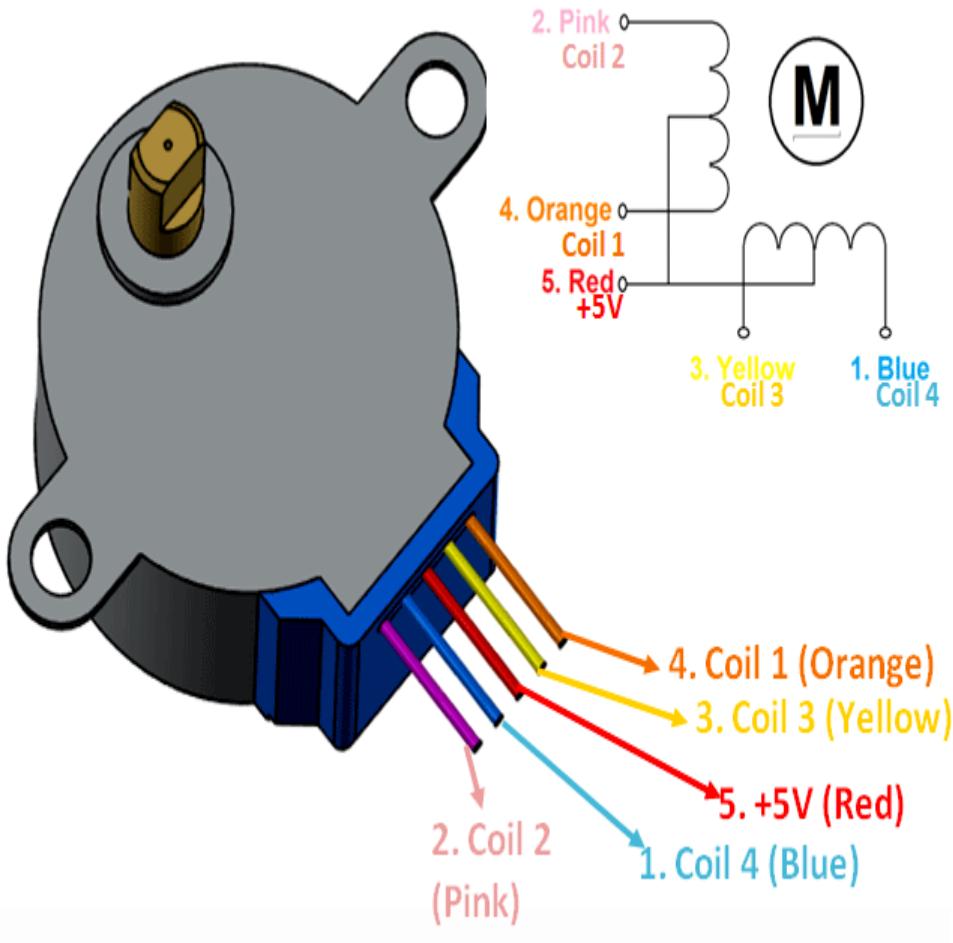
BY ANUBHAV

HOW A STEPPER MOTOR WORKS?

- Each HIGH pulse sent, energizes the coil, attracts the nearest teeth of the cogged wheel and drives the motor one step.
- The sequence of pulses determines the spinning direction of the motor.
- The frequency of the pulses determines the speed of the motor.
- The number of pulses determines how far the motor will turn.

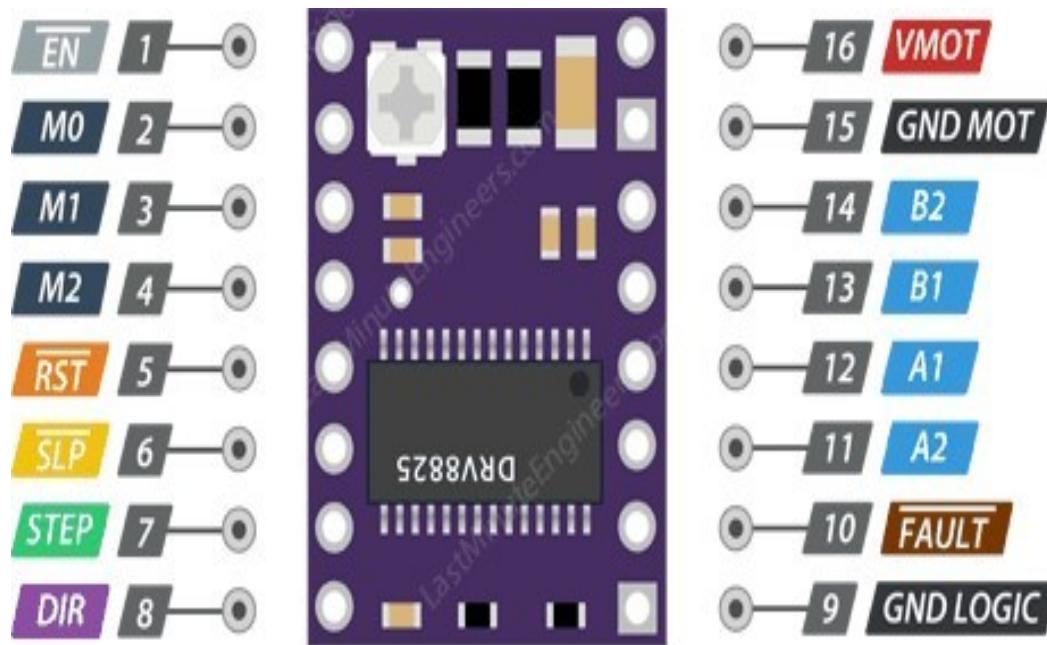


28BYJ-48 STEPPER MOTOR



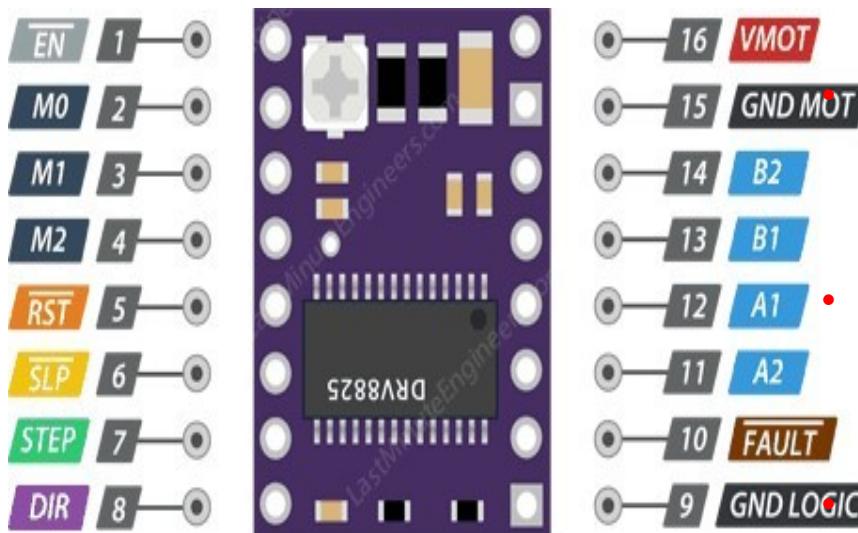
- 4 coil unipolar arrangement.
- Used in CNC machines, precise control movements, security cameras etc.
- Power consumption of motor is around 240mA.
- 2048 steps/rev.
- Max speed: 500 steps/s i.e. around 4s. To complete one rev.

DRV8825 Motor Driver



The DRV8825 stepper motor driver has output drive capacity of up to 45V and lets you control one stepper motor at up to 2.2A output current per coil.

PINOUT DETAILS



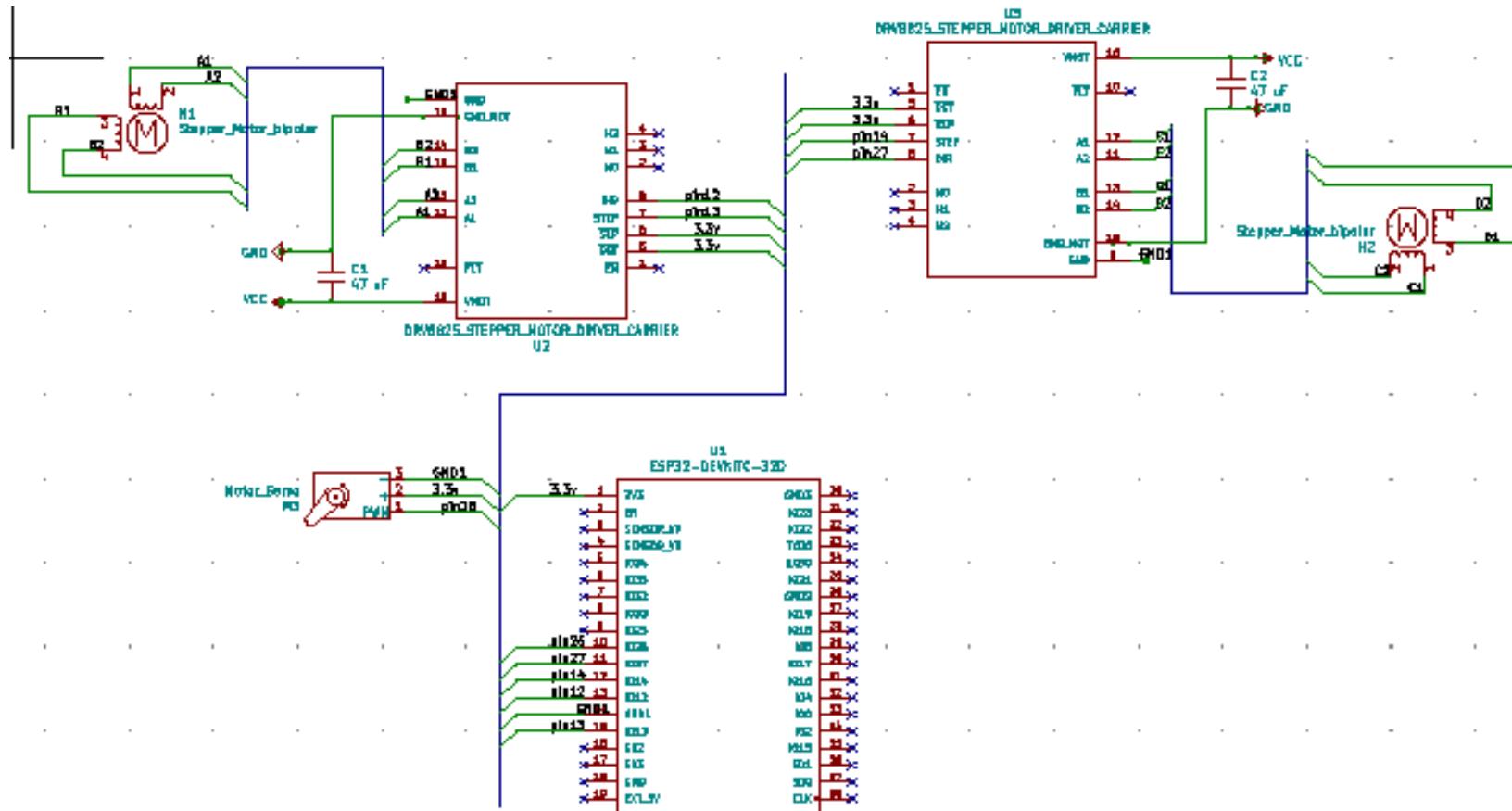
- **B2, B1, A1 & A2** are driver output pins. These are connected to the stepper motor.

- **VMOT & GND MOT** supplies power for the motor which can be 8.2V to 45V.
- Microcontroller's ground should be common with **GND LOGIC** pin.
- **M0, M1 & M2** are step size(resolution) selector inputs. By setting appropriate logic levels to these pins we can set the motors to one of the six step resolutions.
- **STEP** input controls the mirosteps of the motor. Each HIGH pulse sent to this pin steps the motor by number of microsteps set by Microstep Selection Pins.
- **DIR** input controls the spinning direction of the motor. Pulling it HIGH drives the motor clockwise and pulling it LOW drives the motor counterclockwise.
- **EN** Pin is active low input, when pulled LOW(logic 0) the DRV8825 driver is enabled.
- **SLP** Pin is active low input. Meaning, pulling this pin LOW puts the driver in sleep mode, minimizing the power consumption.
- **RST** is also an active low input. When pulled LOW, all STEP inputs are ignored, until you pull it HIGH.
- **FAULT** goes LOW whenever the H-bridge FETs are disabled as the result of over-current protection or thermal shutdown.

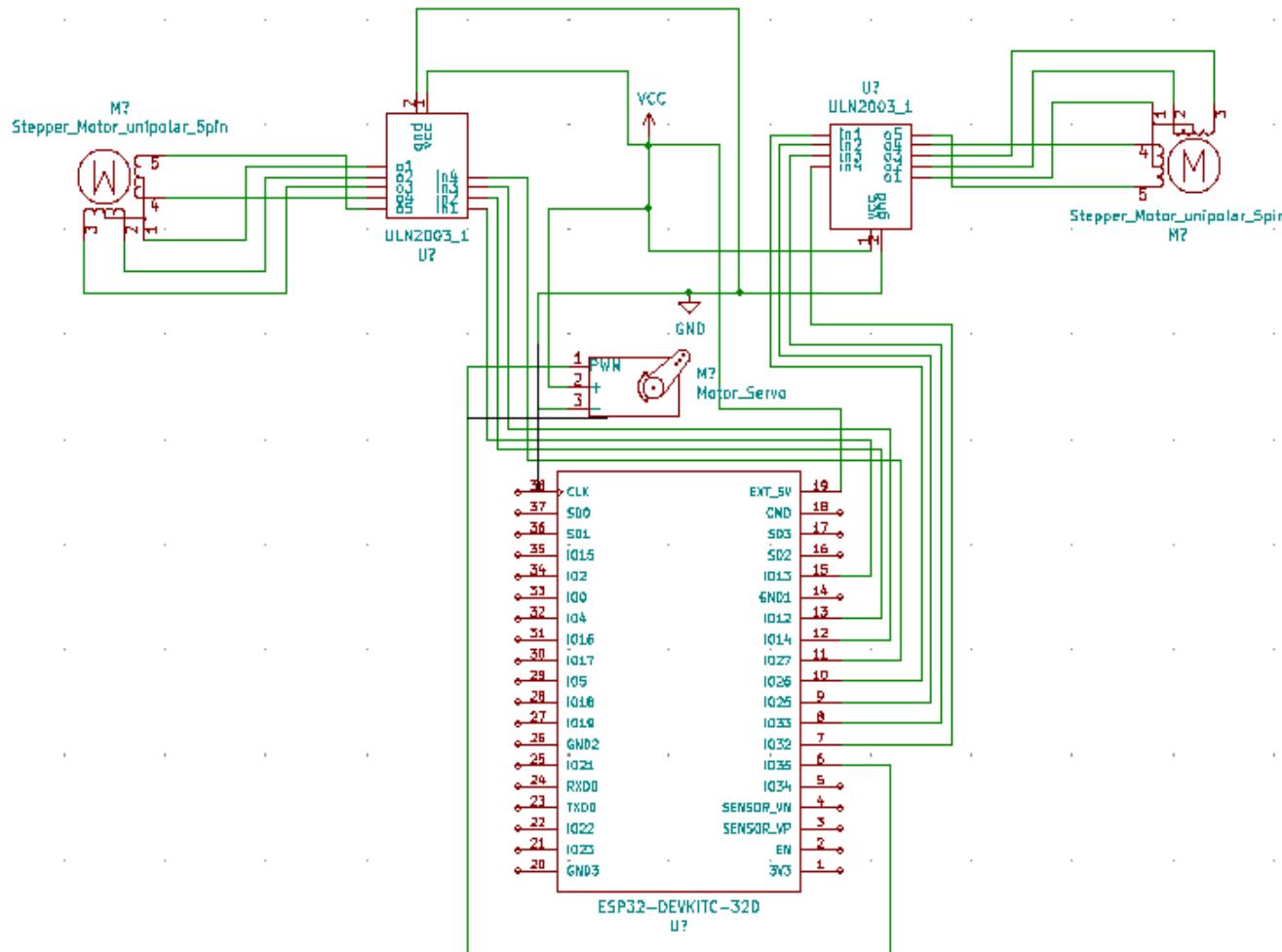
OVERVIEW

- DRAWING THE SCHEMATIC DIAGRAM USING KICAD.
- MAKING THE CONNECTIONS.
- WIRELESS CONTROL
 - CONFIGURING BLUETOOTH ON ESP32
 - SETTING UP STATION MODE USING WiFi ON ESP32.
- BATTERY MANAGEMENT SYSTEM.
- TESTING THE CODE FOR ACCURACY.
- LINKS

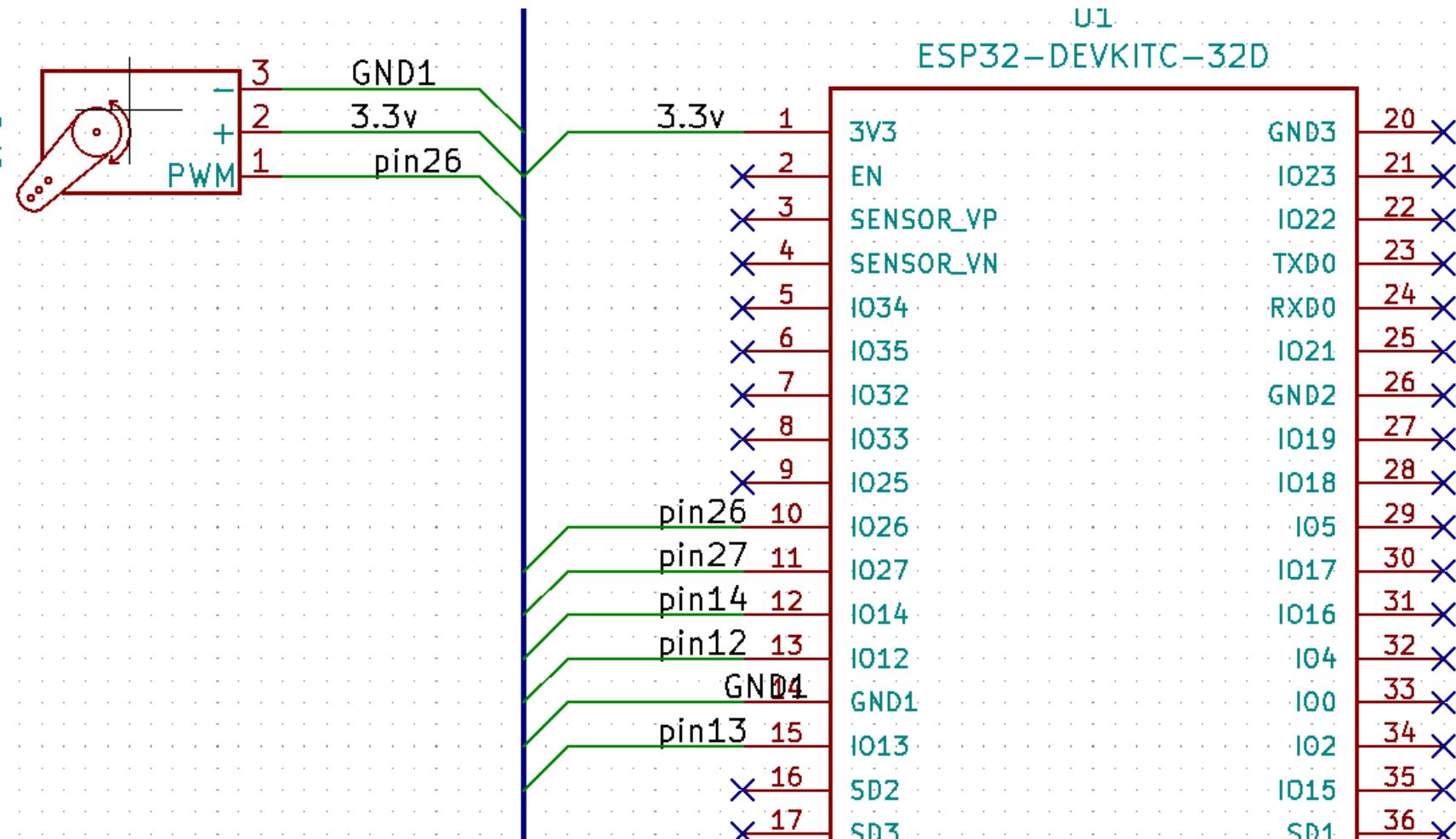
DRAWING SCHEMATIC DIAGRAM USING KICAD



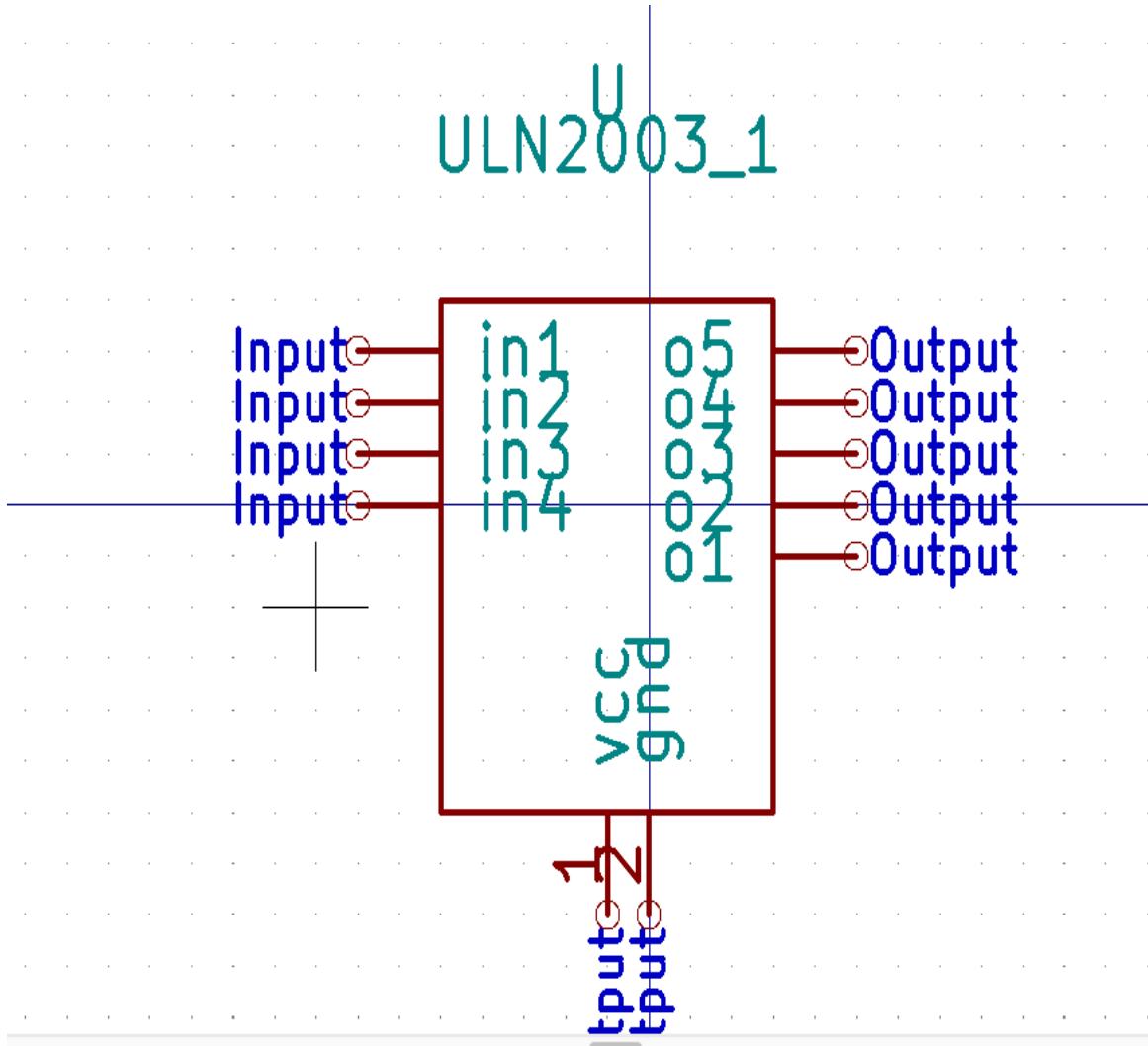
Making the schematic more presentable



Using buses and net names

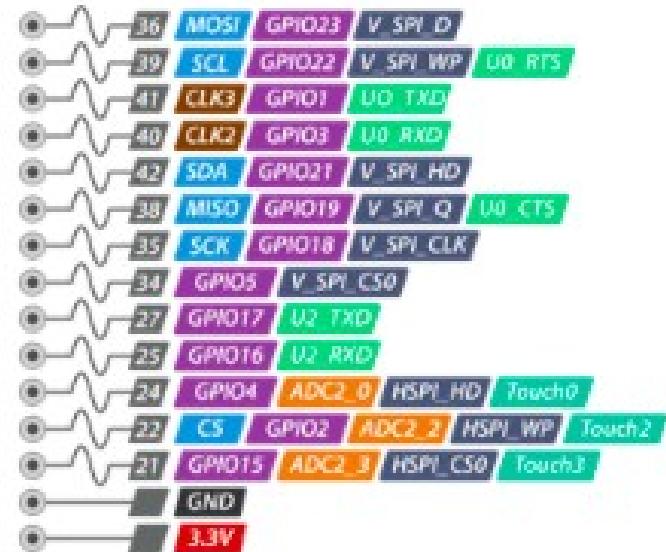
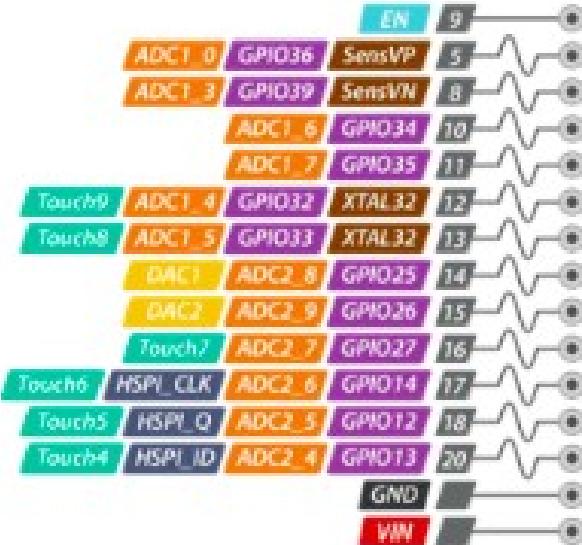


Editing The Symbol Library



MAKING THE CONNECTIONS

ESP32



Power



Control



Arduino



ADC



SPI



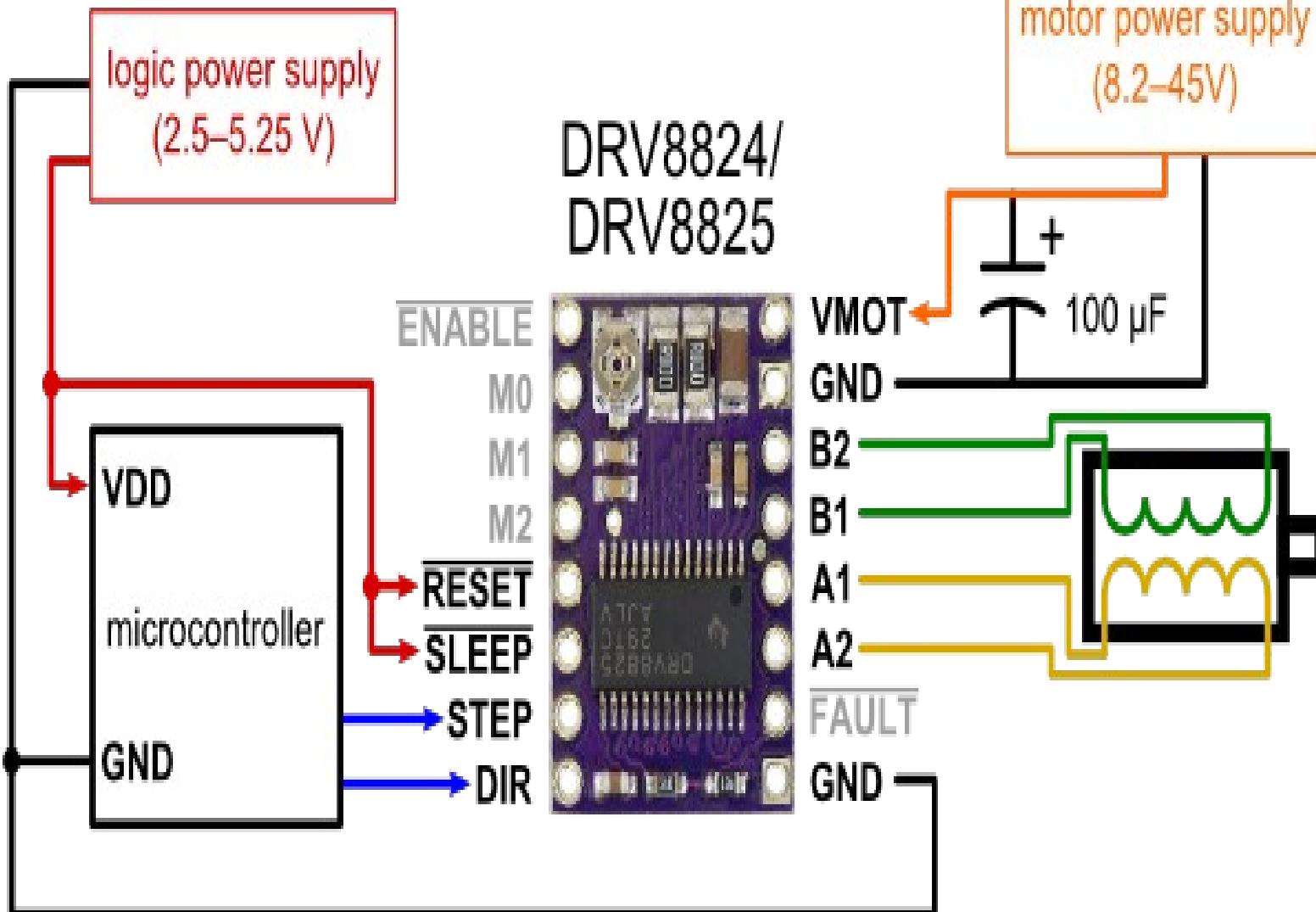
UART



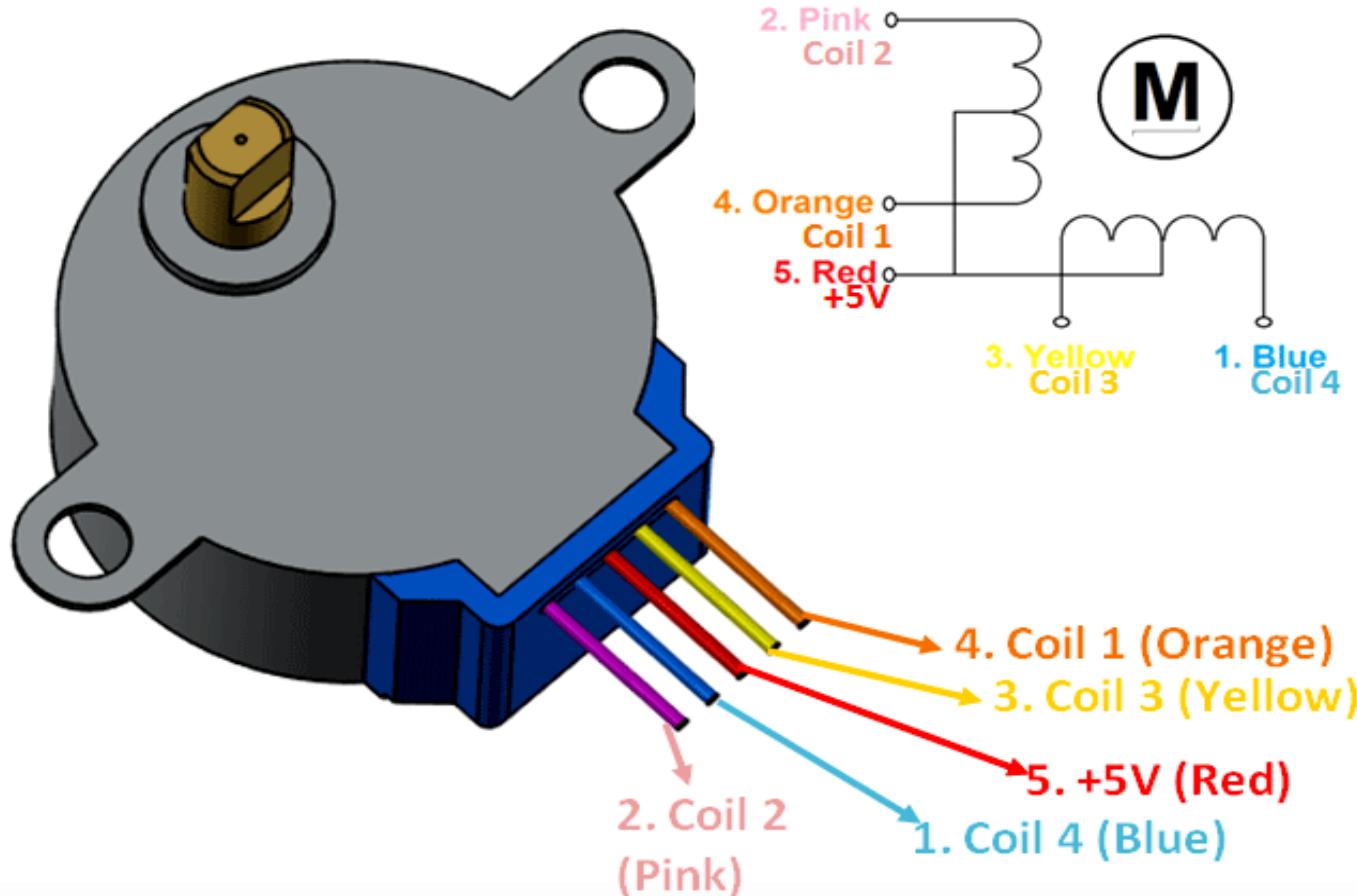
EN



PWM

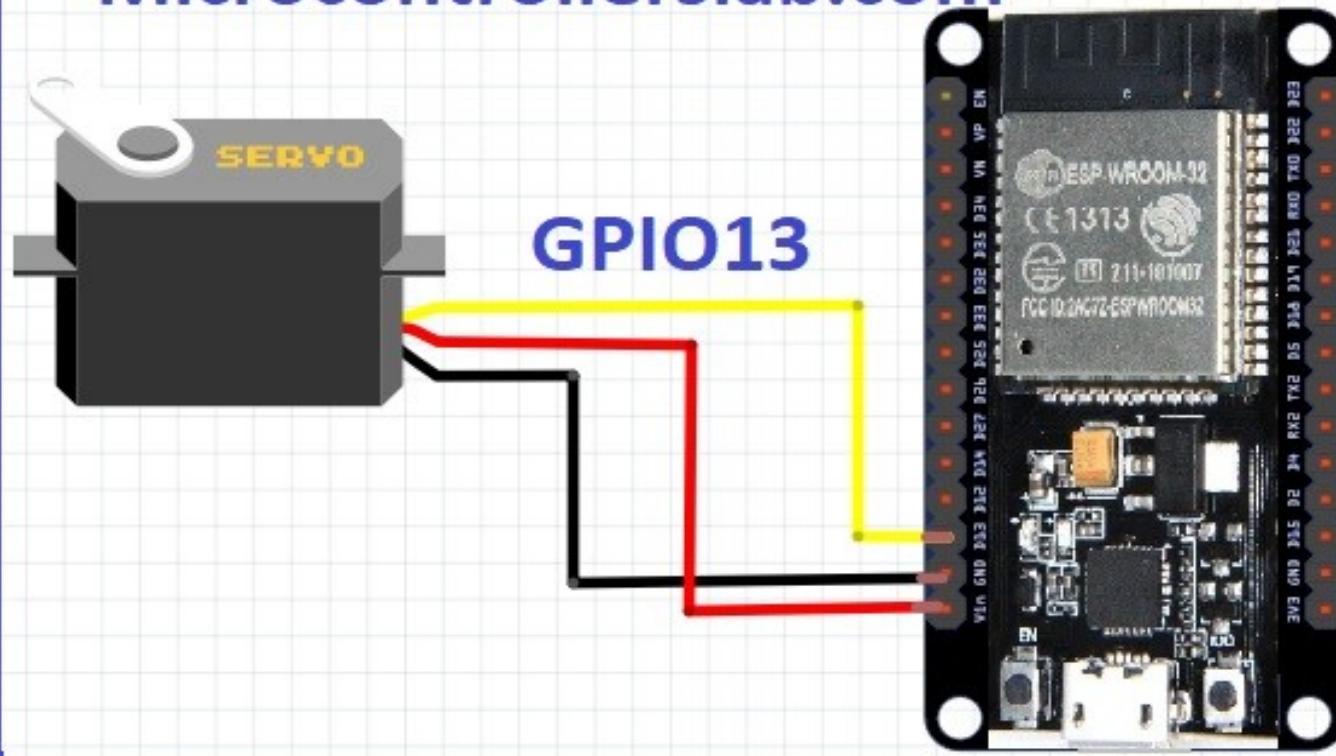


28BYJ-48 5V STEPPER MOTOR



SERVO MOTOR SG90

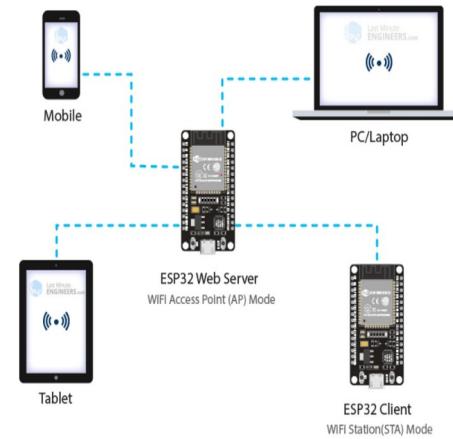
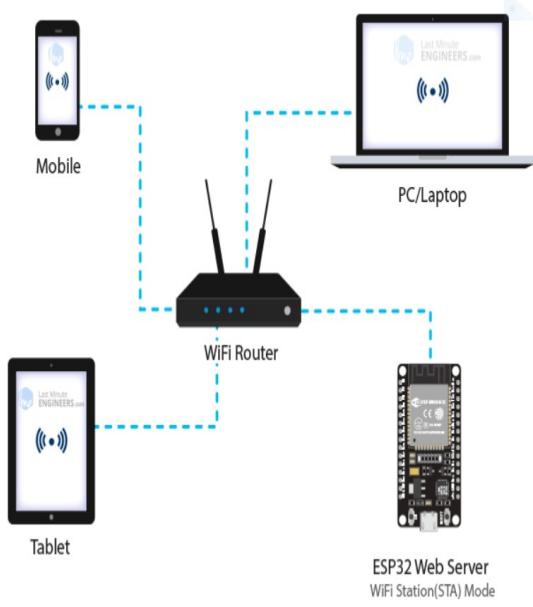
Microcontrollerslab.com



- **Mblock** : Python based open source software in which ESP32 device libraries are installed.
- We were able to connect the robot to this software but the **code wasn't being uploaded on it(Due to fermata problem)**.
- **It has a pen extension but it was only able to work with the sprite. Pen extension cannot control servo.**
- **Upload mode:** The program to be executed is compiled and uploaded onto robot and robot executes the code without any connection to the laptop.
- **Live mode:** interface sends commands one by one to the robot and robot executes each command as it arrives. Robot remains connected to the laptop and no compilation and uploading step.
- **Advantage of Live Mode:** As there is no compilation or uploading step, we can create simple commands, see them execute and solve the problems if any occurring while the command is executed.

- **Upload mode** is better when quick response from the robot is needed as the code runs fast, **latency will be very less compared to live mode** so that reaction is quick.
- **KB IDE**: C++ based, open source by which we were able to interface the robot.
 - No option for creating new blocks.
- **Pictoblox**: Majorly based on Javascript through which we were able to interface the robot.
- Not open source so can't edit the source code to create blocks but can create new blocks from existing blocks.
- **Snap4Arduino**: Javascript based open source software but it was not compatible with ESP32 board.

- ESP32 is configured as **Asynchronous Webserver** as there is no need for periodically executing the handle-client function separately.(need not be active can automatically go to **one of the sleep modes** for power saving).
- **GET** requests are being used to send data to server from client.(less data no need of security for that data).
- **Station Mode:** ESP32 connects to existing WiFi network, gets IP address from the server which we display on serial monitor and this can be used for the webpages to be delivered by MCU.
- **SoftAP Mode:** Microcontroller creates own WiFi network with its own SSID and IP address . SoftAP as it is not wired.
- **Advantage:** Can operate irrespective of credentials of WiFi local network.



WiFi Code

```
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include<FS.h>
#include<Servo.h>

AsyncWebServer server(80);

String input="";
String action="";

const char* ssid = "D-Link";
const char* password = "9447069616";
const char* UNITS = "units";
const char* MOTION = "motion";

int steppin_L = 13;
int dirpin_L = 12;
int steppin_R = 14;
int dirpin_R = 27;
int steps_rev = 2048;
int dist = 13;

int wheel_dia=2.8;
int idata;

int servoPin = 15;
Servo servo;
```

```
const char units_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>
<title>ESP Input</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
</head><body>
<form action="/get">
  Movement:<input type="text" name="motion">
  Number of units:<input type="number" name="units">
  <input type="submit" value="Submit">
</form>
</body></html>)rawliteral";

void setup() {
  servo.attach(servоСon);
  pinMode(steppin_L, OUTPUT);
  pinMode(dirpin_L, OUTPUT);
  pinMode(steppin_R, OUTPUT);
  pinMode(dirpin_R, OUTPUT);

  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("WiFi Failed!");
    return;
  }
  Serial.println();
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
```

```
server.onNotFound([]AsyncWebRequest *request) {
    request->send(404, "text/plain", "Not found");
});

server.begin();

server.on("/", HTTP_GET, [] (AsyncWebRequest *request) {
    request->send(200, "text/html", units_html);
});

server.on("/get", HTTP_GET, [] (AsyncWebRequest *request) {
    request->send(200, "text/html", "Wait for the command to be executed");
    if (request->hasParam(UNITS)) {
        input = request->getParam(UNITS)->value();
        idata = input.toInt();
    }
    if (request->hasParam(MOTION)) {
        action = request->getParam(MOTION)->value();
    }
    Serial.println("The command that will be executed is:"+action+input);
}
```

```
if(action=="forward") {
    servo.write(0);
    forward(idata);
    action="";
    servo.write(20);
}
else if(action=="backward") {
    servo.write(0);
    backward(idata);
    action="";
    servo.write(20);
}
else if(action=="right") {
    servo.write(0);
    right(idata);
    action="";
    servo.write(20);
}
else if(action=="left") {
    servo.write(0);
    left(idata);
    action="";
    servo.write(20);
}
}
});
```

COMPLETE CODING ON ESP32 USING ARDUINO IDE – TURNING THE BOT AT ANY ANGLE, EQUATION OF STRAIGHT LINE TO DRAWING GRAPH AND DRAWING CIRCLE

```
#include "BluetoothSerial.h"
#include <AccelStepper.h>
#include <MultiStepper.h>
#include <Servo.h>
#include "math.h"

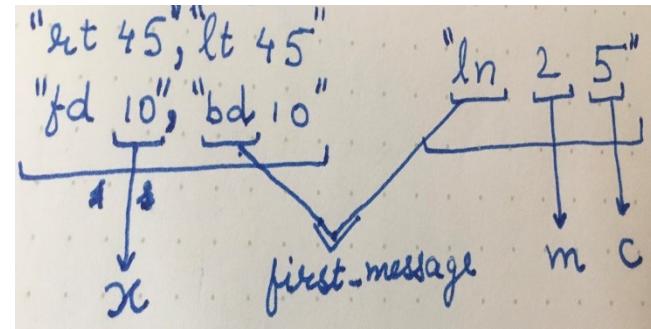
// Check if Bluetooth configs are enabled
#ifndef CONFIG_BT_ENABLED || !defined(CONFIG_BLUEDROID_ENABLED)
    #error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

String message;
int servoPin = 15; int f=1,b=1,r=1,l=1,ln=1,c=1;
AccelStepper motorL(1, steppin_L, dirpin_L);AccelStepper motorR(1, steppin_R, dirpin_R);
Multistepers steppers;
Servo servo; BluetoothSerial SerialBT; //Servo object, and Bluetooth object
int steppin_L = 13;
int dirpin_L = 12;
int steppin_R = 14;
int dirpin_R = 27;
int steps_rev = 2048;
int dist = 13; //Distance between 2 wheels of turtlebot
float wheel_dia=2.8;
```

```

void setup() {
    servo.attach(servoPin);
    pinMode(steppin_L, OUTPUT);
    pinMode(dirpin_L, OUTPUT);
    pinMode(steppin_R, OUTPUT);
    pinMode(dirpin_R, OUTPUT);
    Serial.begin(115200);
    SerialBT.begin("ESP32"); // Bluetooth device name
    Serial.println("The device started, now you can pair it with
bluetooth!");
}

```



```

void loop() {
    // Read received messages
    if (SerialBT.available()){
        String instr = SerialBT.readString();
        int single_instr=instr.indexOf(',');
        int last_instr=instr.indexOf('\n');
        if(single_instr!=-1 &&
(f+b+r+l+ln+c)/6==1)  {

            message=instr.substring(0,single_instr);
            instr.remove(0,single_instr+1);
        }
        else  {
            if(last_instr!=-1 &&
(f+b+r+l+ln+c)/6==1)
                message=instr;    instr="";
        }
    }
}

```

- 1

- 2

```

String first_message=message.substring(0,2);
String second_message=message.substring(3);
int index_of_space = second_message.indexOf(" ");
if(index_of_space == -1){
    double x = second_message.toDouble(); // x is deg
} else{

    String third_message =
second_message.substring(0,index_of_space); //slope
    String fourth_message =
second_message.substring(index_of_space+1); //inter.
    double m = third_message.toDouble(); // m is slo
}

```

```
if (first_message=="fd"){
    servo.write(90); //pen down
    forward(x);
    servo.write(0); //pen up
    message = "";
    SerialBT.print("forward command
executed");
}else if (first_message=="rt"){
    servo.write(90); //pen down
    right(x);
    servo.write(0); //pen up
    message = "";
    SerialBT.print("right command
executed");
}else if(first_message=="lt"){
    servo.write(90); //pen down
    left(x);
    servo.write(0); //pen up
    message = "";
    SerialBT.print("left command executed");
} else if(first_message=="bd"){
    servo.write(90); //pen down
    backward(x);
    servo.write(0); //pen up
```

- 3

```
} else if(first_message=="ln"){
    line(m,c); message = "";
    SerialBT.print("line command
executed");
} else if(first_message=="cl")
{
    circle(x); message = "";
    SerialBT.print("circle
command executed");
}}
```

All Functions

```
void left(double degrees){  
    double rotation = degrees / 360;  
    double distance = dist * 3.14 * rotation;  
    int steps = step(distance);  
    digitalWrite(dirpin_L, HIGH);  
    digitalWrite(dirpin_R, HIGH);  
    for (int i = 0; i < steps; i++)  
    {digitalWrite(steppin_L, LOW);  
    digitalWrite(steppin_R, LOW);  
    digitalWrite(steppin_L, HIGH);  
    digitalWrite(steppin_R, HIGH);  
    ...  
    }  
}
```

```
int step(float distance){  
    int steps = distance * steps_rev / (wheel_dia *  
3.1412);  
    return steps;  
}
```

```
void right(double degrees){  
    double rotation = degrees / 360;  
    double distance = dist * 3.14 * rotation;  
    int steps = step(distance);  
    digitalWrite(dirpin_L, LOW);  
    digitalWrite(dirpin_R, LOW);  
    for (int i = 0; i < steps; i++)  
    {digitalWrite(steppin_L, LOW);  
    digitalWrite(steppin_R, LOW);  
    digitalWrite(steppin_L, HIGH);  
    digitalWrite(steppin_R, HIGH);  
    ...  
    }  
}
```

```
void forward(double distance) {  
    int steps = step(distance);  
    digitalWrite(dirpin_L, LOW);  
    digitalWrite(dirpin_R, HIGH);  
    for (int i = 0; i < steps; i++)  
    {digitalWrite(steppin_L, LOW);  
    digitalWrite(steppin_R, LOW);  
    digitalWrite(steppin_L, HIGH);  
    digitalWrite(steppin_R, HIGH);  
    ...  
    }  
}  
  
void backward(double distance) {  
    int steps = step(distance);  
    digitalWrite(dirpin_L, HIGH);  
    digitalWrite(dirpin_R, LOW);  
    for (int i = 0; i < steps; i++)  
    {digitalWrite(steppin_L, HIGH);  
    digitalWrite(steppin_R, LOW);  
    digitalWrite(steppin_L, LOW);  
    digitalWrite(steppin_R, HIGH);  
    ...  
    }  
}
```

```
void circle(double r){  
    double r1 = r - dist/2;  
    double r2 = r + dist/2;  
    int steps_L = step(2*3.14*r1);  
    int steps_R = step(2*3.14*r2);  
    long positions[2];  
    positions[0] = steps_L;  
    positions[1] = steps_R;  
    steppers.moveTo(positions);  
    steppers.runSpeedToPosition(); //  
Blocks until all are in position  
}
```

```
void line(double slope, double intercept){  
  
    servo.write(0);  
        // pen up  
  
    int steps = step(intercept);  
  
    if(intercept < 0){  
        // for backward movement  
  
        digitalWrite(dirpin_L, HIGH);  
        digitalWrite(dirpin_R, LOW);  
  
    } else {  
        // for forward movement  
  
        digitalWrite(dirpin_L, LOW);  
        digitalWrite(dirpin_R, HIGH); }  
  
    for (int i = 0; i < steps; i++) {  
        // moving to y-intercept  
  
        digitalWrite(steppin_L, HIGH);  
        digitalWrite(steppin_R, HIGH);  
        delay(5);  
        digitalWrite(steppin_L, LOW);  
        digitalWrite(steppin_R, LOW);  
  
    delay(5); }
```

```
double radians=atan(slope);  
  
double degrees = radians*180/3.14;  
degrees = degrees-90;  
  
if(degrees<0){  
    // i.e. right rotation  
  
    degrees = degrees*(-1);  
    right(degrees);  
    servo.write(90);  
        // pen down  
  
    forward(10); } else {  
    // i.e. left rotation  
  
    left(degrees);  
    servo.write(90);  
        // pen down  
  
    forward(10); }}
```

WIRELESS CONTROL OF TURLEBOT

1. Bluetooth

- Bluetooth serial monitor can be used to control the bot.

2. WiFi

- Turtlebot can be controlled using a simple webpage.

Configuring Bluetooth for ESP32

```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;

void setup() {
    Serial.begin(115200);
    SerialBT.begin("ESP32test"); //Bluetooth device name
    Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop() {
    if (Serial.available()) {
        SerialBT.write(Serial.read());
    }
    if (SerialBT.available()) {
        Serial.write(SerialBT.read());
    }
    delay(20);
}
```



Bluetooth: On

[Turn Bluetooth Off](#)

Devices



Infinity Glide 500

Not Connected



ESP32

Not Connected



iPad

Not Connected

Mac OS X terminal window showing serial communication with an ESP32 device:

```
./dev/cu.ESP32-ESP32SPP
```

The terminal window includes the following controls at the bottom:

- Autoscroll Show timestamp
- Newline
- 115200 baud
- Clear output



Bluetooth



Jabra STEP Wireless v0.3.5



Infinity Glide 500



ZEB-BE350



Elizabeth



XECH Smart LED



ESP32



The screenshot shows a mobile application interface for managing Bluetooth connections. At the top, there are two tabs: "DEVICES" (selected) and "TERMINAL". The status bar indicates signal strength, battery level (28%), and the current time (9:38 PM). Below the tabs, there are two sections: "BLUETOOTH CLASSIC" and "BLUETOOTH LE".

- BLUETOOTH CLASSIC:**
 - ESP32**
24:6F:28:7C:6C:3A
 - Elizabeth**
68:76:4F:8D:C7:7C
 - Infinity Glide 500**
00:1E:7C:8E:09:77
 - JBL Flip 4**
00:42:79:A2:10:E2
 - JBL T460BT**
F4:BC:DA:66:E8:57
 - Jabra STEP Wireless v0.3.5**
50:1A:A5:29:4B:09
 - Jagadeesh Pai (GT-N710)**
BC:79:AD:AC:16:3A
 - Jagadeesh's iPad**
2C:61:F6:6B:4A:A3
 - Lenovo K50a40**
- BLUETOOTH LE:**

On the right side of the screen, there is a vertical stack of seven small gray rectangular buttons labeled M1 through M7. Below the device list, there are several small black navigation icons.

Log output (visible on the right):

```
21:35:43.047 Connecting to ESP32 ...
21:35:48.434 Connection failed: read failed, socket might close
d or timeout, read ret: -1
21:38:28.442 Connecting to ESP32 ...
21:38:29.572 Connected
```

Commands

- Forward- fd (dist. In cm.)
- Backward – bd (dist. In cm.)
- Right- rt (angle in degrees)
- Left – lt (angle in degrees)
- Line – ln (slope) (intercept)
- Circle- cl (radius)

Configuring Wi-Fi

- Station mode
- softAP mode

```
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <EEPROM.h>
#include<FS.h>
#include<Servo.h>
```

```
AsyncWebServer server(80);
String command="";
String input="";
const char* ssid = "";
const char* password = "";
const char* UNITS = "units";
const char* MOTION = "motion";
```

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8

IP Address: 192.168.1.9 ←

Autoscroll Show timestamp Newline 115200 baud Clear output

The screenshot shows a web browser window with the following details:

- Header:** The title bar says "ESP Input".
- Address Bar:** Shows "Not Secure | 192.168.1.9".
- Form Fields:** Two input fields labeled "Movement:" and "Number of units:", followed by a "Submit" button.
- COMMANDS Section:** A dark green box containing the heading "COMMANDS" and a bulleted list of movement commands.

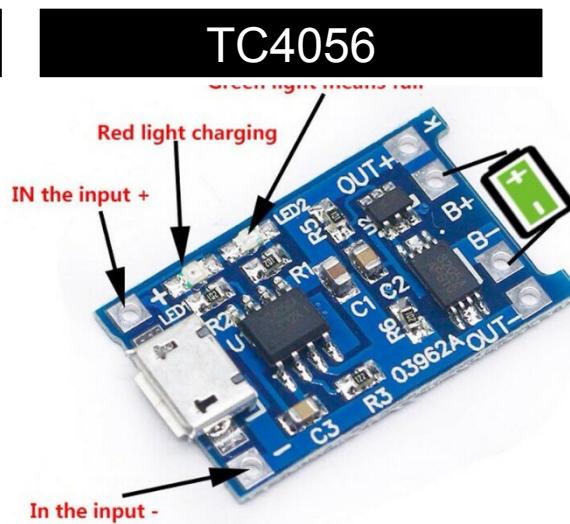
COMMANDS

- forward
- backward
- right
- left

BATTERY MANAGEMENT SYSTEM

COMPONENTS	CURRENT REQUIREMENT	VOLTAGE REQUIREMENT
ESP32	~70mA	6V-12V
DRV8825	Depends on the motor	>7.2V
28BYJ-48 STEPPER MOTOR	260mA	5V
SERVO MOTOR SG90	~150mA	3-6V
TOTAL	~740mA	>7.2V

18650 3.7V BATTERY

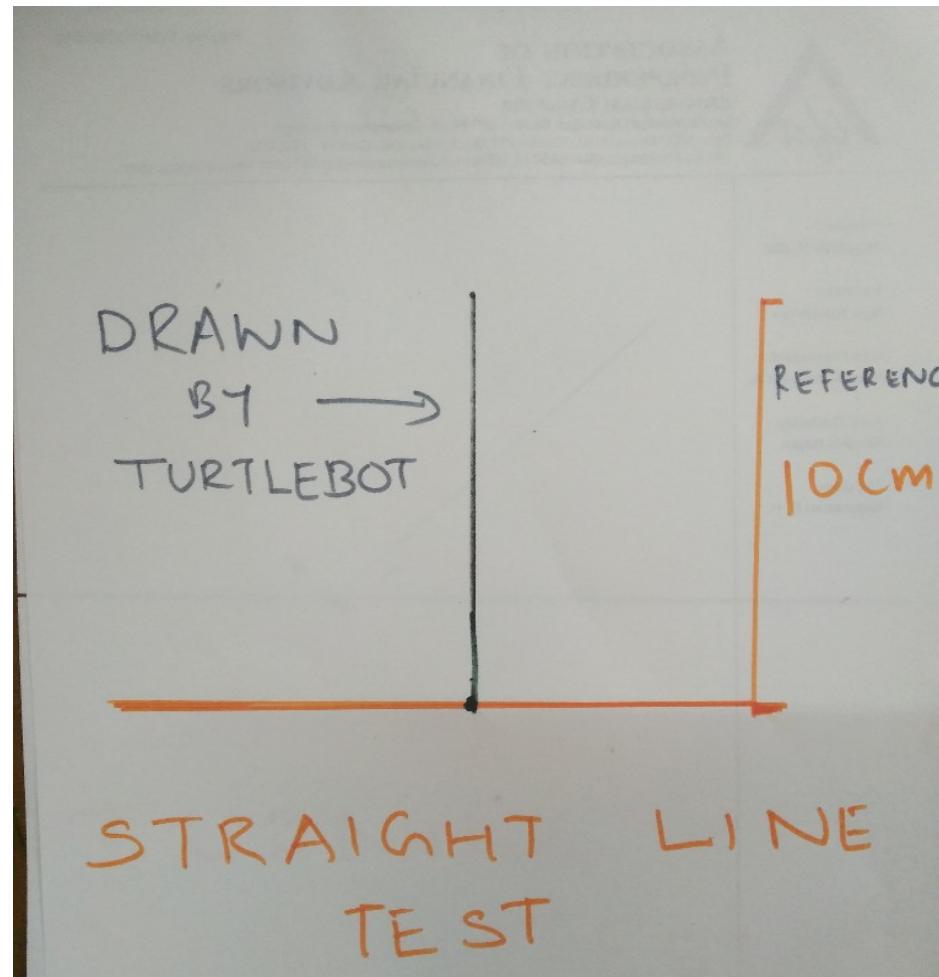


LIST

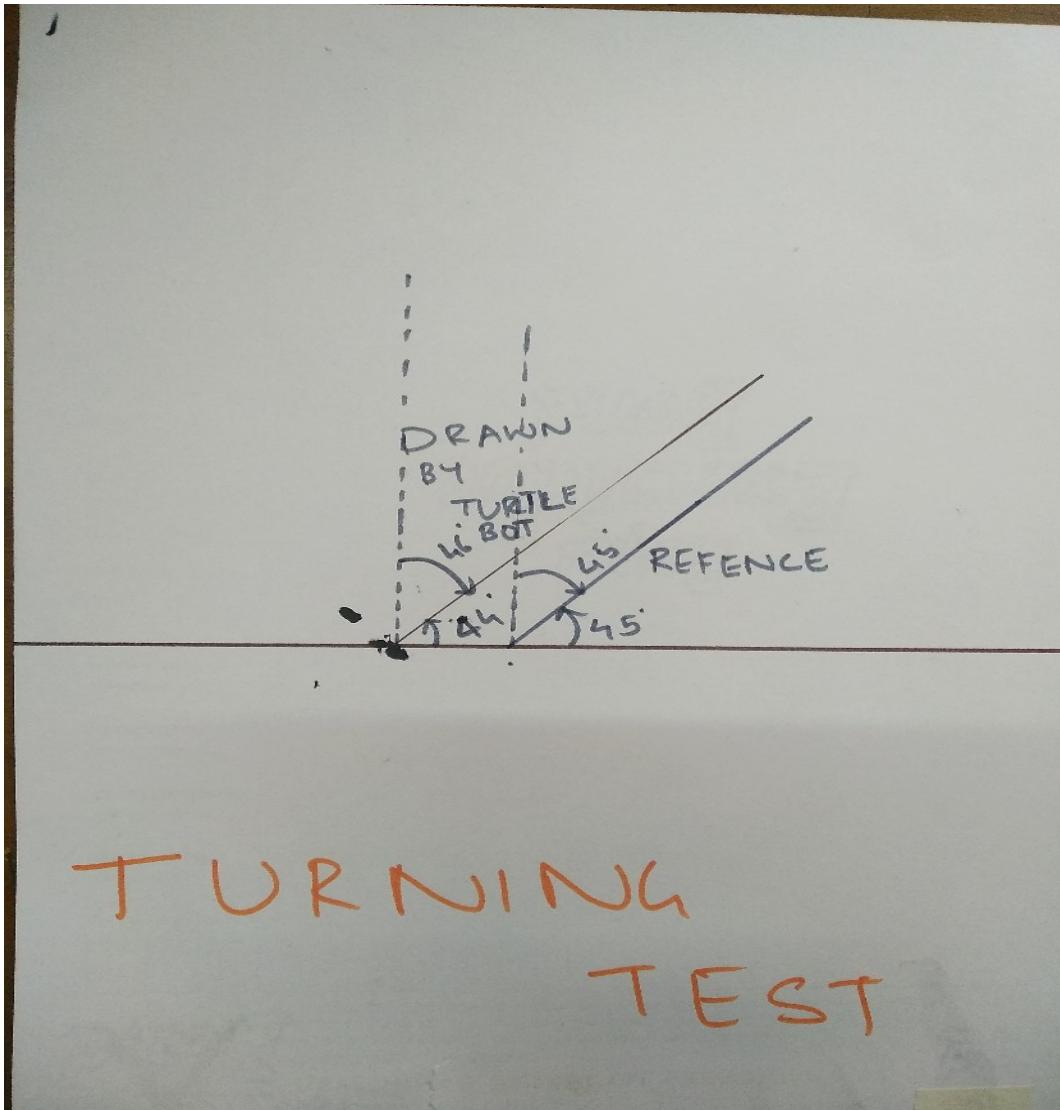
- 2 X 3.7V 3000mAh BATTERY IN SERIES
- 2 X TC4056 CHARGING MODULE

TESTING

Straight line test



Angle test



LINKS TO VIDEOS

Giving basic commands to turtle bot:

[https://drive.google.com/file/d/1IqURgPejmygs-NDymq2YInCCsQGx-_8/view?
usp=sharing](https://drive.google.com/file/d/1IqURgPejmygs-NDymq2YInCCsQGx-_8/view?usp=sharing)

Straight line test

[https://drive.google.com/file/d/12NjBvPkTxHSuO2kUyQ-ttJPAmx2gdVkl/view?
usp=sharing](https://drive.google.com/file/d/12NjBvPkTxHSuO2kUyQ-ttJPAmx2gdVkl/view?usp=sharing)

Angle test

[https://drive.google.com/file/d/182E2UXXKGur9vsdOEEpNczUd_yS9xaC5/vie
w?usp=sharing](https://drive.google.com/file/d/182E2UXXKGur9vsdOEEpNczUd_yS9xaC5/view?usp=sharing)

The link for the video for setting up the WiFi on ESP32 is

[https://drive.google.com/file/d/1_KRV_31bVkJzDSp_PRCWEgpMbcHOPk/v
iew?usp=sharing](https://drive.google.com/file/d/1_KRV_31bVkJzDSp_PRCWEgpMbcHOPk/view?usp=sharing)

LINKS FOR OUR VIDEOS

The link for WiFi control of the robot

1) Forward motion:

https://drive.google.com/file/d/14-rpY3t9FzSPIW_7IvS5MGizevlk8Cao/view?usp=drivesdk

2) Backward motion:

https://drive.google.com/file/d/1li_B-5eQ-PNpNNRiwqN2KZFJyDKmZGaq/view?usp=drivesdk

Turtlebot tries to draw 'A'

https://drive.google.com/file/d/1TyyJFoYREN8ObCxuNGViKgtV5nu6_u5t/view?usp=sharing

Turtlebot tries to draw a circle

<https://drive.google.com/file/d/1AY-KWkjH9jjBOrNws6DDFNbZUIFBGZbY/view?usp=sharing>

Turtlebot executing multiple commands sequentially

<https://drive.google.com/file/d/1zljFVV1Mzp06uLFbJJ4elzD-28XZ1Gmr/view?usp=sharing>

TURTLEBOT

3D MODELING
-DHAMMAPADA MOHAPATRA

IMPORTANCE AND NEED OF 3D MODELING

- Firstly its a quantitative approach towards designing and assembly of components.
- It gives exact position, relative position and distance, orientation of various components in an assembly.
- Secondly it is quite easy for interpretation by any person and give his/her feedbacks.
- Thus always giving a scope to rectify any error and simultaneously providing opportunity for further upgradation of the project.
- Hence always better than trial and error method of approach.

SOFTWARE SELECTION

- It is preferred to use Open Source software for the project.
- For this purpose selection was made between OpenSCAD, BRL CAD, Free CAD and similar other software.
- On a personal note, I am comfortable with drag and drop based approach of modelling and hence FreeCAD v0.18 was selected for 3D modeling.

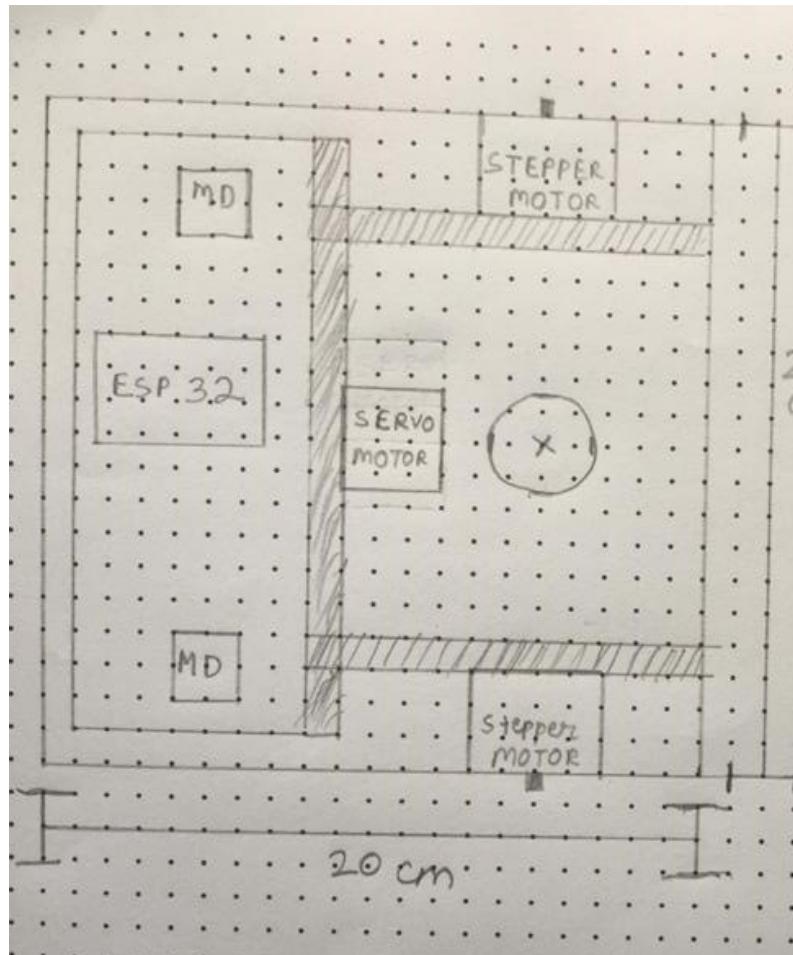
WORK FLOW

- Entire 3D modeling was divided into 5 parts:
 1. A primary and rough chassis design (subjected to changes at every next stage)
 2. Electro-Mechanical assembly
 3. Electrical assembly
 4. Battery management system
 5. Pen lifting and lowering mechanism

ROUGH DESIGN OF CHASSIS

- ▶ Various chassis design were made initially and on basis of following constraints selection was made:
 1. Weight and torque balancing
 2. Specific orientation of mechanical & electro -mechanical equipments
 3. Geometry of components
- A selected design drawn on orthographic paper is attached in the next slide
- Also in the figure, additive tolerance of one to two millimeters is given o the components.

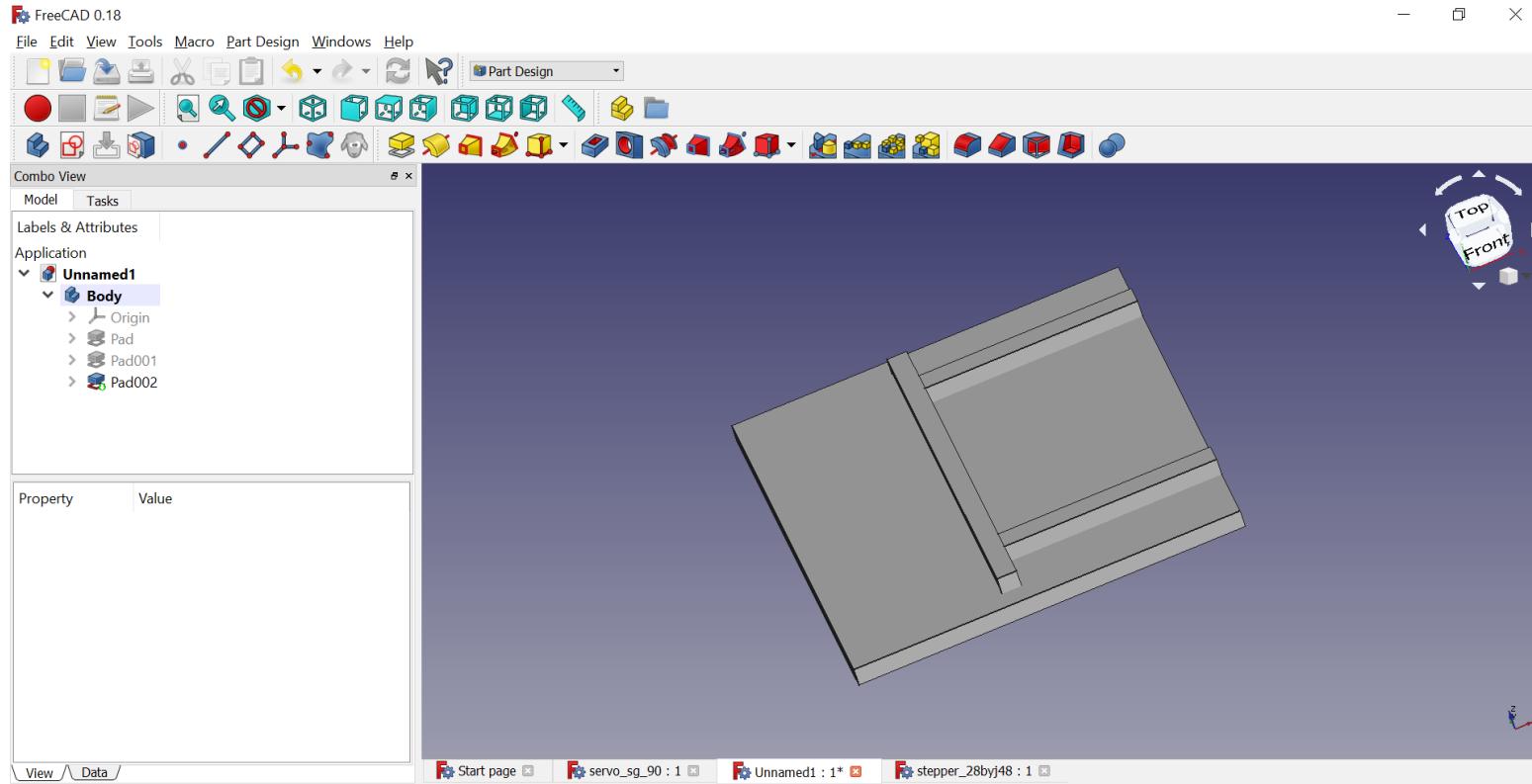
PROPOSED 20X20 CM CHASSIS



	Actual dimension		Used
Motor driver	20.5 x 15.1	X	30 x 30
ESP 32	51 x 30	✓	Enough Space is available
Servo motor	23 x 12.2 & 29	X	30 x 30
Stepper	See dim 35 x 19 + 1	X	40 x 30
Bread board	174 x 66	✓	180 x 70 + 1 cm is left

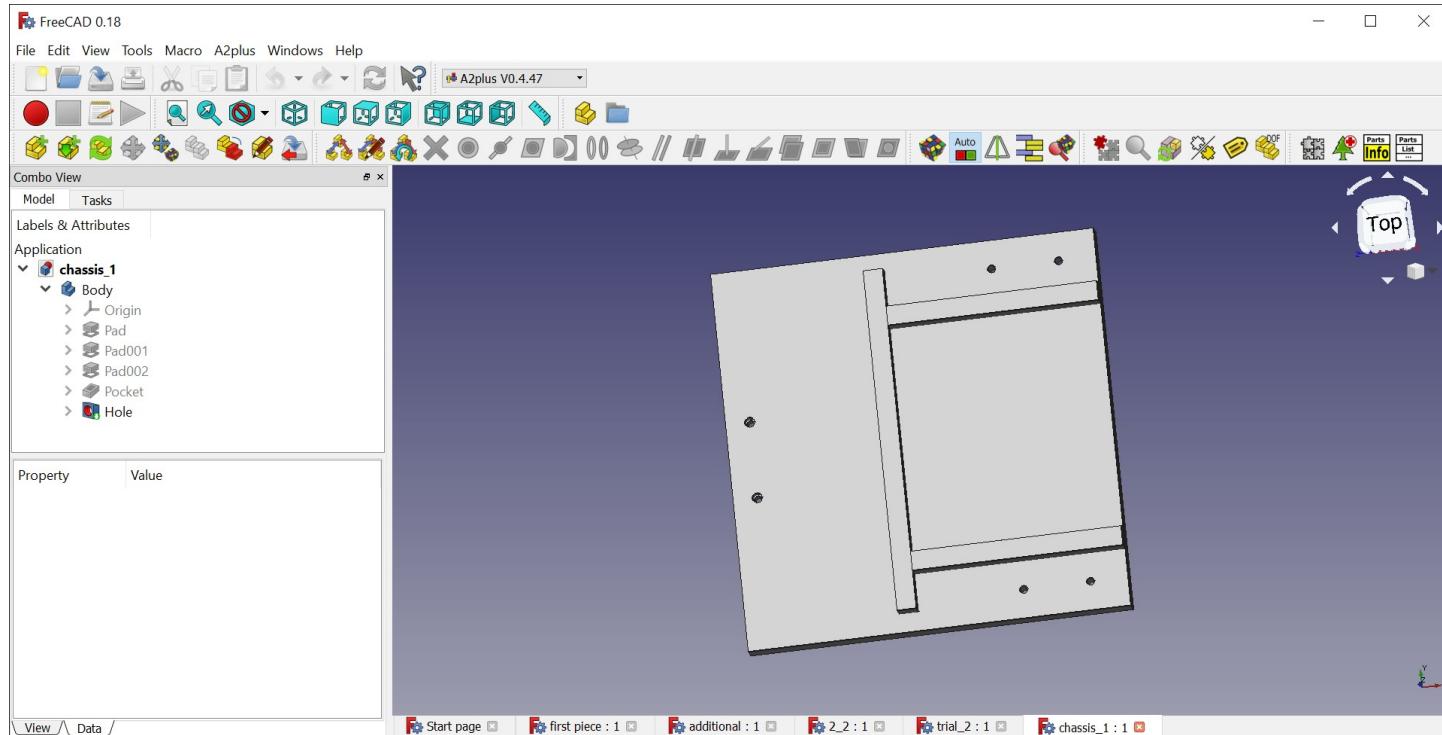
1 grid = 1cm
Proposed Chassis design (20 cm X 20 cm)

CHANGES IN CHASSIS DESIGN



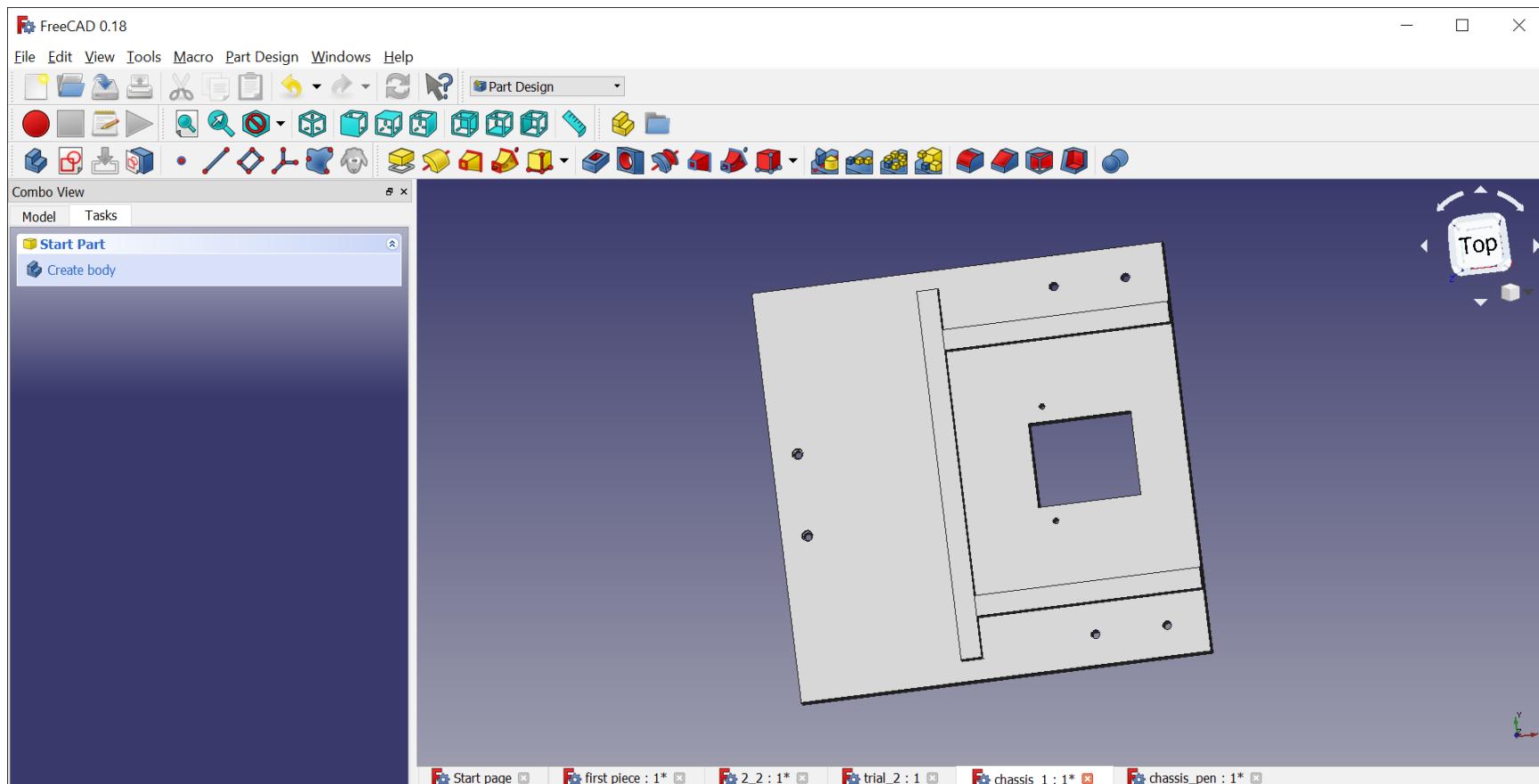
Basic design

CHANGES IN CHASSIS DESIGN



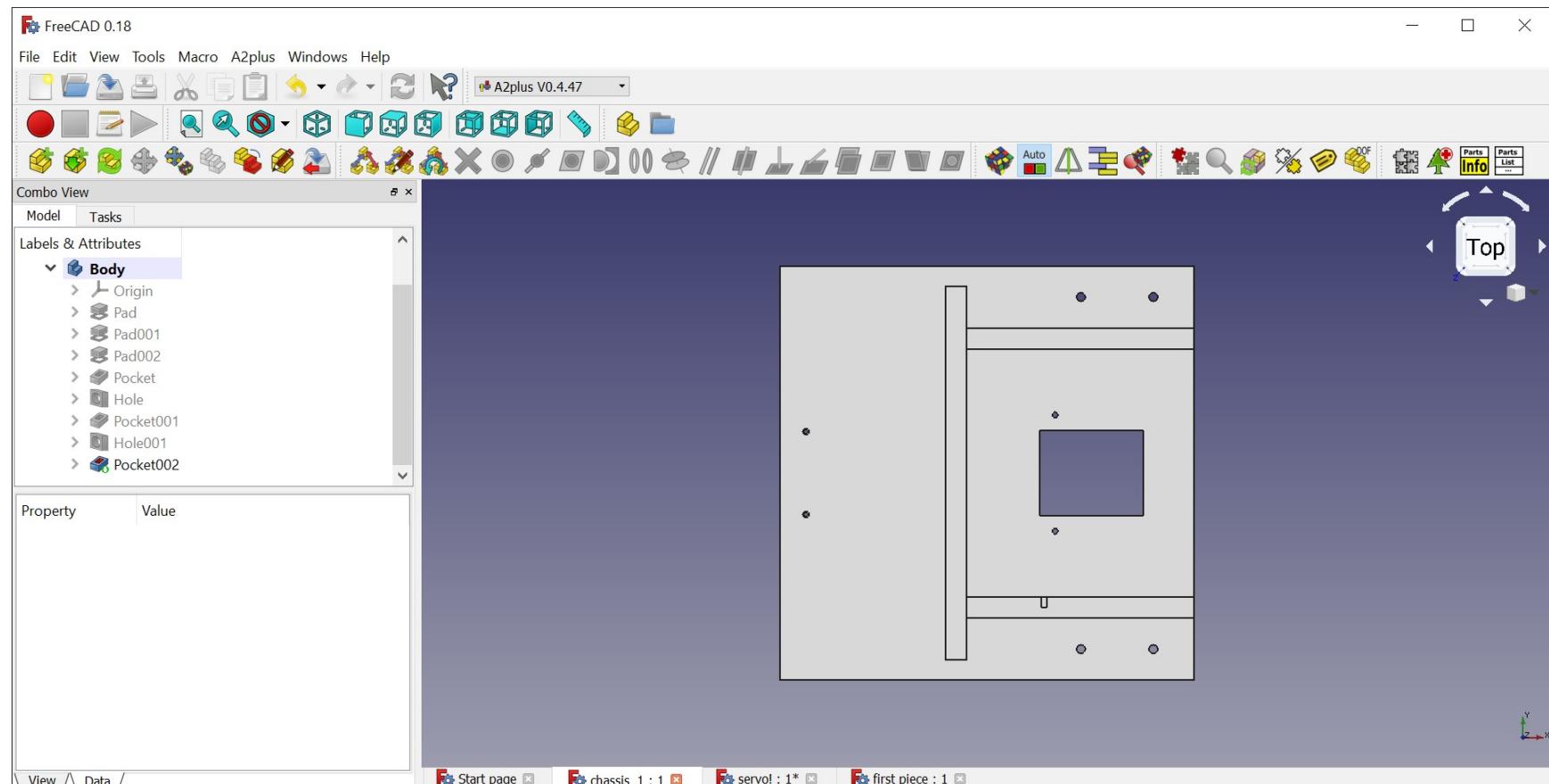
Chassis with holes :
Front : Caster Ball : M2
Rear : stepper motor brackets : M5

CHANGES IN CHASSIS DESIGN



With added holes for pen lifting and lowering techniques

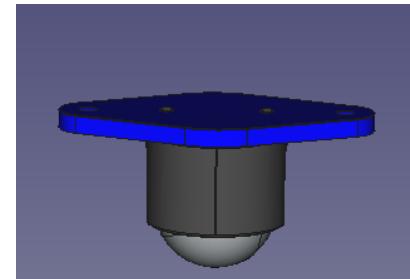
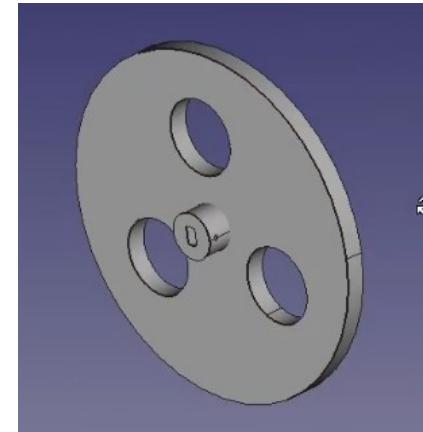
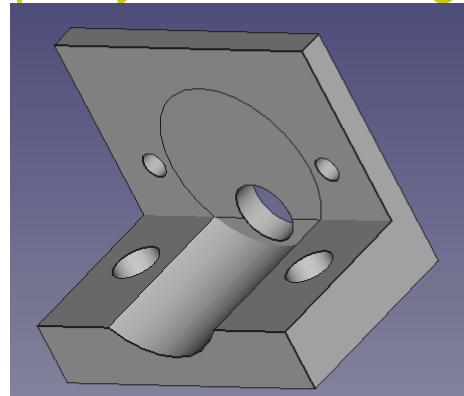
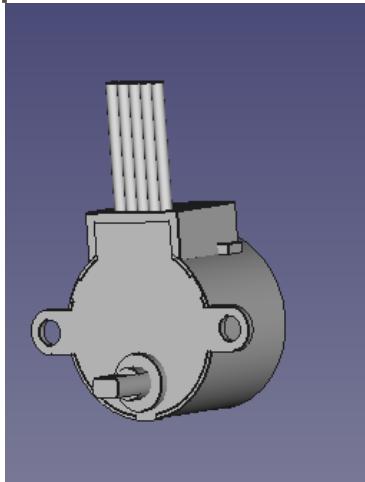
CHANGES IN CHASSIS DESIGN



With added cut on lower support wall to accommodate servo motor

ELECTRO MECHANICAL ASSEMBLY

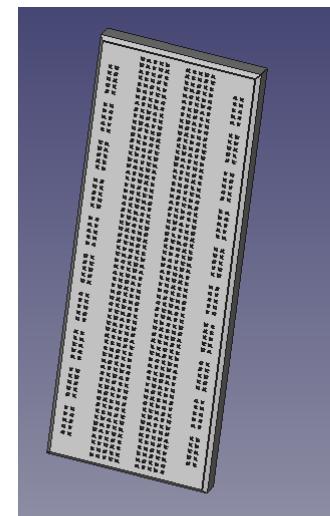
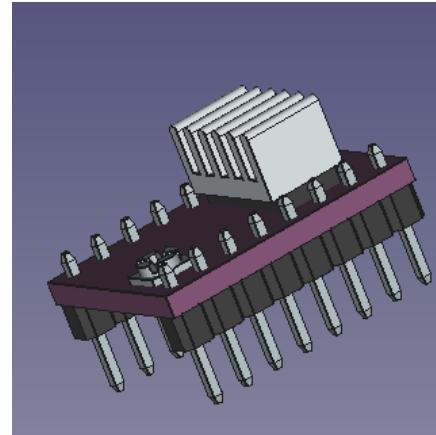
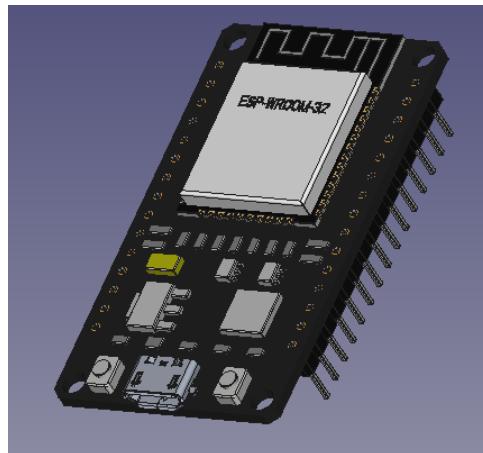
- It involves attachment of wheels, caster ball, stepper motor, support brackets with the chassis via fasteners
- Assembly: <https://youtu.be/6wjJIstEaW0>
- Exploded assembly : <https://youtu.be/zw7gvPr8HGQ>



ELECTRO MECHANICAL ASSEMBLY

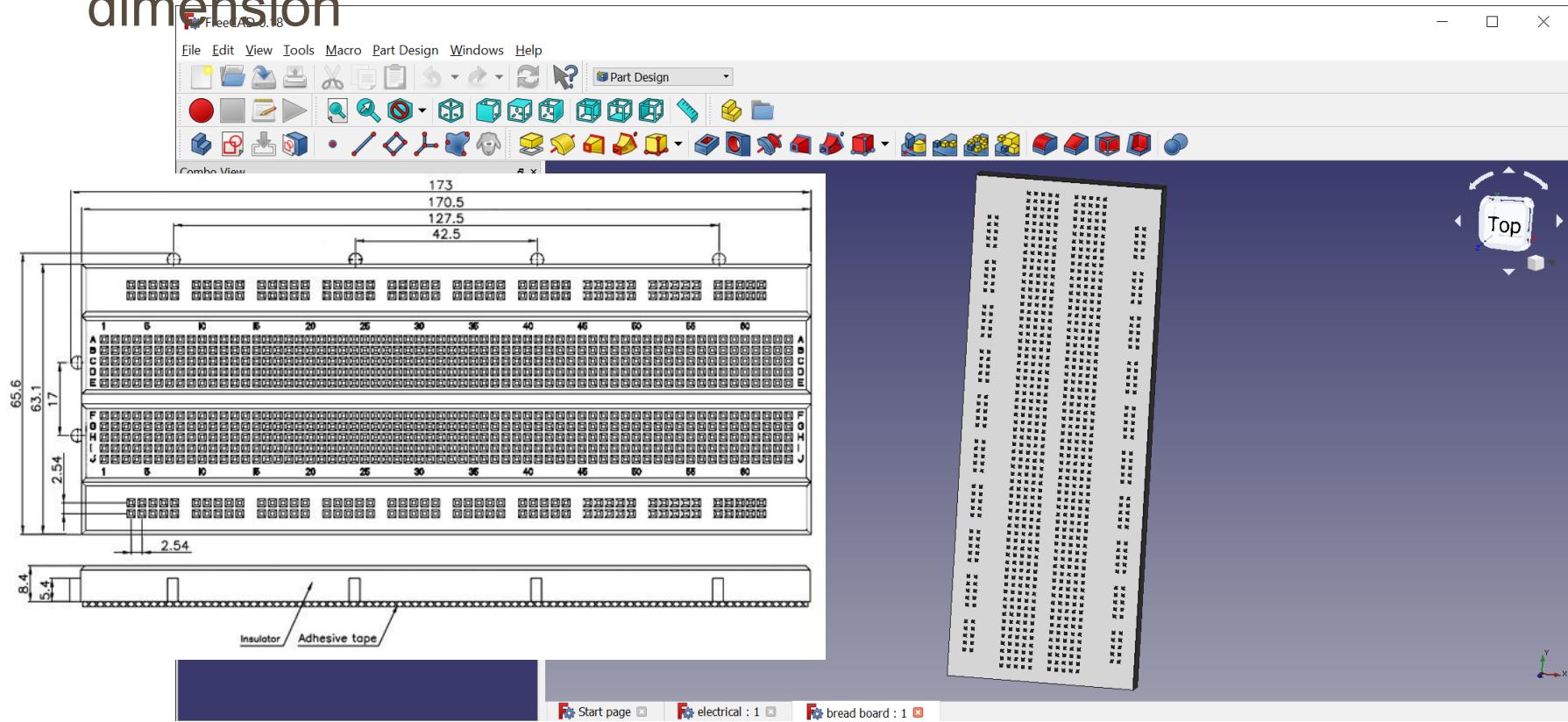
ELECTRICAL ASSEMBLY

- It involves electrical parts –
- ESP 32 DEVKITC (Micro-controller)
- Motor Driver (DRV8825)
- Breadboard (170.5X 63.1X 8.4 mm)



DESIGN OF BREADBOARD

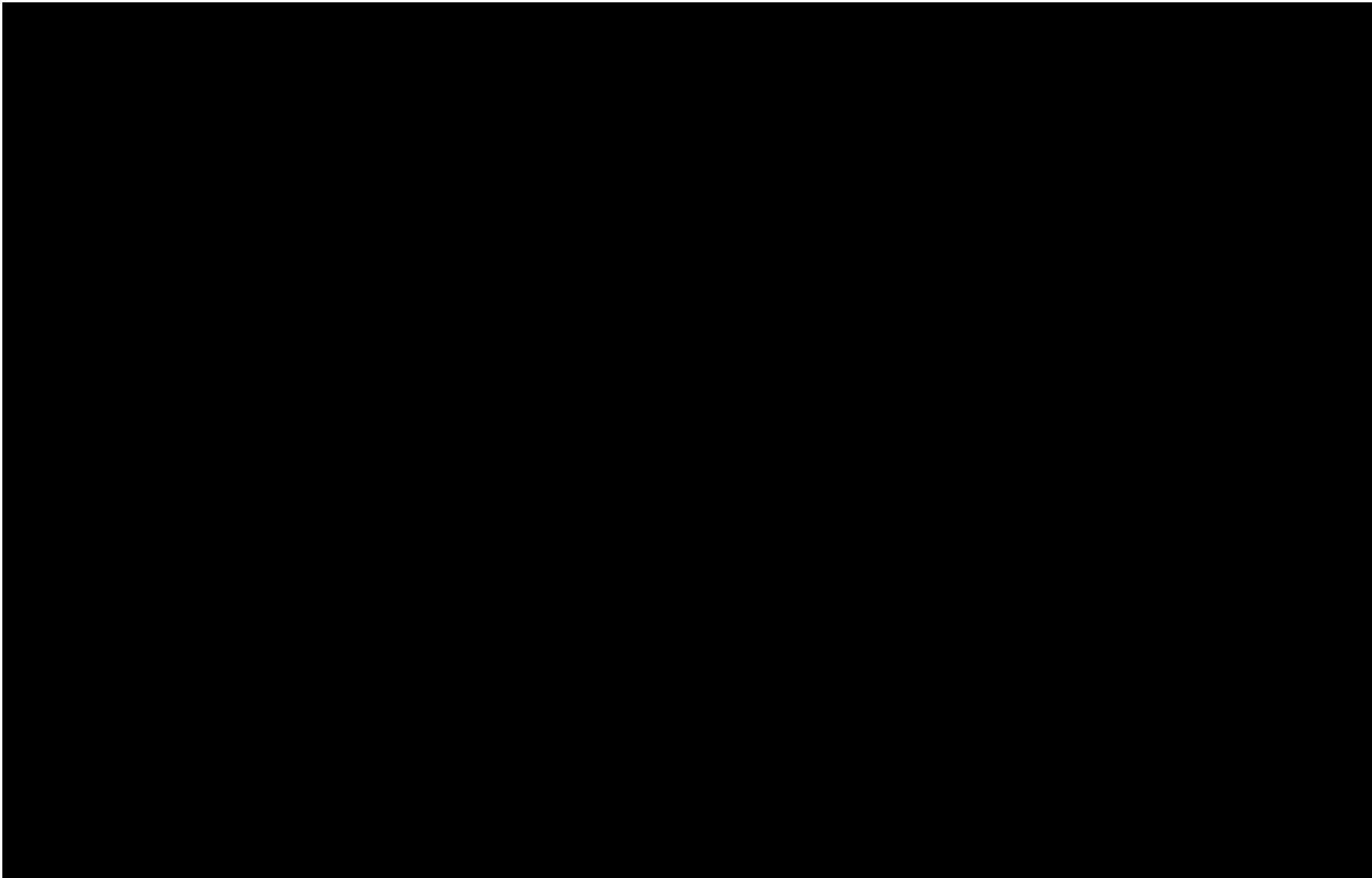
- Complete design was done on basis of attached dimension



DESIGN OF BREADBOARD

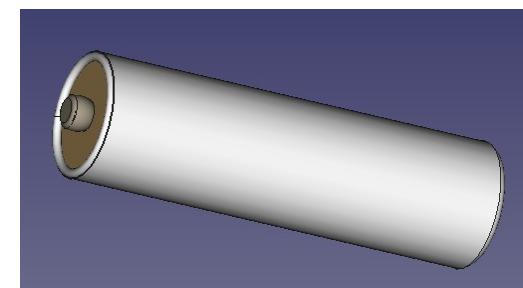
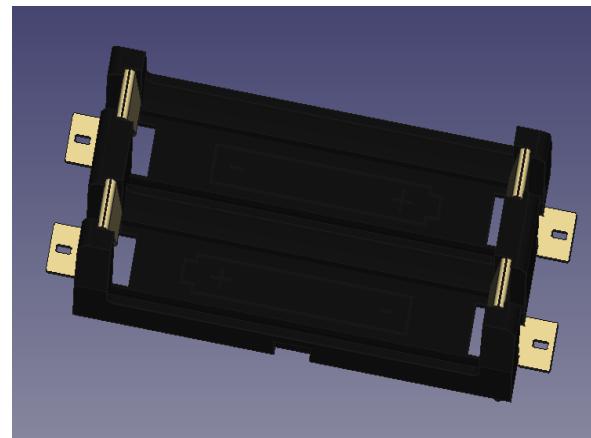
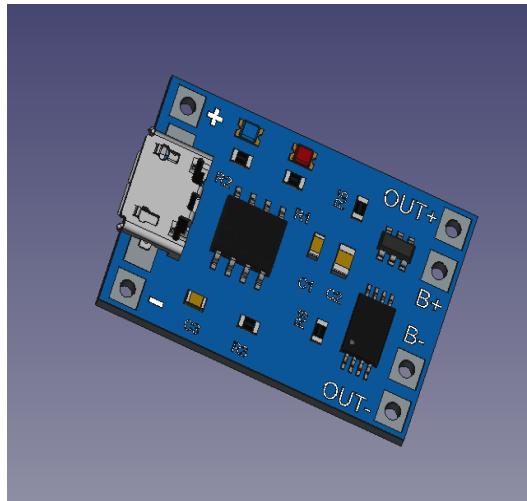
- A lot of variables were missing and gave rise to errors
 1. Dimension of single hole is missing
 2. Position of at least one hole from either the ends are missing
 3. Inter-distancing of holes of various rows and columns are missing
- Design of breadboard :
- <https://youtu.be/nMw-iiXcBq8>

ELECTRICAL ASSEMBLY

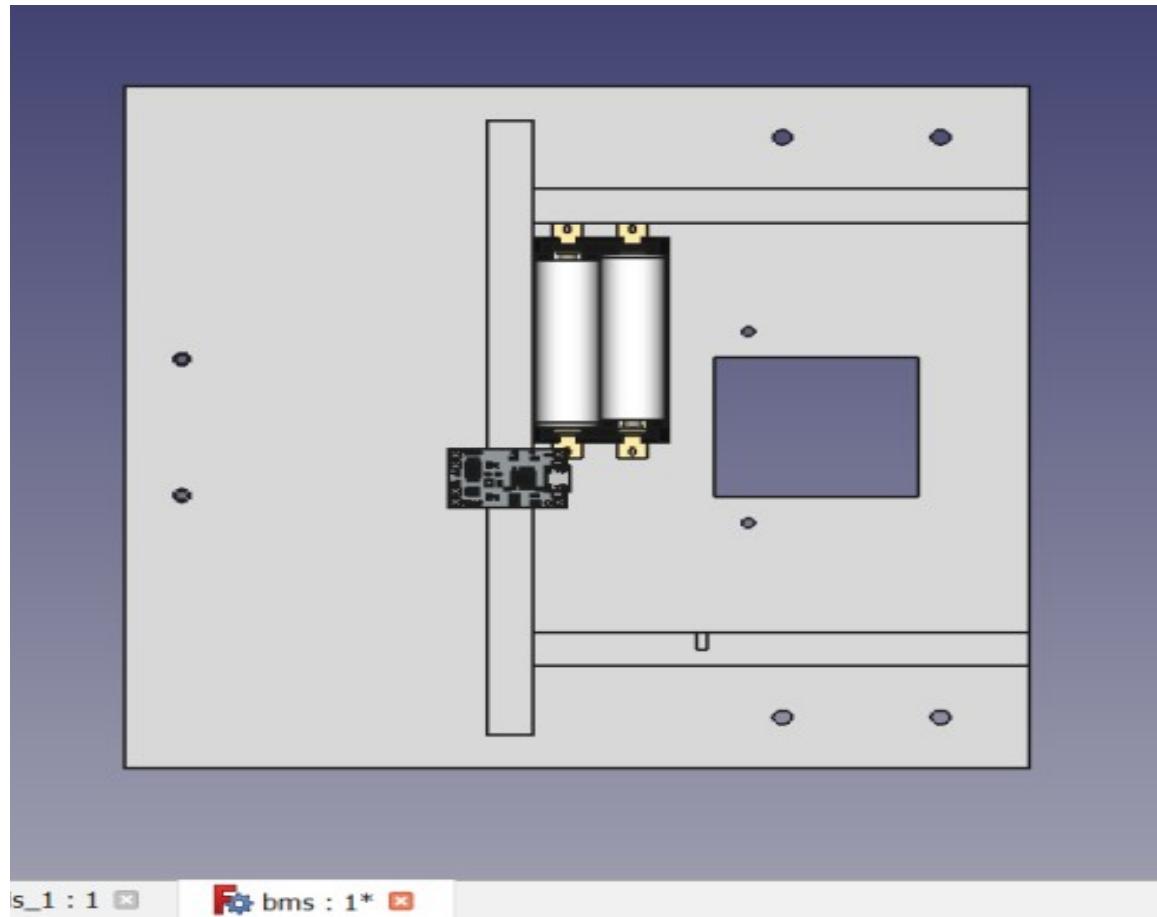


BATTERY MANAGEMENT SYSTEM

- For this purpose we used TC4056 charging module, Li ion battery (3.7v) and a battery holder.

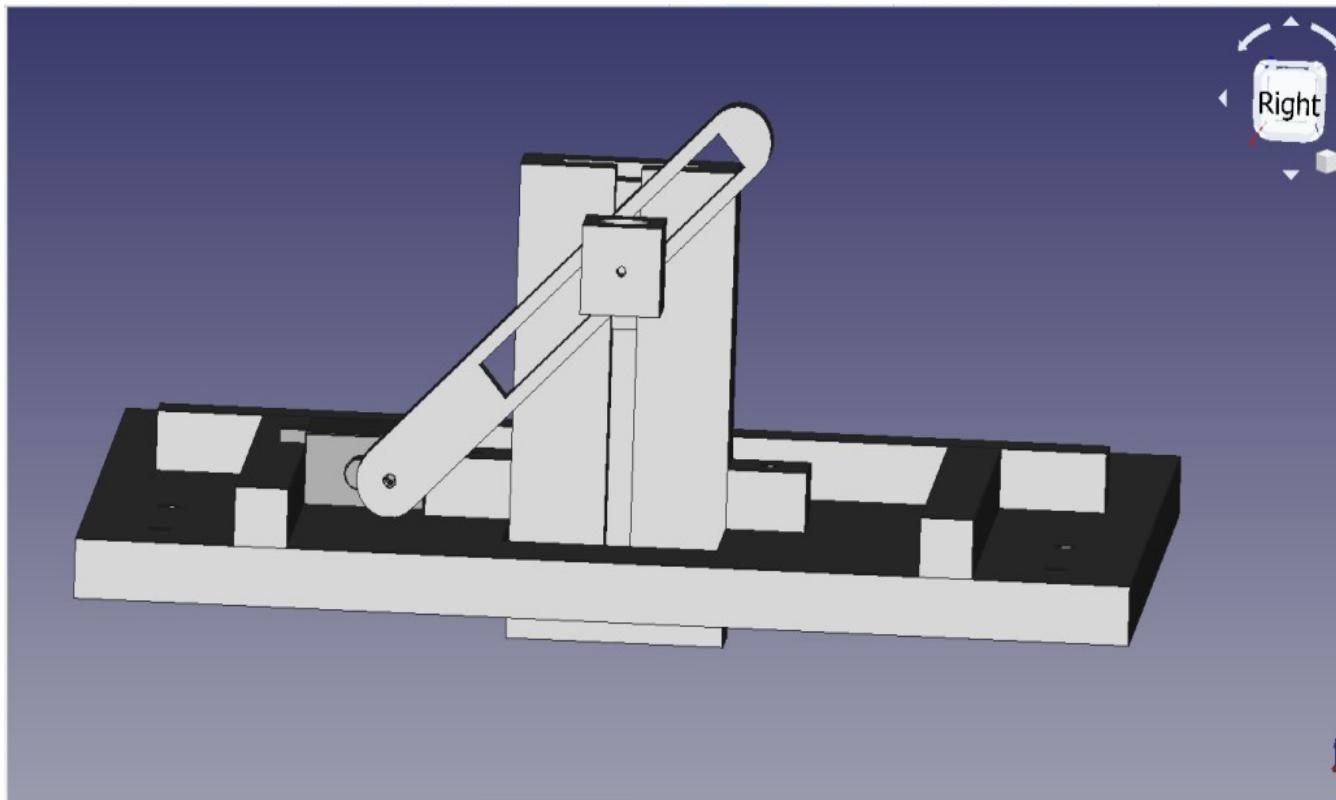


BATTERY MANAGEMENT SYSTEM



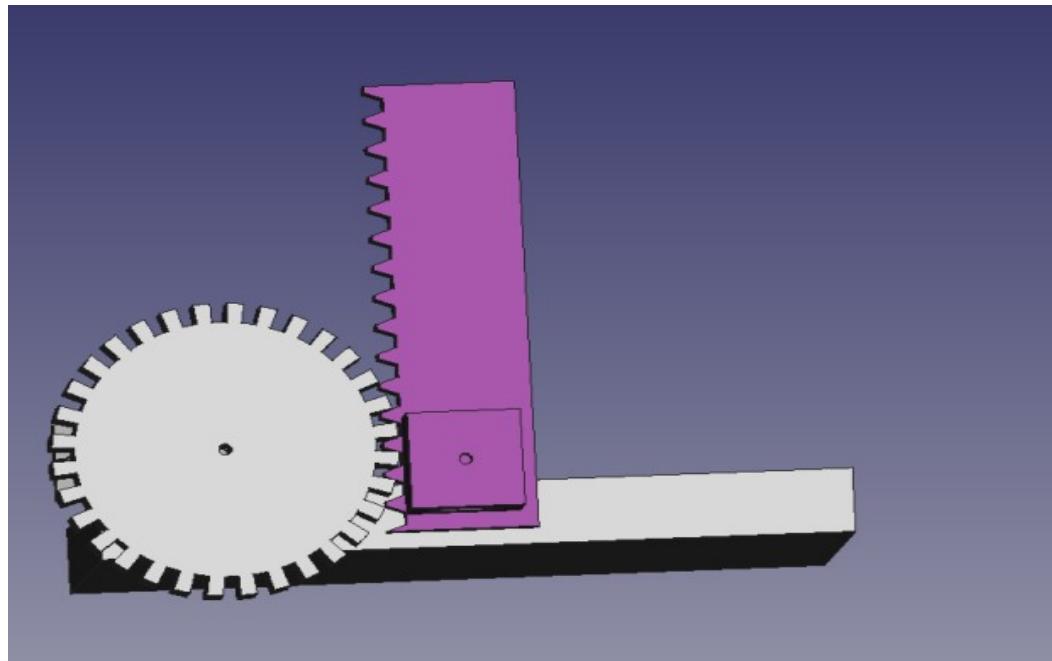
PEN LIFTING AND LOWERING MECHANISM

- Sliding mechanism were preferred.
- Assembly of servo motor with mechanism.



PEN LIFTING AND LOWERING MECHANISM

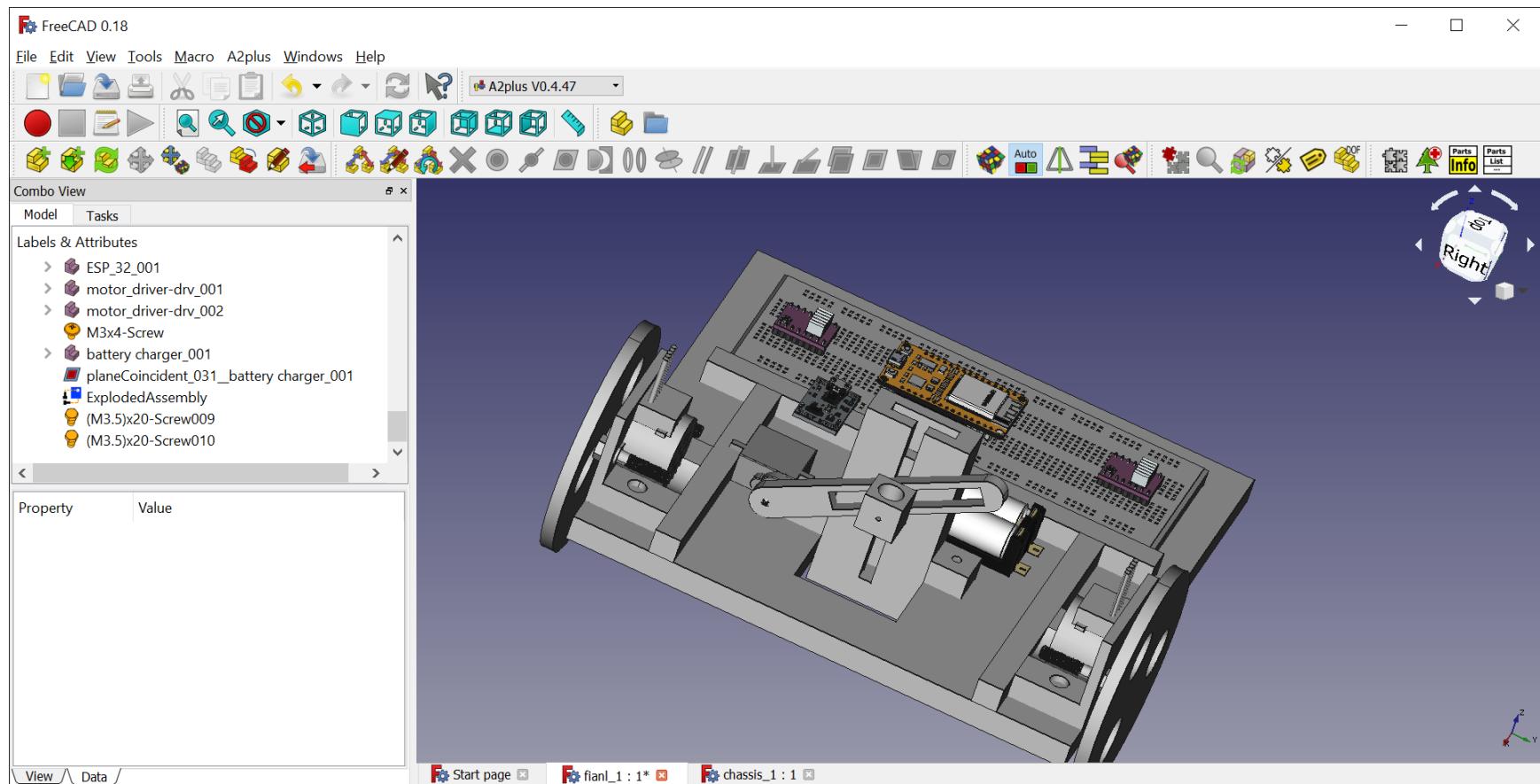
- Another mechanism i.e. Rack and pinion mechanism was also proposed
- It involves a spur gear (pinion) and a toothed slider called as rack. Gear is attached to servo motor and it meshes with rack. It also has a casing for rack to fix its position.



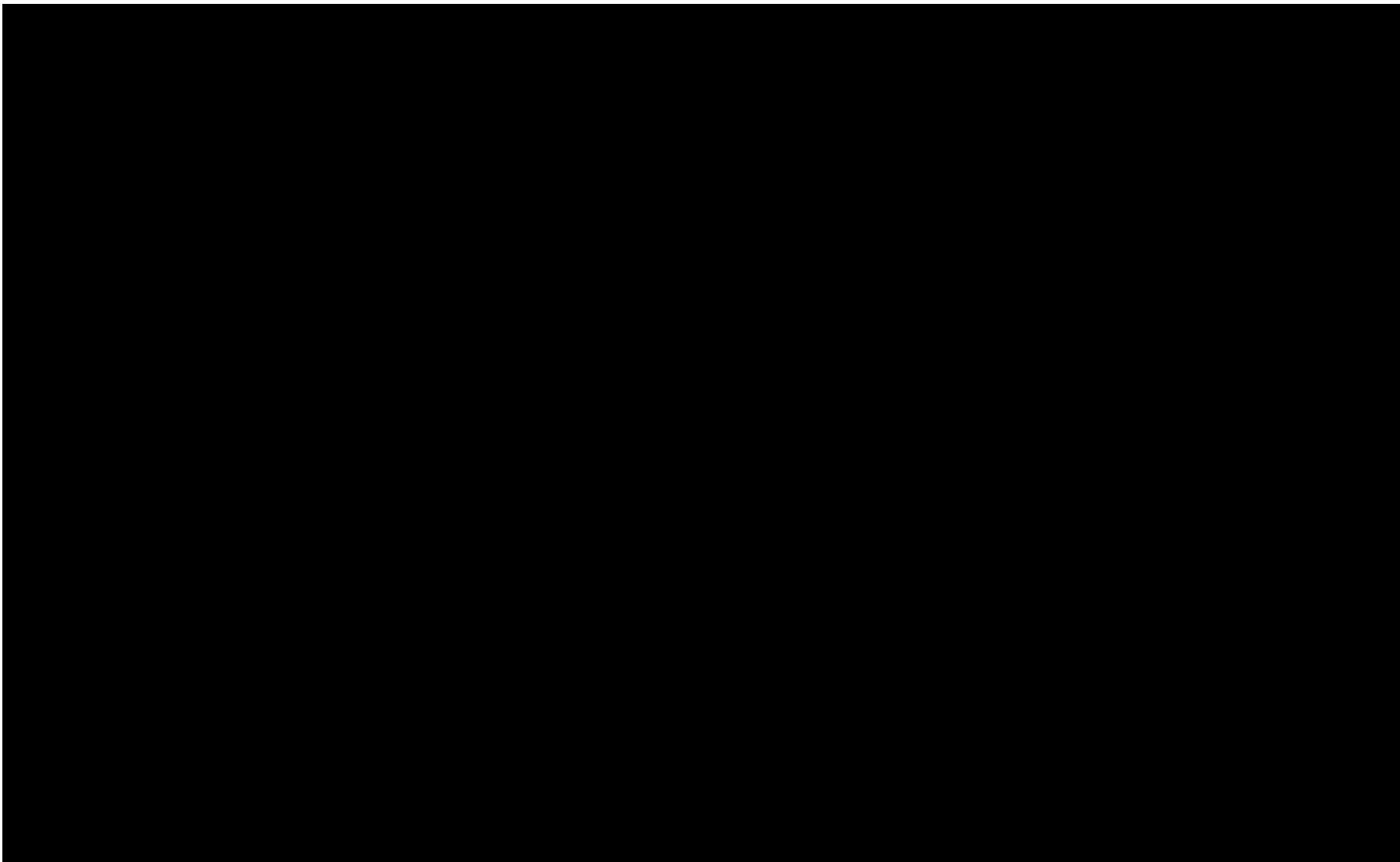
PEN LIFTING AND LOWERING MECHANISM

- Out of the two mechanism sliding was preferred because as in case of rack and pinion mechanism, there is a higher probability that it may not work either due to lack of meshing (even a bit of inaccuracy may lead to failure) and other factors like over and undercutting of meshed tooth.

RESULT: COMBINED ASSEMBLY WITH FASTENERS ATTACHED



RESULT: COMBINED ASSEMBLY WITH FASTENERS ATTACHED



SOME OTHER IMPORTANT POINTS

- Thick wheels were preferred to increase friction and reduce the momentum of bot thus allowing a smooth and non-sliding turn.
- Discussion were held on orientation of pen holding part (i.e. tilt or no tilt) and its impact on friction . In conclusion the weight of pen is balanced by pen holding part and not by ground. Thus it's a zero normal contact and hence weight of pen doesn't play in determining friction.

LINKS FOR DESIGN AND ASSEMBLY:

- <https://youtu.be/nMw-iiXcBq8> - breadboard design 1
- <https://youtu.be/Kqc17-Jy2LI> - breadboard 2/electrical assembly
- <https://youtu.be/WBas0Xoxg9Q> - electrical and electromechanical assembly
- <https://youtu.be/Vnx021O-YiY> - wheel design
- <https://youtu.be/UswX56JsxDY> - 9V dc battery design
- <https://youtu.be/pRFn5OuC8Yc> - exploded trial assembly (prefinal)
- <https://youtu.be/6wjJlstEaW0> - electro-mechanical assembly
- <https://youtu.be/zw7gvPr8HGQ> - exploded electromechanical assembly
- https://youtu.be/dIWP_I81fDU - exploded electrical assembly
- <https://youtu.be/EHoBY8gCtp8> - exploded assembly pre-final
- <https://youtu.be/eRUBuXxHkuU> - exploded assembly final
-