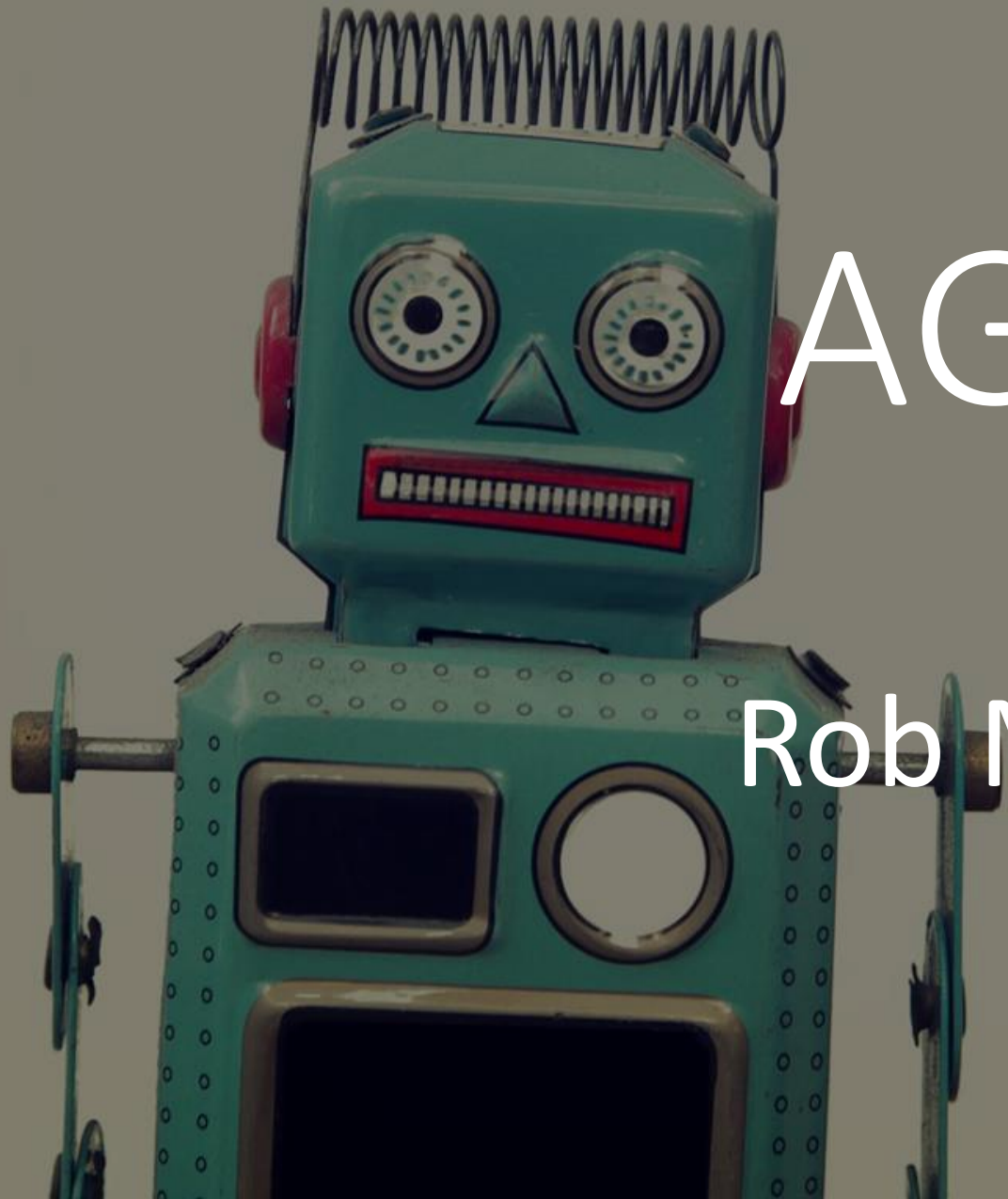
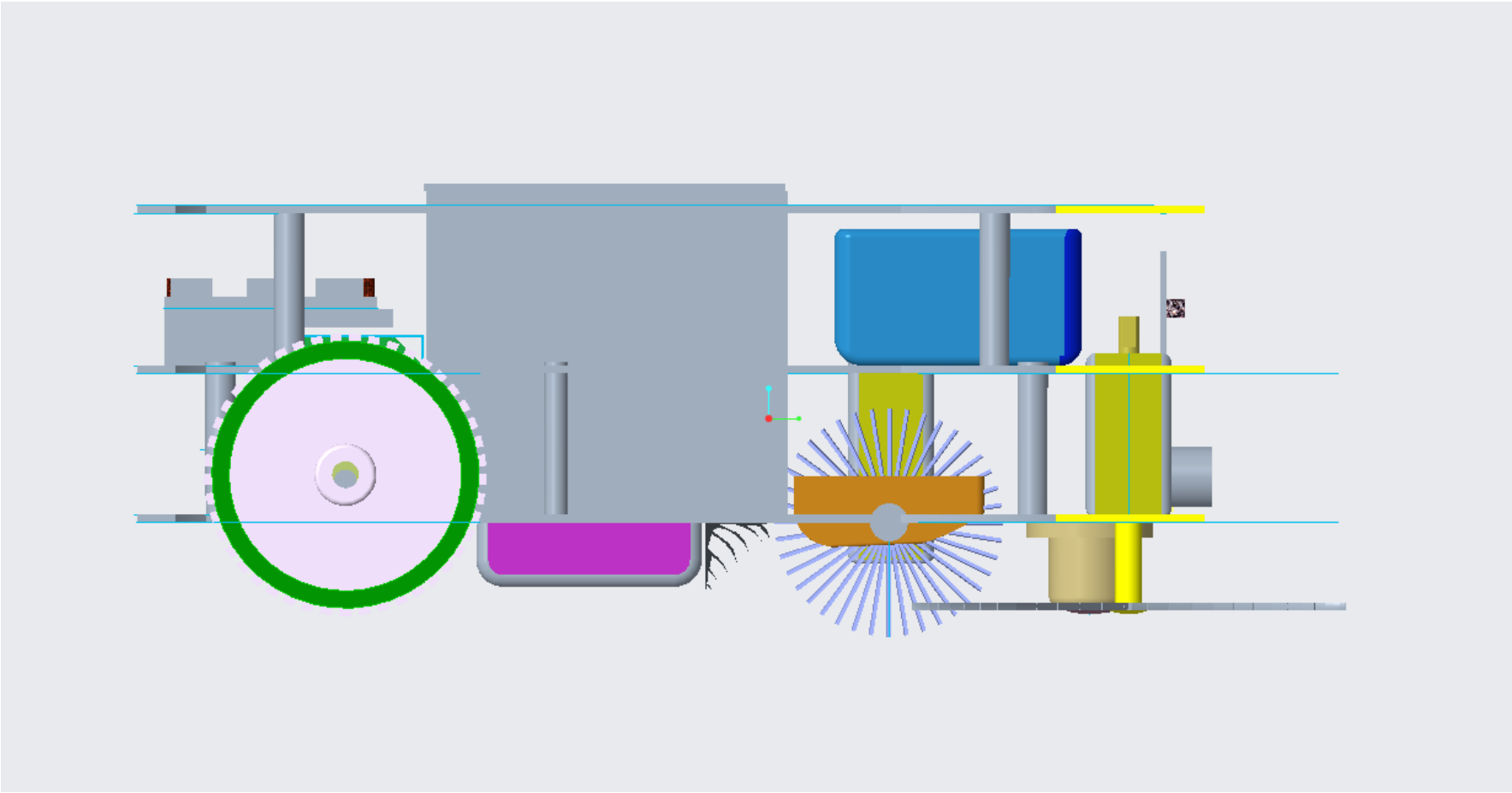


AGV

Rob Mop

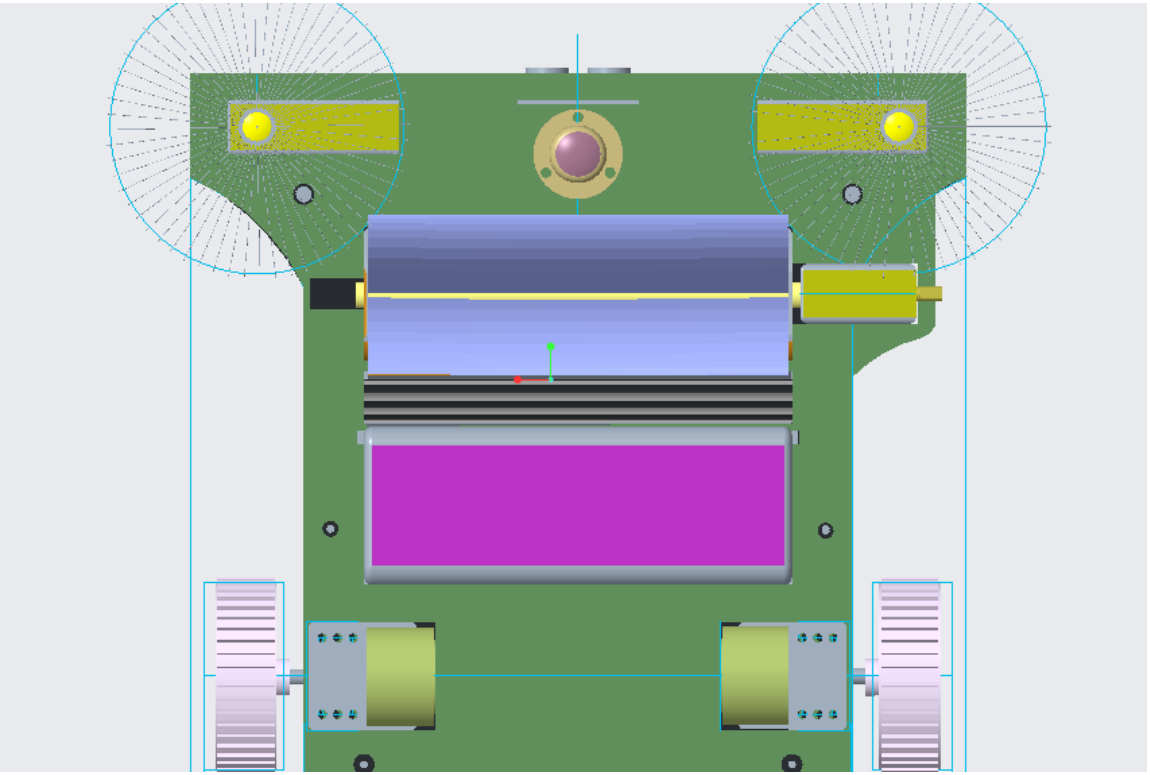
Automatic guided vehicle (AGV)

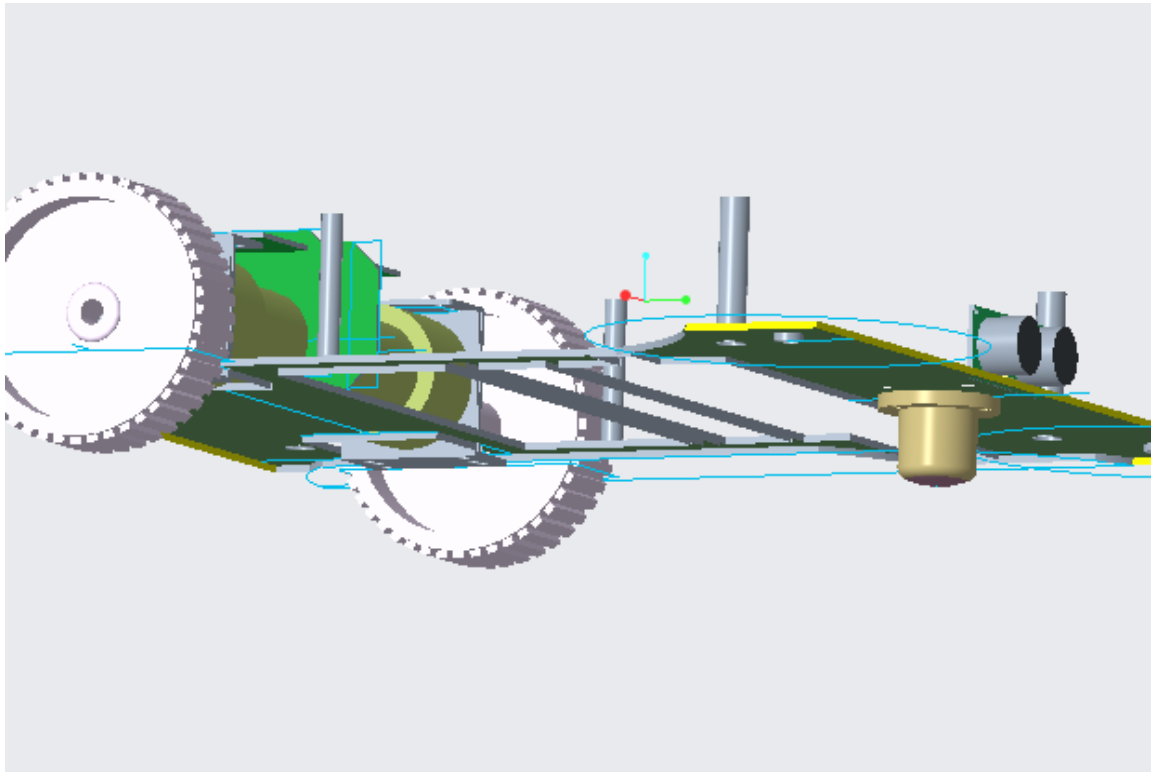


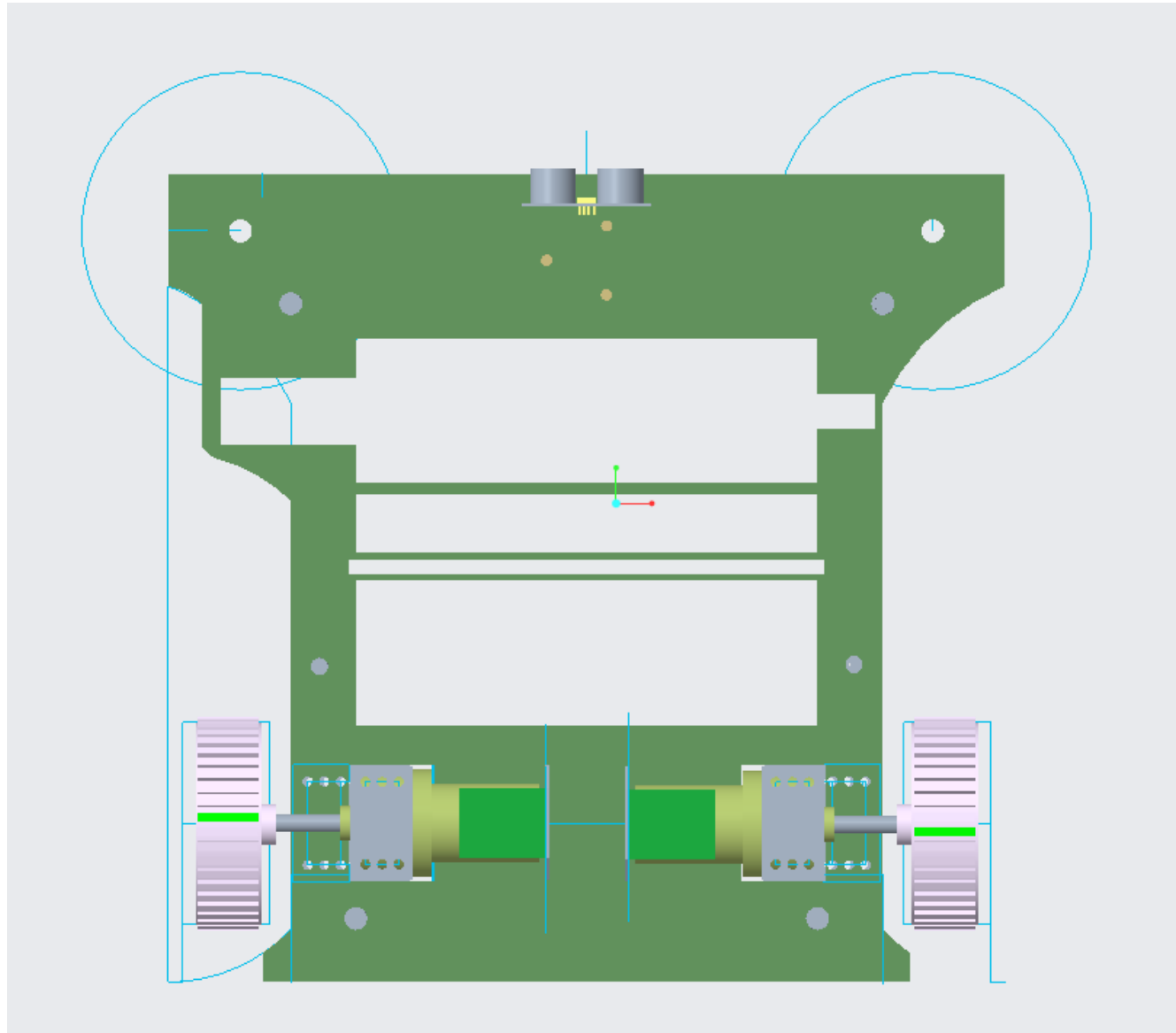
Components used

Base	Senosrs	Controllers	Others
Acrylic sheets-3mm	Ultrasonic sensor	Raspberry pi -4 model B	Motor
	Infrared sensor	Arduino UNO	Wheels-r(3.5cm)
	IMU(Inertial measurement unit)	Motor controller	Castor wheel
	Encoder		Clamps
			Spacers
			Breadboard

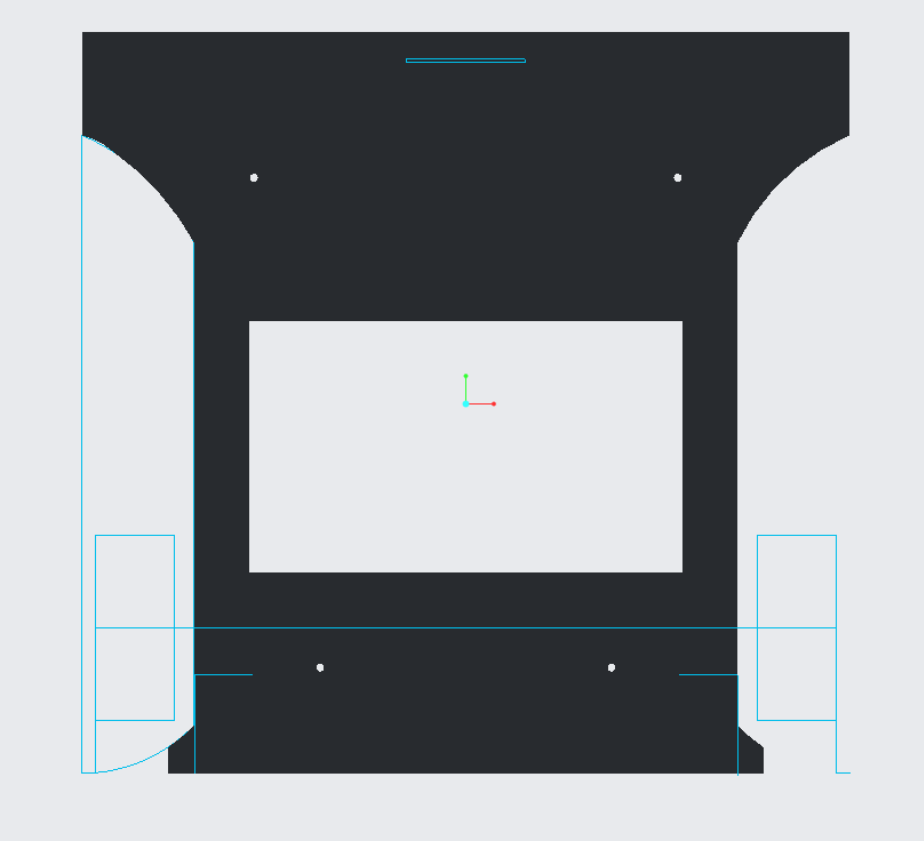
Base

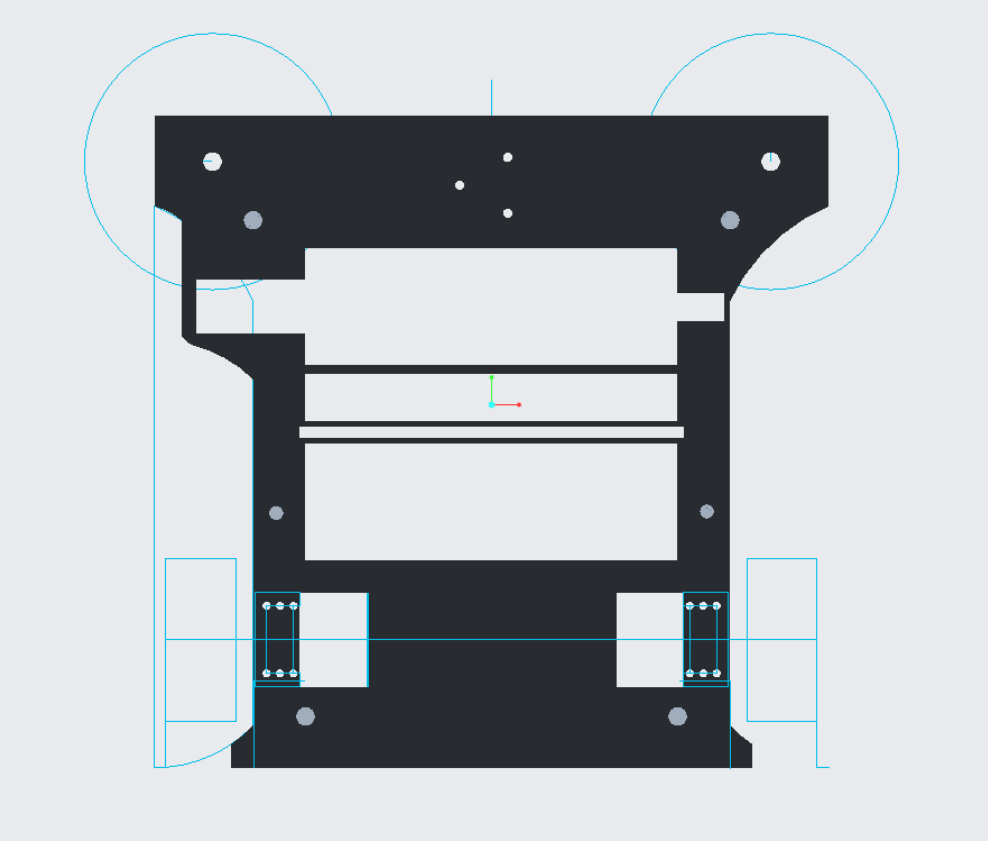




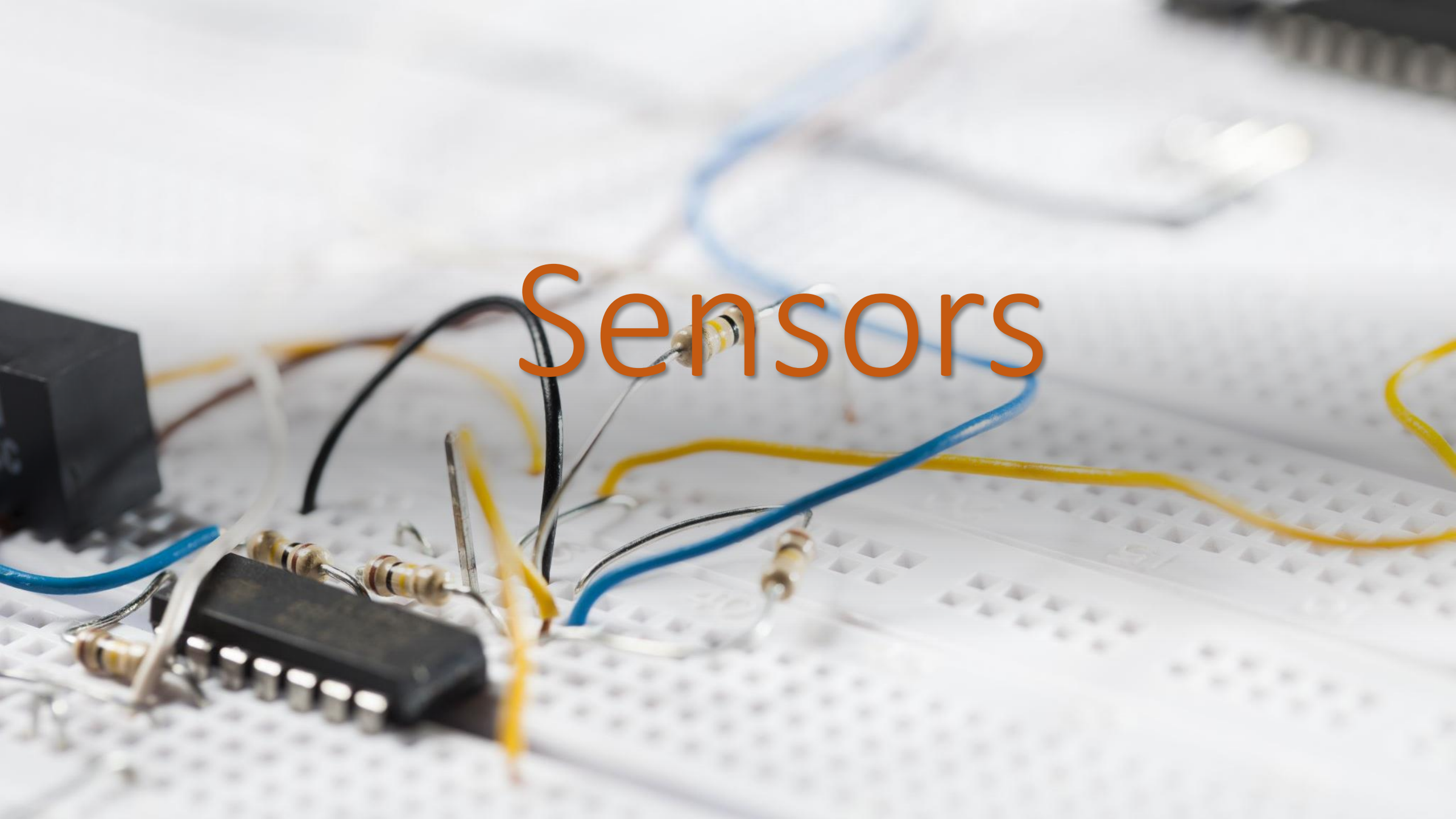


Base part -1



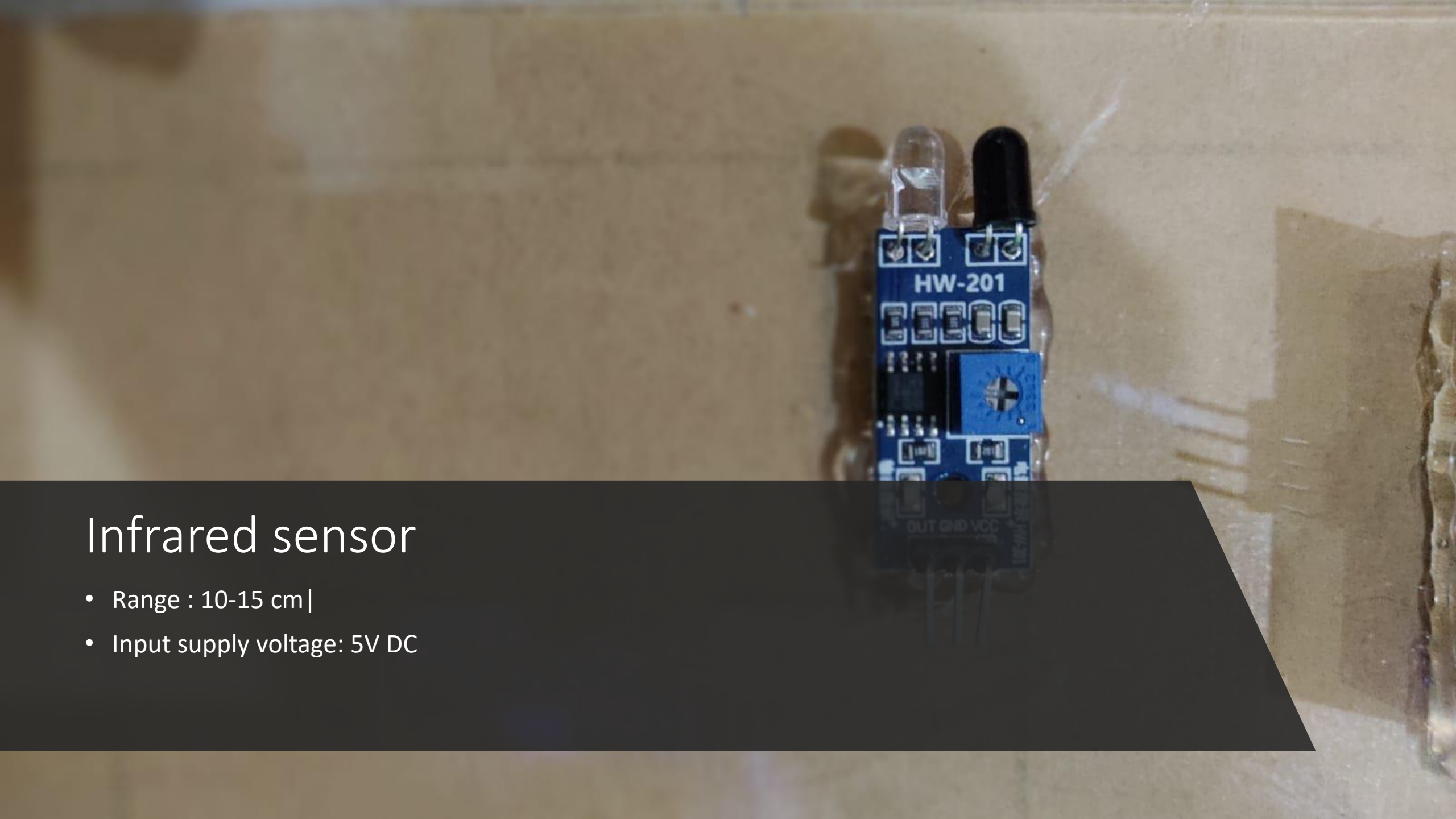


Acrylic sheet – Base part 1


Sensors

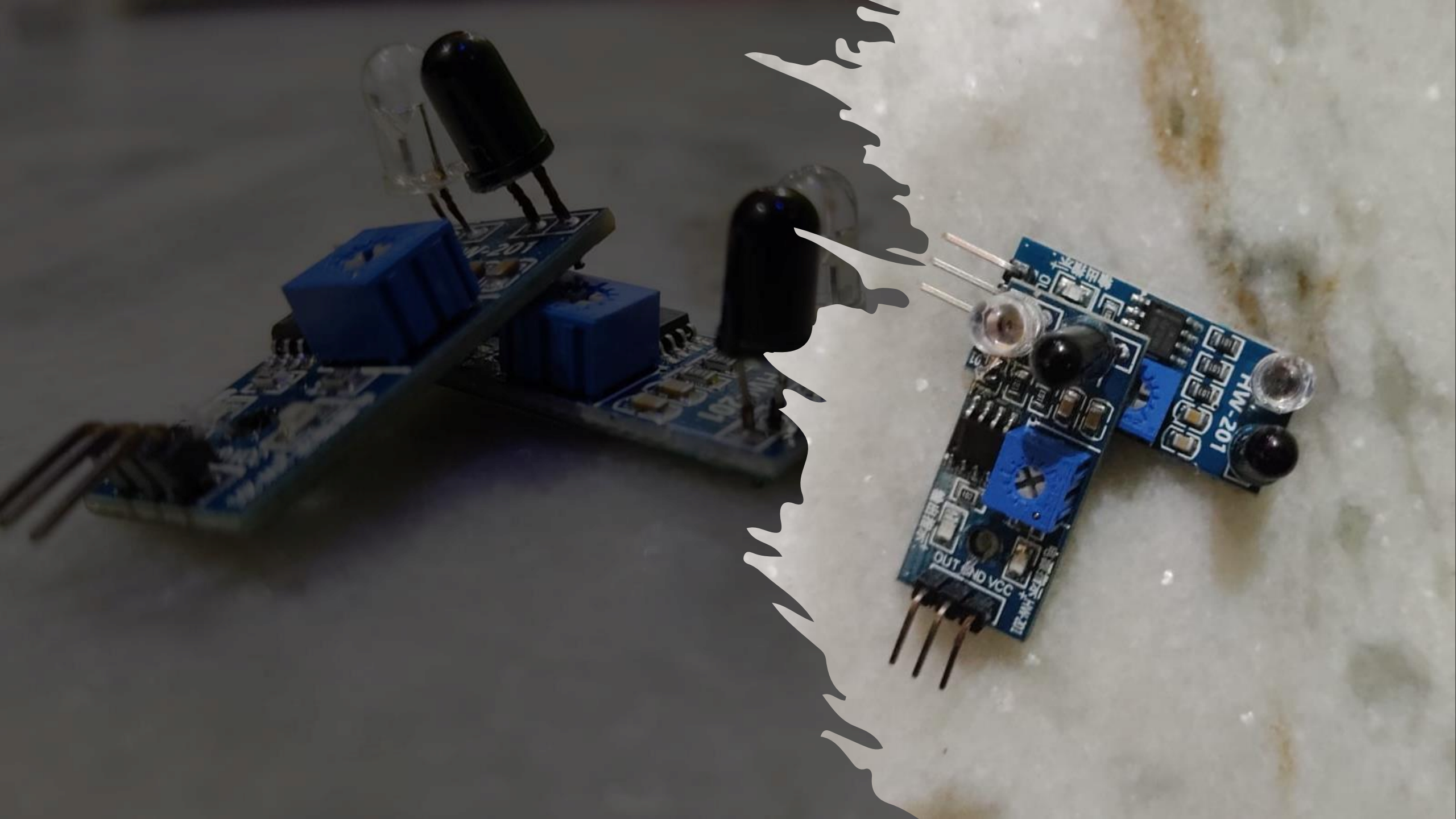
Ultrasonic sensor

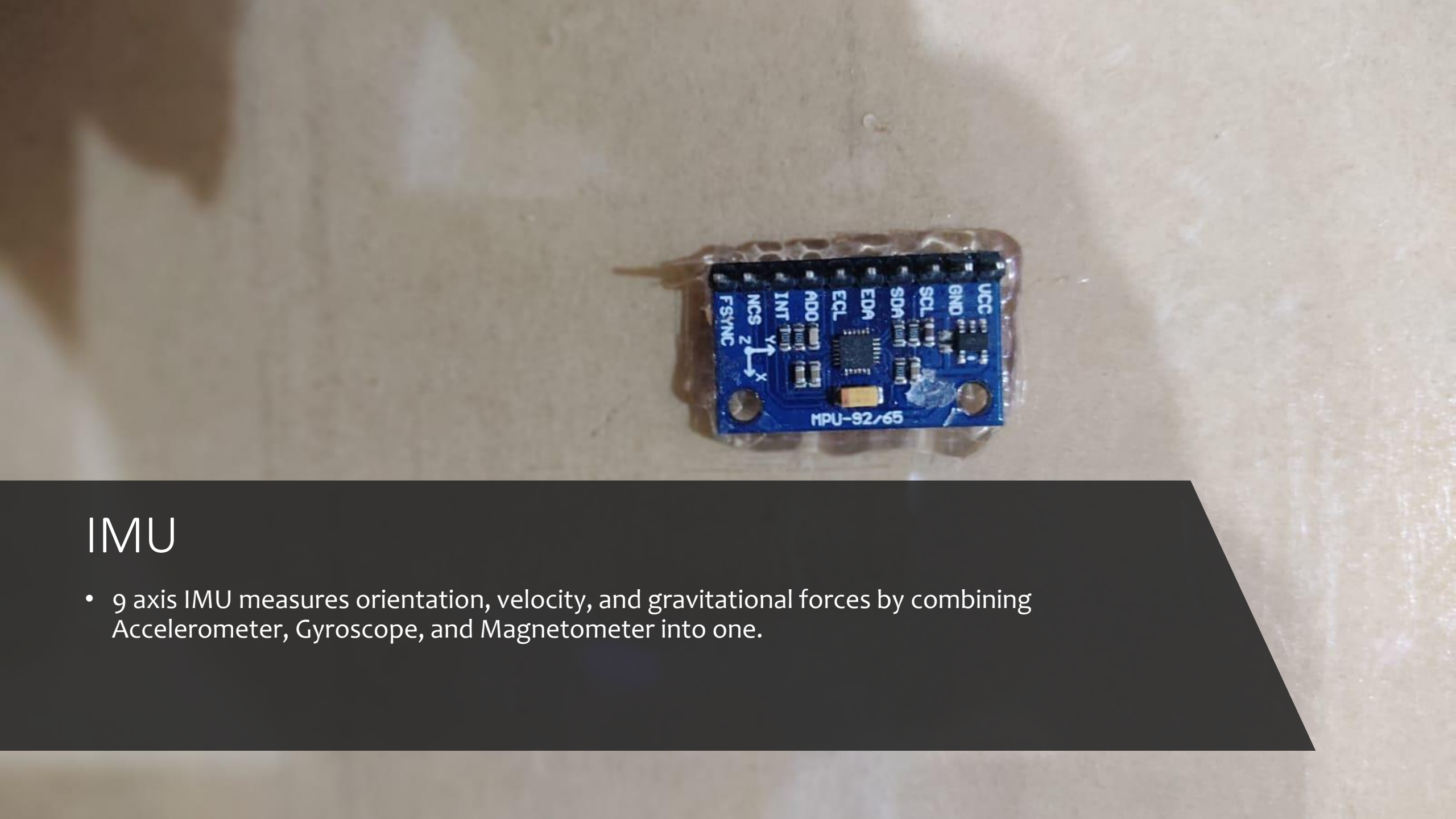
- Voltage capacity: 5V
- Output signal: 5V high and 0V low
- Sensor angle not more than 15 degree
- Sensor bounce back: 2cm to 450cm



Infrared sensor

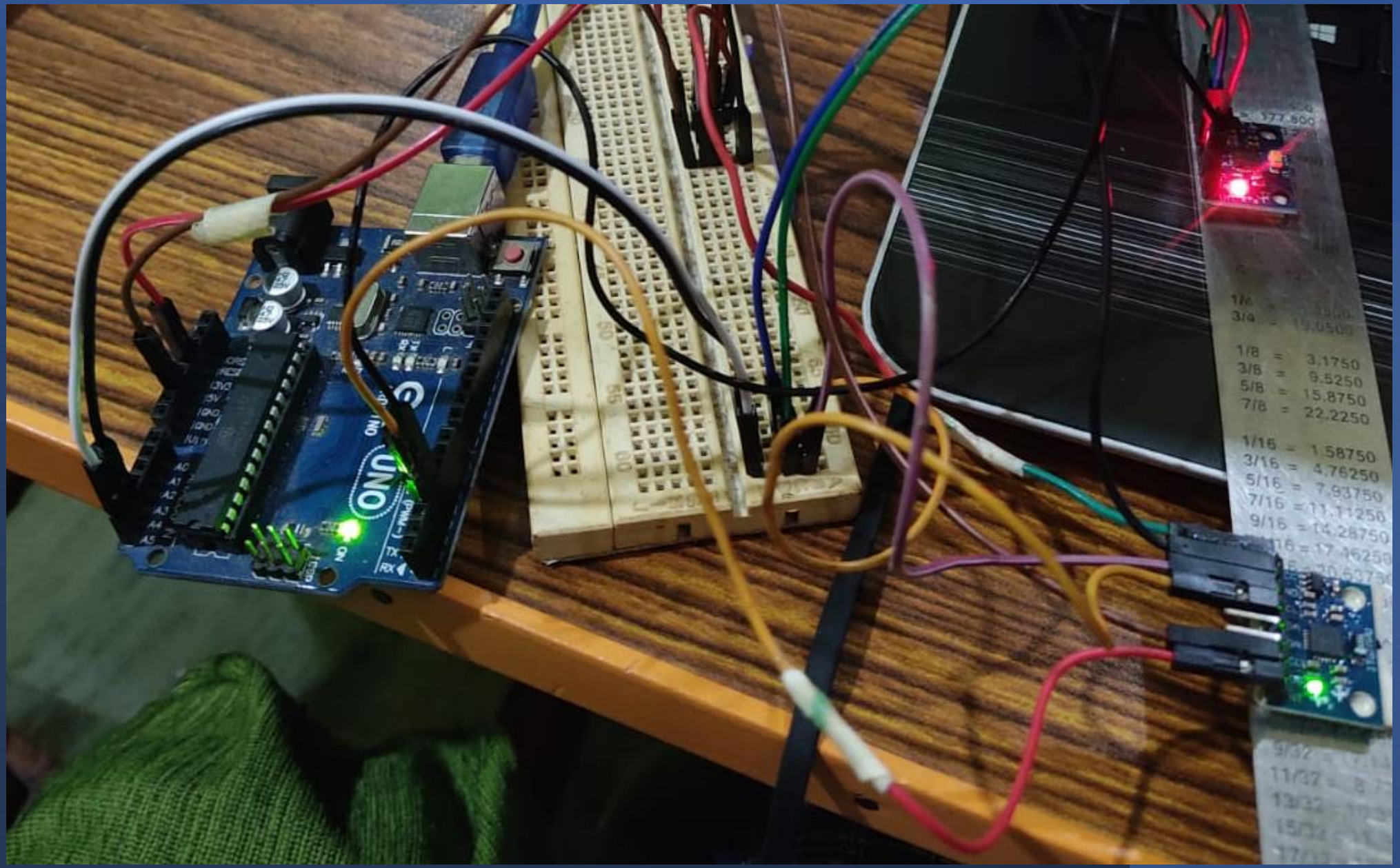
- Range : 10-15 cm|
- Input supply voltage: 5V DC

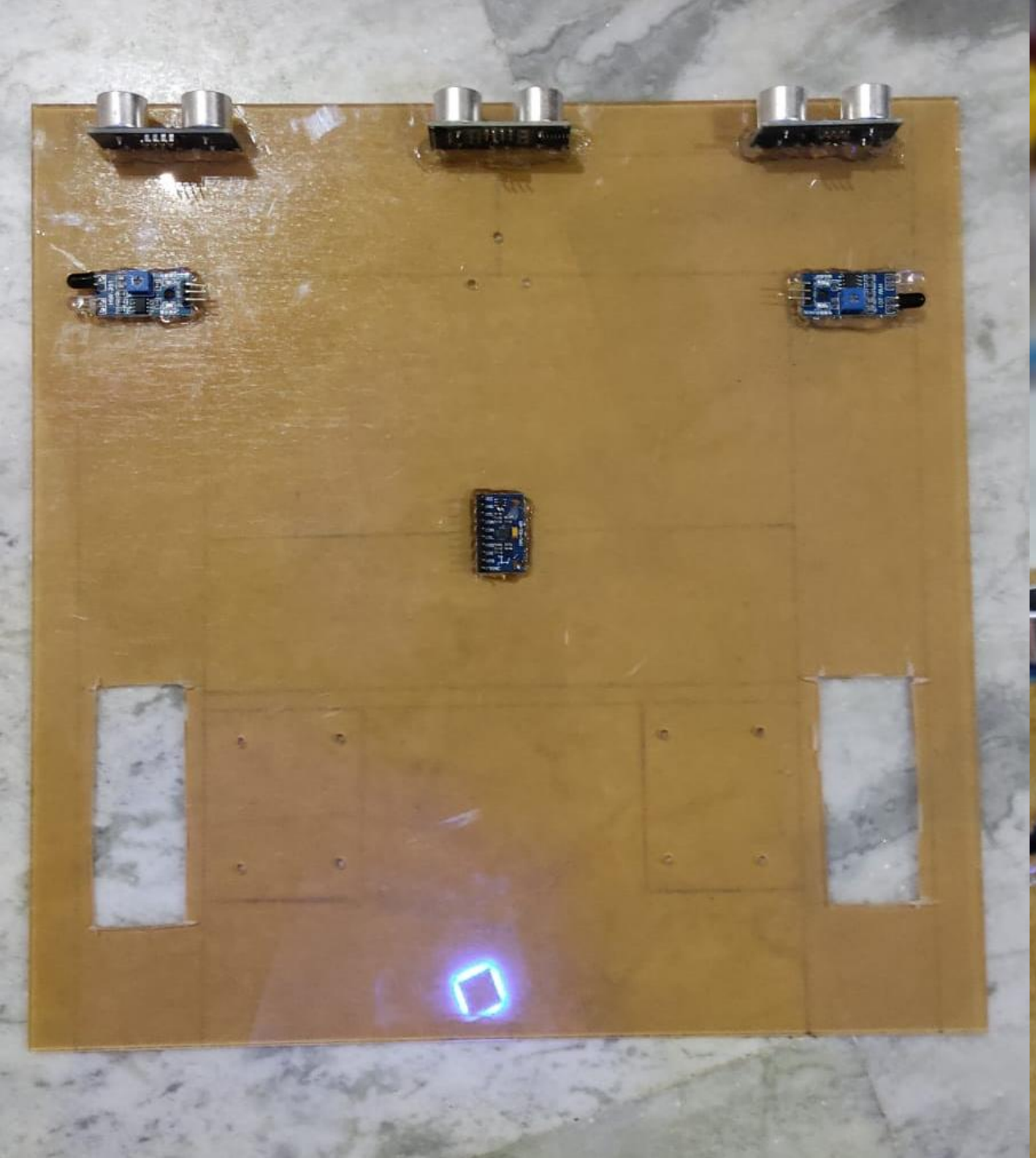

TW-201
OUT GND VCC

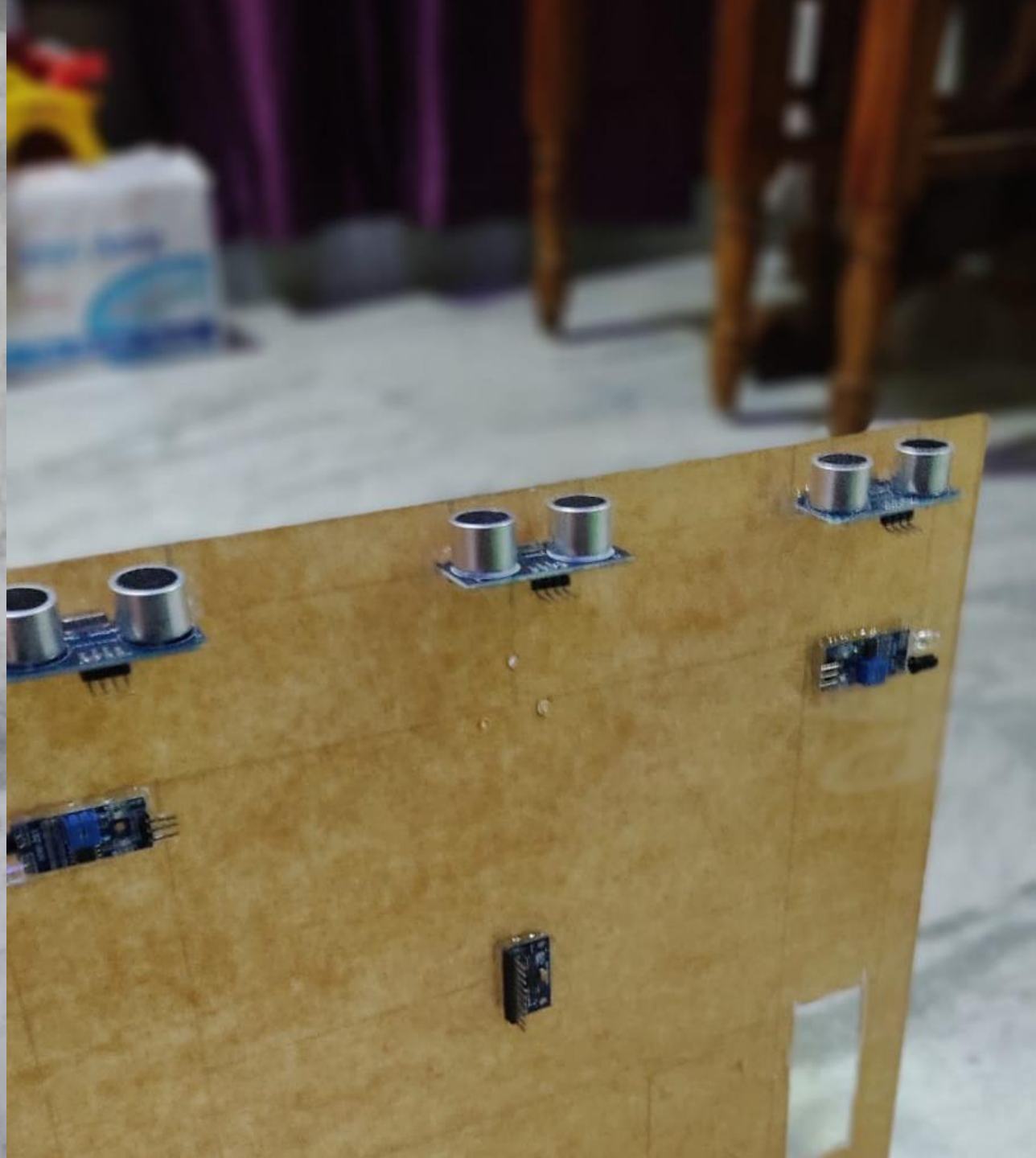


IMU

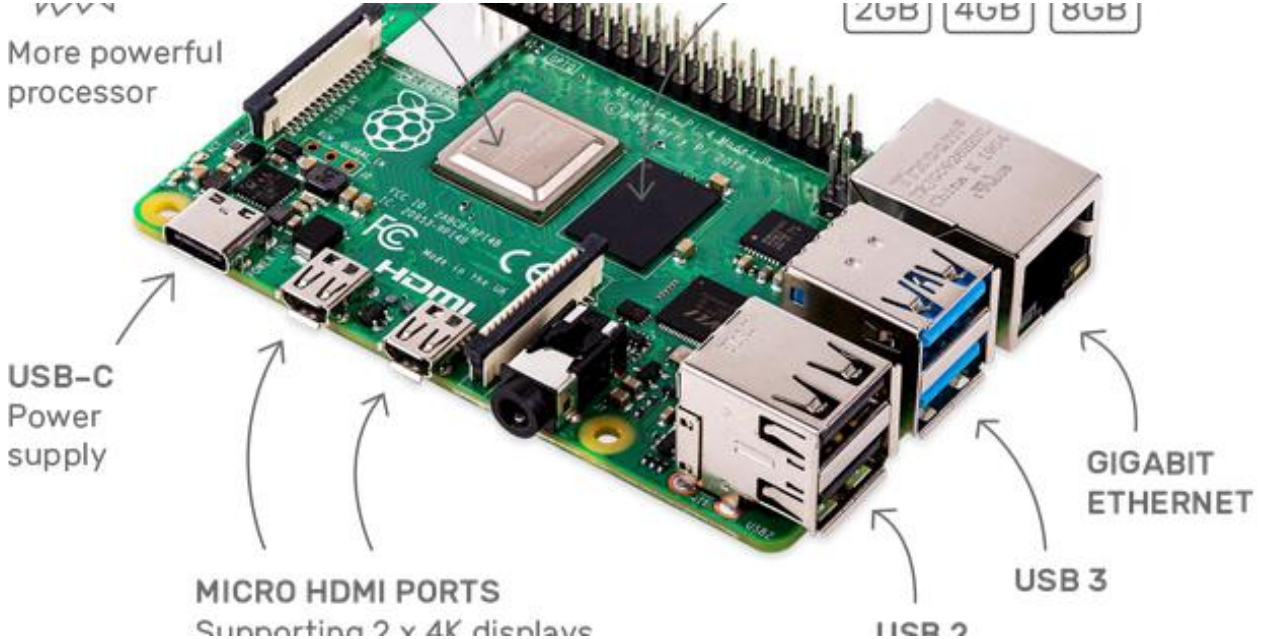
- 9 axis IMU measures orientation, velocity, and gravitational forces by combining Accelerometer, Gyroscope, and Magnetometer into one.


UNO
1/8 = 3.1750
3/8 = 9.5250
5/8 = 15.8750
7/8 = 22.2250
1/16 = 1.58750
3/16 = 4.76250
5/16 = 7.93750



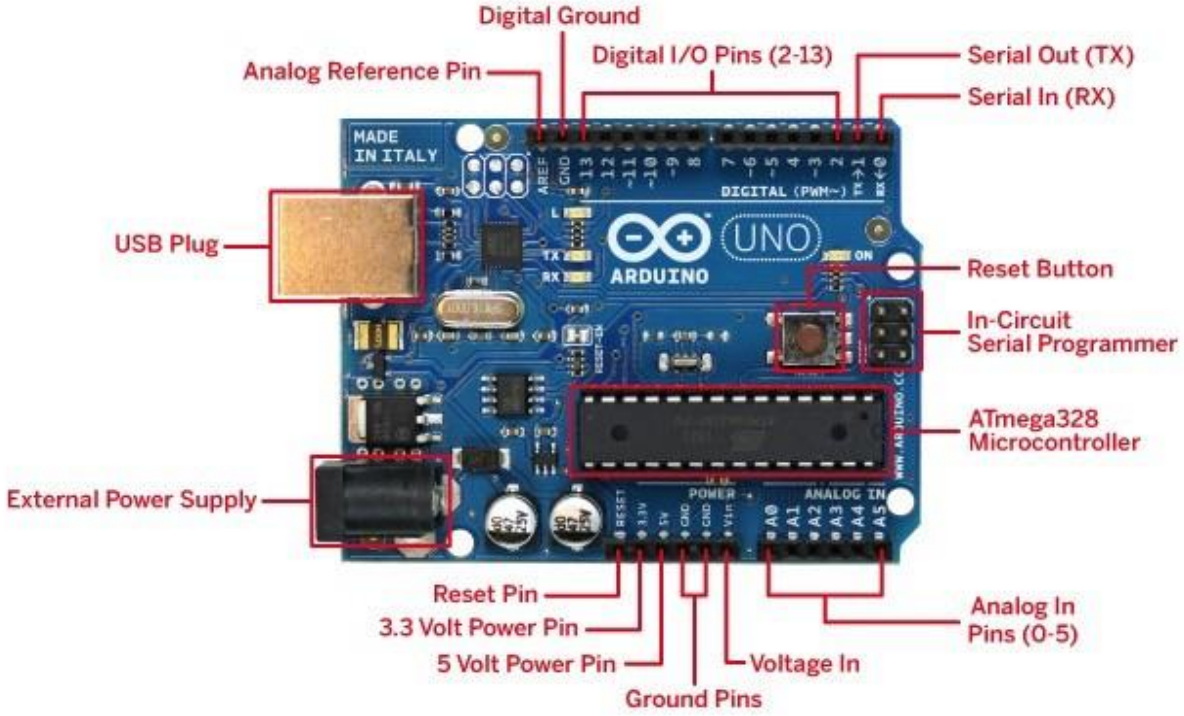



Controllers



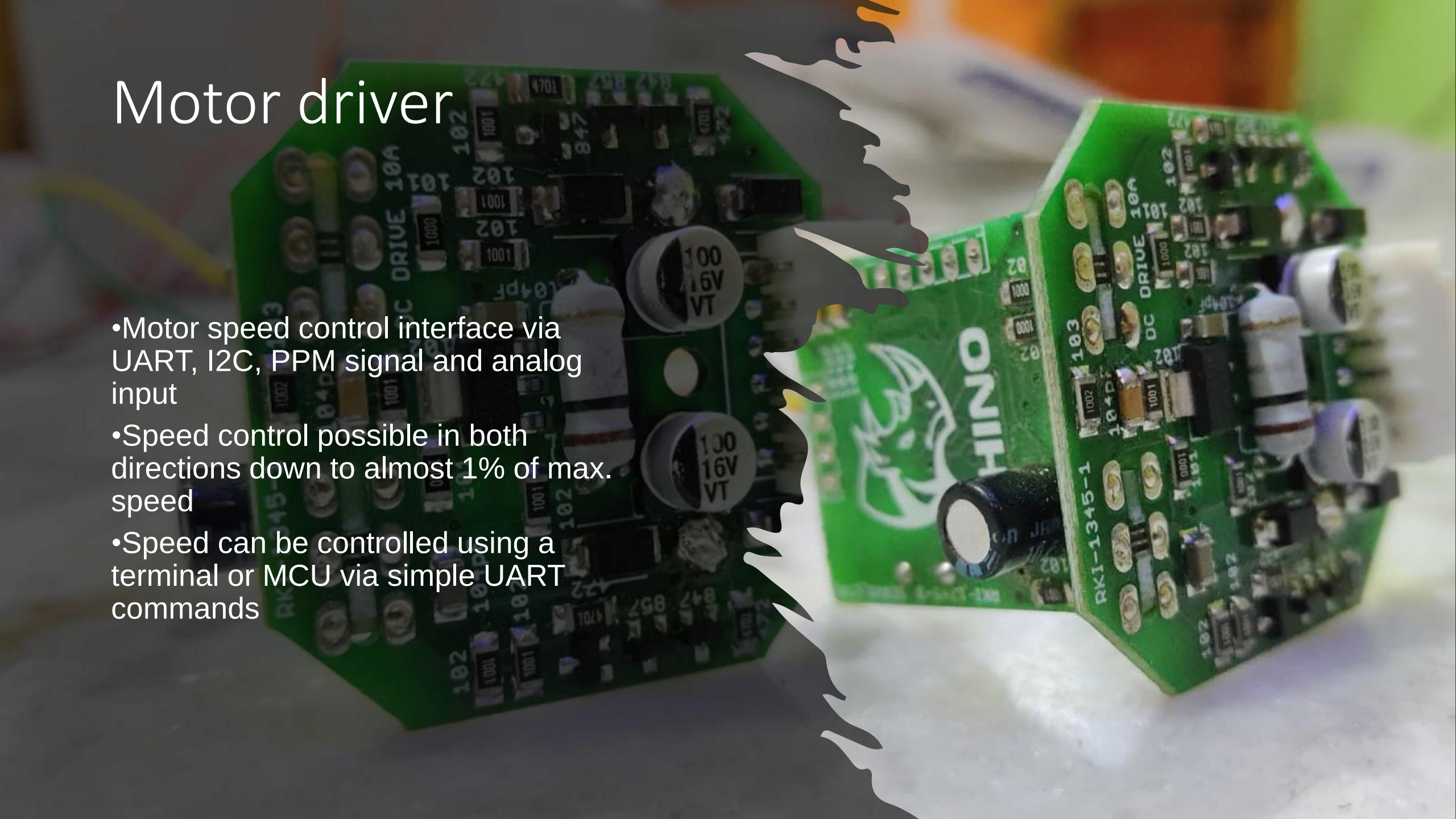
Raspberry pi model 4-B

Arduino UNO


Digital Ground
Analog Reference Pin
Digital I/O Pins (2-13)
Serial Out (TX)
Serial In (RX)
USB Plug
Reset Button
In-Circuit
Serial Programmer
ATmega328
Microcontroller
External Power Supply
Reset Pin
3.3 Volt Power Pin
5 Volt Power Pin
Ground Pins
Voltage In
Analog In
Pins (0-5)


UNO
ARDUINO
DIGITAL (PWM~)
POWER
ANALOG IN

Motor driver



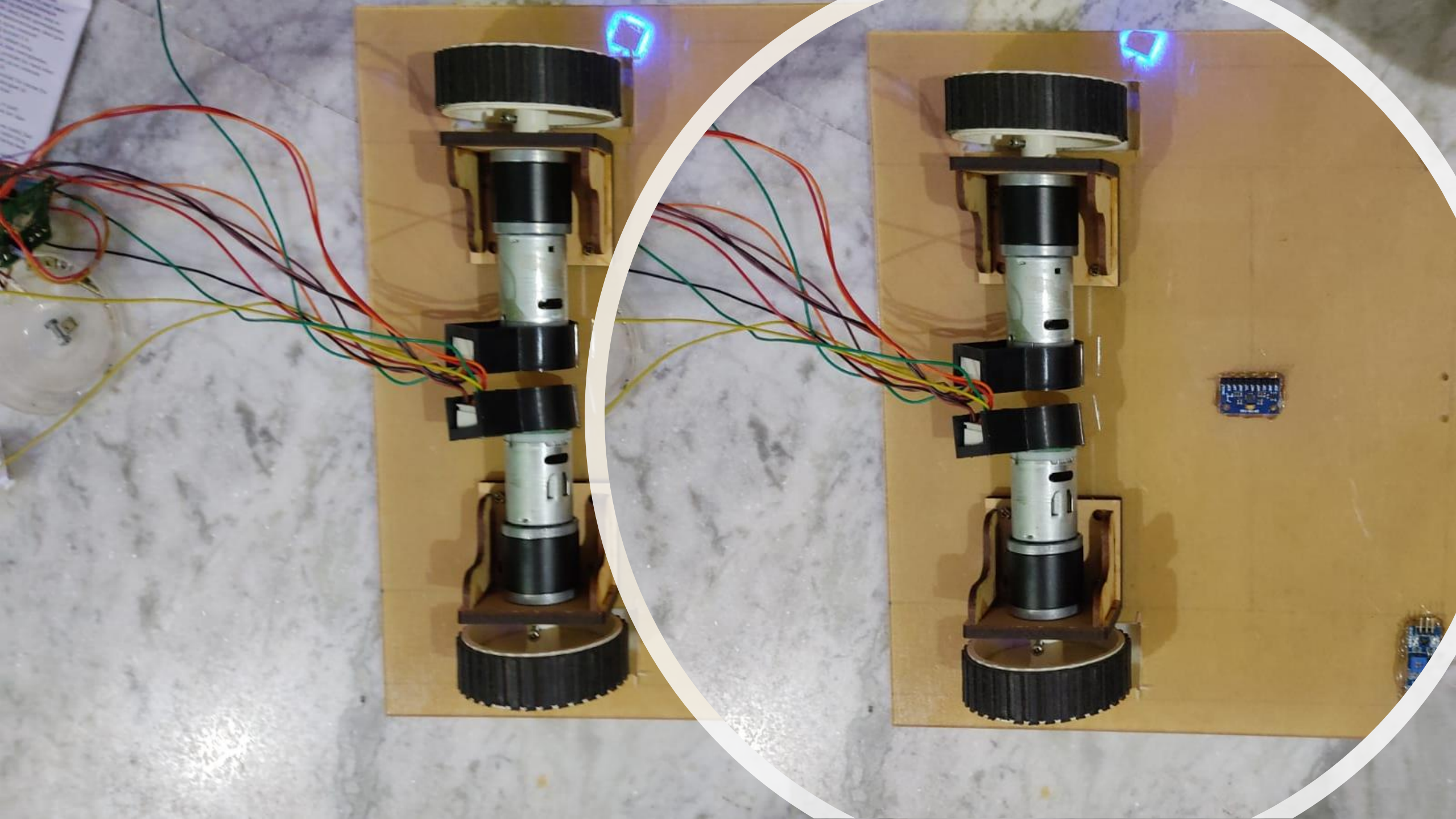
- Motor speed control interface via UART, I2C, PPM signal and analog input
- Speed control possible in both directions down to almost 1% of max. speed
- Speed can be controlled using a terminal or MCU via simple UART commands

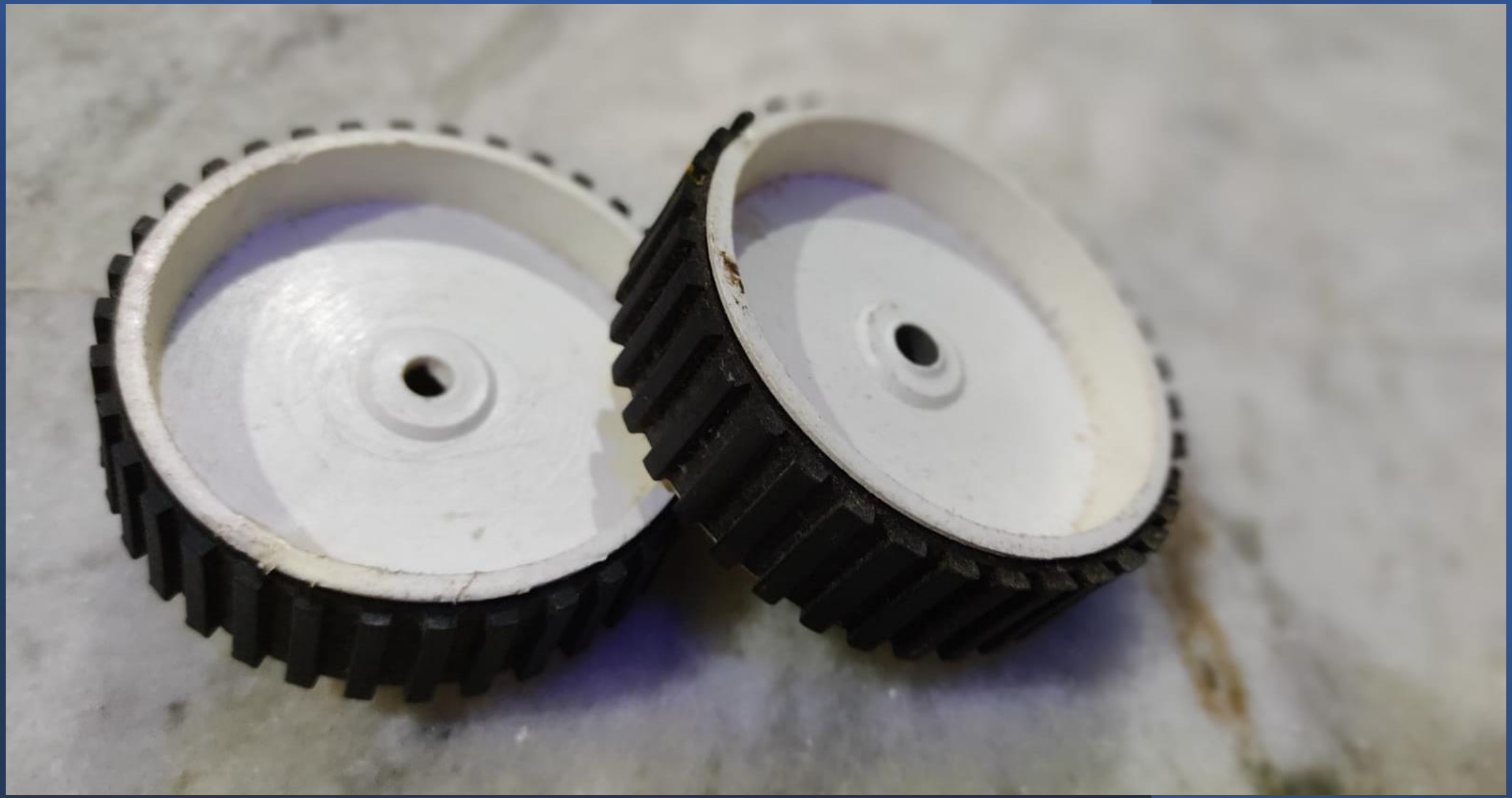
Motor

- 200 RPM 12V Rhino Planetary High Precision Encoder Geared Motor with Metal Planetary Motor High Precision Encoder and Metal Gears
- 18000 RPM base motor
- 3 stage gearbox for optimum high torque operation
- Motor rated Torque is 15kgcm along with gearbox and stall torque is 60kgcm, however it is recommended to use the motor at rated torque for optimum life and efficiency.
- Shaft is D type with total length of 16 mm and D shape in 12 mm.
- 6mm Dia shaft with M3 thread hole for tight mounting.
- Shaft can be coupled using CNC coupling 6 mm or using fixed coupling as per requirement.
- Back shaft length is 9 mm
- Gearbox diameter is 32 mm.
- Motor Diameter 28.5 mm
- Length 81.5 mm without shaft
- 300gm weight
- No-load current : 800 mA, Load current : upto 7.5 A(Max)

Encoder

- Counts per revolution (CPR) :1,33,600
- Red: EncB
- Orange: EncA
- Yellow: Motor-
- Green: Motor+
- Brown : Vcc +5 V DC
- Black : Gnd. *This is to be kept isolated from Motor Ground.





Encoder check

Need to observe the encoder data when motor spins..

- In psrt.ino the program is not reliable as the data keeps toggling from 0-32600- -32600-0 .
- so the problem is with volatile int as the encoder value is very large , so debugged by using volatile long.
- Along with it the atomic block library is also not giving result as expected , so removed it.

- Hence the final code is written in WATE.ino .

psrt.ino

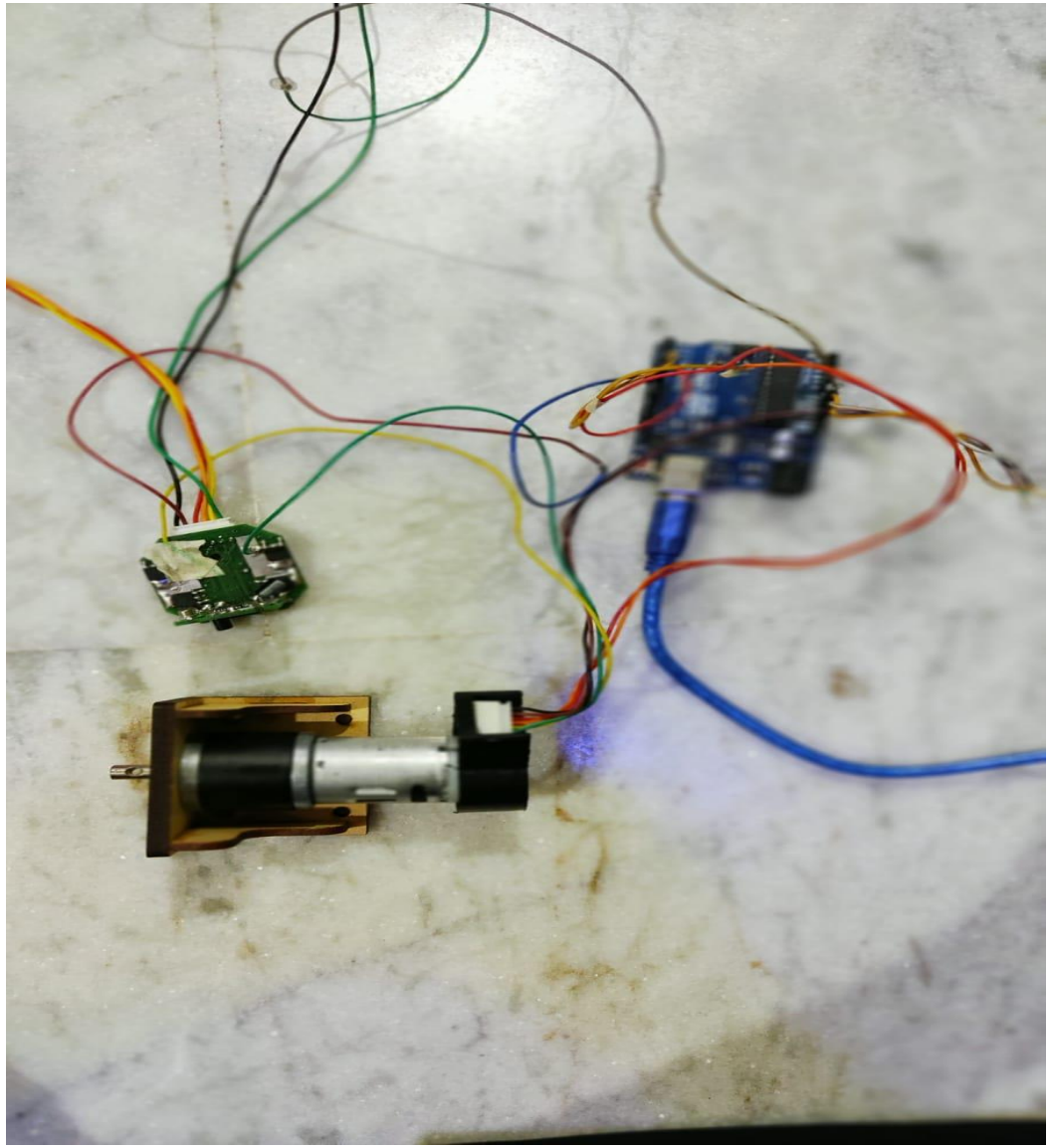
```
#include <util/atomic.h> // For the ATOMIC_BLOCK macro

#define ENCA 3// YELLOW

#define ENCB 8 // WHITE

Volatile signed  long posi = 0; // specify posi as volatile signed

void setup() {

Serial.begin(9600);

pinMode(ENCA,INPUT_PULLUP);

pinMode(ENCB,INPUT);

attachInterrupt(digitalPinToInterrupt(ENCA),readEncoder,RISING);

}

}
```

```
void loop() {
  // Read the position in an atomic block to avoid a potential
  // misread if the interrupt coincides with this code running
  // see: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/
  int pos = 0;
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    pos = posi;
  }
Serial.println(pos);
}
void readEncoder(){
  int b = digitalRead(ENCB);
  if(b > 0){
    posi=posi+100;
  }
  else{
   posi= posi-100;
  }
}
```

WATE.ino

```
const int A=3;
const int B=8;
volatile long tick =0; //this where the error is present //unsigned
 void setup(){
  Serial.begin(115200);
  pinMode(A,INPUT_PULLUP);
  pinMode(B,INPUT);

attachInterrupt(digitalPinToInterrupt(A),Funct,RISING);

}
 void loop(){
  Serial.println(tick);
  }

 void Funct(){
 int b = digitalRead(B);
  if(b== 1){
   tick =tick+100;
    }
  else if(b==0){
   tick=tick-100;

  }

}
```

Motor test with Arduino



Motor control with driver

```
//------------------------++++++++++++++++++++++++++++++++++++++++++++--
#define CHANNEL_NUMBER 1 //set the number of channel
#define CHANNEL_DEFAULT_VALUE 1500 //set the default value
#define FRAME_LENGTH 4800 //set the PPM frame length in microseconds (1ms = 1000µs)
#define PULSE_LENGTH 1660 //set the pulse length in microseconds (The PPM signal pulse width must range from 600us to 2.4ms. The motor speed will be zero at PPM pulse width of 1.51ms)
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is negative
#define sigPin 9  //set PPM signal output pin on the arduino
int ppm[CHANNEL_NUMBER];
//------------------------++++++++++++++++++++++++++++++++++++++++++++--
const int A=3;
const int B=8;
volatile long tick =0;// specify posi as volatile: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/
```

```
void setup(){
 //initiallize default ppm values
 for(int i=0; i<CHANNEL_NUMBER; i++){
    ppm[i]= CHANNEL_DEFAULT_VALUE;
 }
pinMode(sigPin, OUTPUT);
 digitalWrite(sigPin, !onState);  //set the PPM signal pin to the default state (off)

 cli();
 TCCR1A = 0; // set entire TCCR1 register to 0
 TCCR1B = 0;
 OCR1A = 100;  // compare match register, change this
 TCCR1B |= (1 << WGM12);  // turn on CTC mode
 TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
 TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
 sei();
 //------------------------++++++++++++++++++++++++++++++++++++++++++++----------
 Serial.begin(115200);
 pinMode(A,INPUT_PULLUP);
 pinMode(B,INPUT_PULLUP);
 attachInterrupt(digitalPinToInterrupt(A),Funct,RISING);
}
```

```
void loop(){
.println(tick);
}
ISR(TIMER1_COMPA_vect){  //leave this alone
static boolean state = true;
TCNT1 = 0;
if (state) {  //start pulse
digitalWrite(sigPin, onState);
OCR1A = PULSE_LENGTH * 2;
state = false;
} else{  //end pulse and calculate when to start the next pulse
static byte cur_chan_numb;
static unsigned int calc_rest;
digitalWrite(sigPin, !onState);
state = true;
```

```
if(cur_chan_numb >= CHANNEL_NUMBER){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PULSE_LENGTH;
      OCR1A = (FRAME_LENGTH - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PULSE_LENGTH) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}
void Funct(){
 int b = digitalRead(B);
  if(b== 1){
    tick =tick+1;
     }
  else if(b==0){
   tick=tick-1;
}
}
```

Control motor direction and position with driver and encoder..

- #define CHANNEL_NUMBER 1 //set the number of channel
- #define CHANNEL_DEFAULT_VALUE 1500 //set the default value
- #define FRAME_LENGTH 4800 //set the PPM frame length in microseconds (1ms = 1000µs)
- #define onState 1 //set polarity of the pulses: 1 is positive, 0 is negative
- #define sigPin 9 //set PPM signal output pin on the arduino
- int PULSE_LENGTH; //set the pulse length in microseconds (The PPM signal pulse width must range from 600us to 2.4ms. The motor speed will be zero at PPM pulse width of 1.51ms)
- int ppm[CHANNEL_NUMBER];

- //////////////////////
- const int A=3; //encoderA to pin3 and encoderB to pin8
- const int B=8;
- volatile long tick =0; //this where the bug is present //unsigned

- int Encoder_cpr ;
- int distance = ?;

Setup

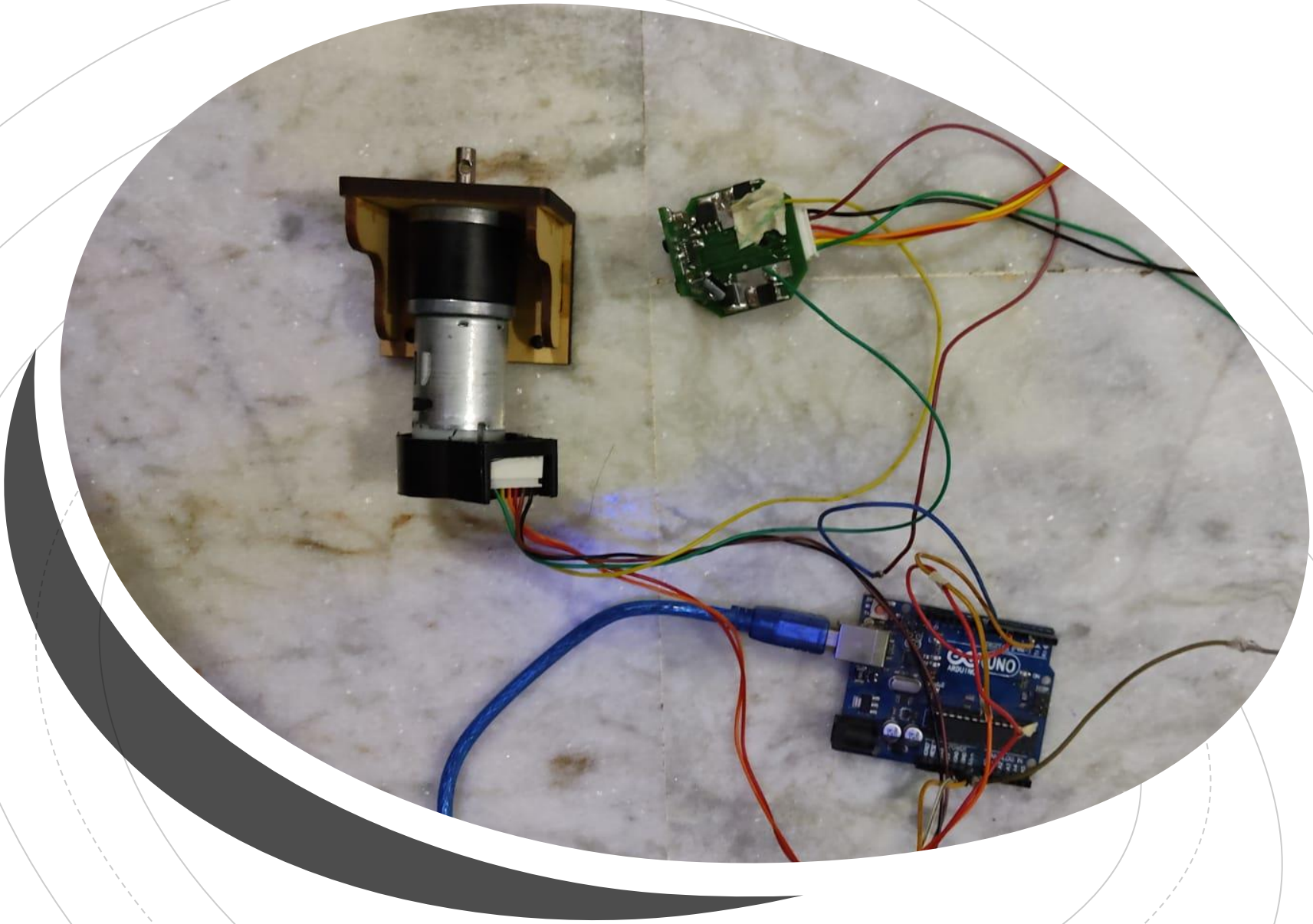
```
void setup(){
Serial.begin(115200);
//initiallize default ppm values
for(int i=0; i<CHANNEL_NUMBER; i++){
ppm[i]= CHANNEL_DEFAULT_VALUE;
}
pinMode(sigPin, OUTPUT);
digitalWrite(sigPin, !onState);  //set the PPM signal pin to the default state (off)
pinMode(A,INPUT_PULLUP);
pinMode(B,INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(A),Funct,RISING);
cli();
TCCR1A = 0; // set entire TCCR1 register to 0
TCCR1B = 0;
 OCR1A = 100;  // compare match register, change this
 TCCR1B |= (1 << WGM12);  // turn on CTC mode
 TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
 TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
 sei();
Encoder_cpr=((distance/44)*7*133600*radius); //wheel radius
}
void loop(){
 Serial.println(tick);
 }
```

Motor control

```
ISR(TIMER1_COMPA_vect){  //leave this alone
if( tick >= Encoder_cpr){
  PULSE_LENGTH = 1510 ;
 }

 static boolean state = true;

 TCNT1 = 0;

if (state) {  //start pulse
   digitalWrite(sigPin, onState);
   OCR1A = PULSE_LENGTH * 2;
   state = false;
 } else{  //end pulse and calculate when to start the next pulse
   static byte cur_chan_numb;

 static unsigned int calc_rest;

   digitalWrite(sigPin, !onState);
   state = true;
 if(cur_chan_numb >= CHANNEL_NUMBER){
     cur_chan_numb = 0;
     calc_rest = calc_rest + PULSE_LENGTH;
     OCR1A = (FRAME_LENGTH - calc_rest) * 2;
     calc_rest = 0;
   }
   else{
     OCR1A = (ppm[cur_chan_numb] - PULSE_LENGTH) * 2;
     calc_rest = calc_rest + ppm[cur_chan_numb];
     cur_chan_numb++;
   }
 }
}
```

Position check from encoder (feedback)

```
void Funct(){
int b = digitalRead(B);
if(b== 1){
tick =tick+100;
}
else if(b==0){
tick=tick-100;
}
}
```



Tested

Speed ✓

Position ✓

Test product -1.0