1928
K. N. Toosi University
of Technology

Instrumentation Final Project

The Use of an IMU Sensor and Image Processing Techniques for the Measurement of Steering Angle

Group Members:

Arman Javan Sekhavat
Amin Elmi Ghiasi
Mohammad Maleki Abyaneh

Table of Contents

Introduction

In this project, an ESP32 CAM Dev board is held stationary with respect to the body of a car and it is used to capture images from the different postures of its steering wheel. The rotation angle corresponding to each posture is measured using an MPU6050 IMU sensor, and these measurements are stored in a CSV file.

A CNN model is designed and trained using the aforementioned images as the input and the rotation angles as the labels for these images. After training, this model is able to predict the rotation angle of the steering wheel from an unseen input image.

Finally, the steering angle measurements of the CNN model and the IMU sensor are compared to each other.

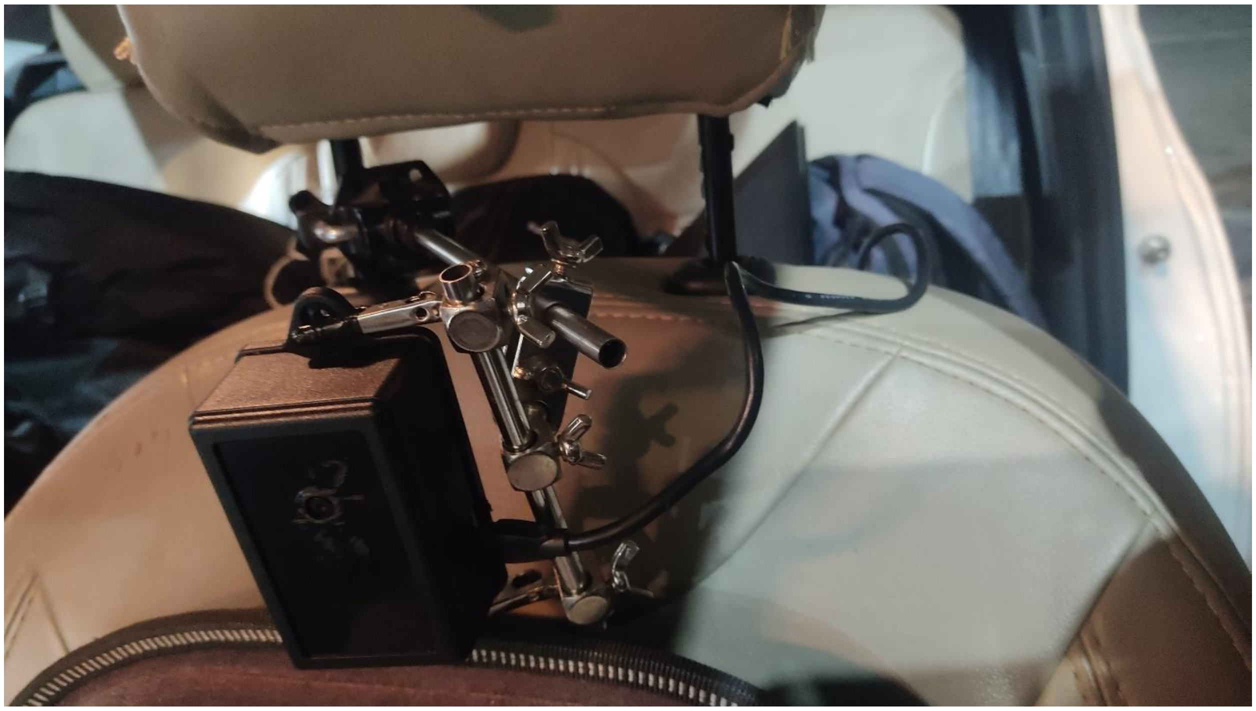


Fig. 1. Camera setup

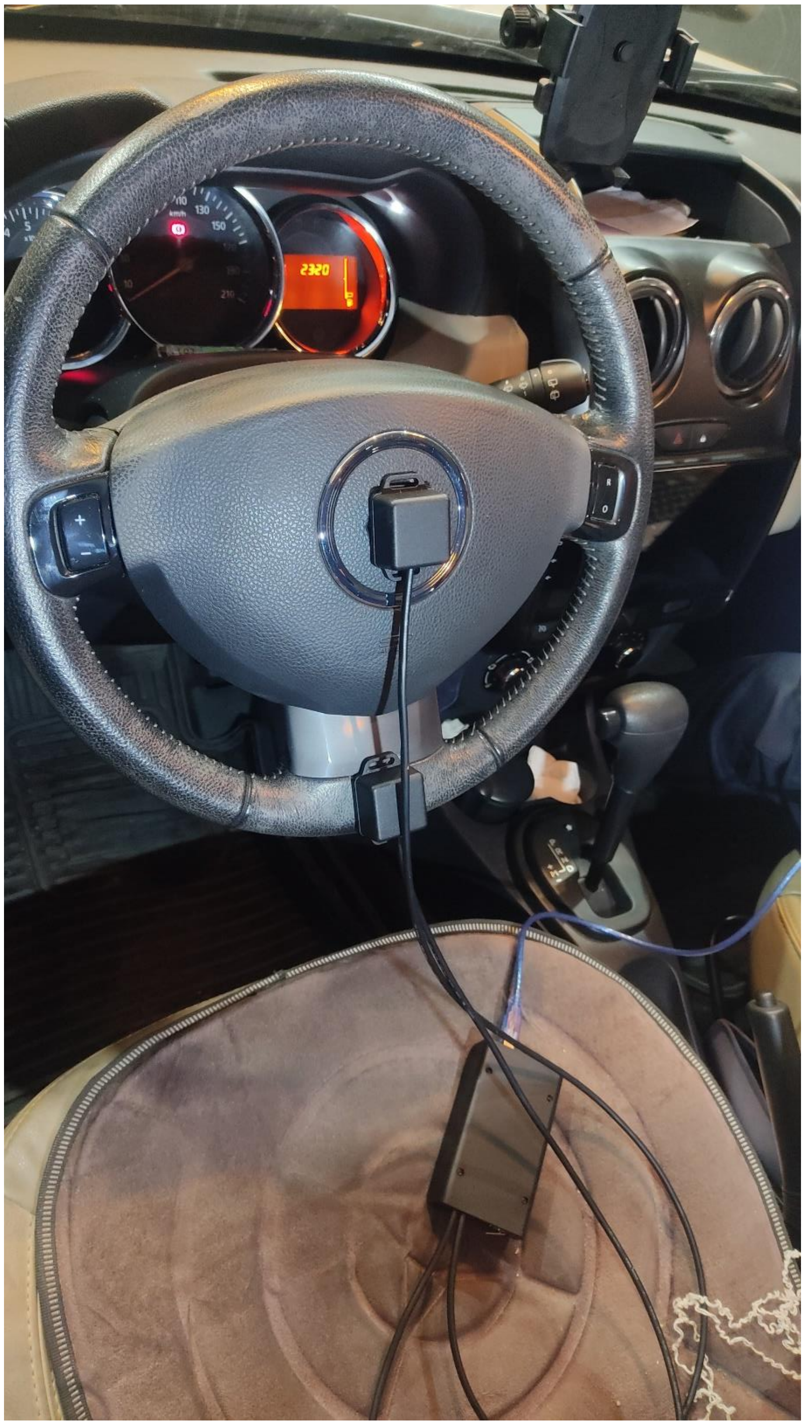


Fig. 2. IMU sensors setup

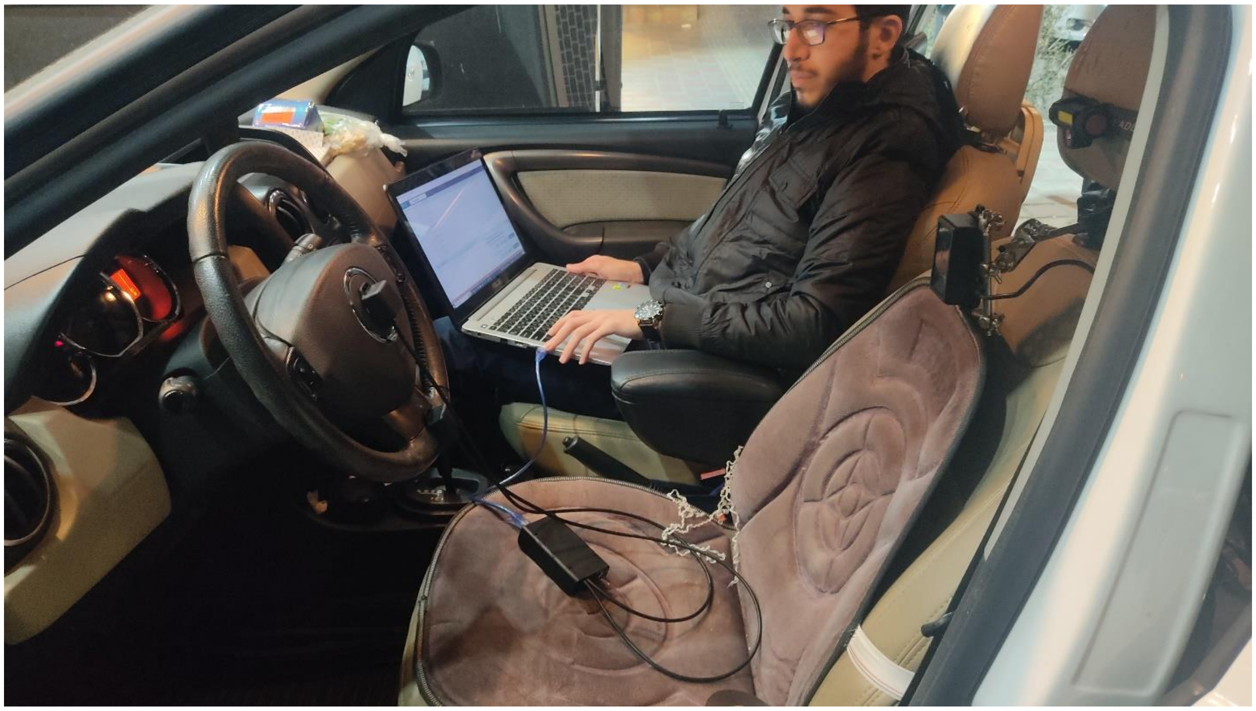


Fig. 3. Data collection process

Image Processing Technique

The image processing technique used in this project utilizes a CNN (Convolutional Neural Network), which is implemented in a Python program, using the Keras library. The input tensor to this neural network is a 3-channel color RGB image with its width and height equal to 800 and 600, respectively. The output tensor is a 2-dimensional vector in the form $[\cos\theta\ ,\ \sin\theta]^T$, where θ is the steering angle. The logic behind this choice is discussed later in this report.

Gathering Data

In order to train the aforementioned CNN, a large number of images and the corresponding steering angles is required. The images are taken from the different postures of the steering wheel, and the corresponding angle is measured using the IMU sensor. The measured angles are obtained through a CSV file, which contains the measurements and their corresponding times. Note that the times are expressed relative to the time when the measurement had been started.

But we had to find a way to match each image with its corresponding steering angle. We achieved this by using two switches. One of the switches is connected to our ESP32 CAM board and the other to an Arduino UNO board. When the switch is triggered, the reference time is set and after that, all of the times are measured with respect to this reference time. Note that the switches are triggered simultaneously. Suppose an image is to be matched with its corresponding steering angle; The approach below is followed:

1) The time when the images are captured is considered. This time is measured by the ESP32 CAM board.
2) A search is performed inside the CSV file to find the nearest time to the time obtained from step 1. Then, the corresponding steering angle will be the angle we were searching for.

The images and the corresponding capture times are sent to a computer through a USB cable. A Python program is then run on the computer whose purpose is to receive the sent data and store them on the hard drive of the computer. The received images are stored as JPG files in a dedicated folder named “images” and the received times are stored in a text file named “times.txt”. Here are the codes which are used for the capturing and transmission of images for the ESP32 CAM board and the Python program:

receiver.py

```
import serial

s = serial.Serial()
s.port = 'COM5'
s.baudrate = 460800
s.close()
s.open()
n = 0

textFile = open("times.txt", 'wt')

msg = bytes([100])

s.write(msg)


while True:
    # receiving the time
    time = s.read(8)
    time = int.from_bytes(time, byteorder = 'little')

    # receiving the image size
    imgBytes = s.read(2)
    imgBytes = int.from_bytes(imgBytes, byteorder = 'little')

    # receiving the image content
    buf = s.read(imgBytes)
    textFile.write(str(time) + '\n')


    F = open("C:/Users/User/Desktop/images/" + str(n) + ".jpg", 'wb')
    F.write(buf)
    F.close()


    n += 1

textFile.close()
```

Important notes:

1) The ‘’pySerial’’ library is used to establish the serial communication between the ESP32 CAM board and the aforementioned Python program.
2) The maximum functioning standard baud rate was found to be 460800 for this communication channel.
3) The times are expressed in milliseconds and 8-byte integers are used to represent them.

sender.cpp

```
#include "esp_camera.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "driver/rtc_io.h"
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <Arduino.h>
```

```
/*
  1) Connect the USB cable
  2) Activate the switch
  3) Run the Python script
*/


// ----------------------------------  Pin definition for camera
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22


#define baud_rate 460800
const int switchPin = 13;

bool activated = false;
bool stat = false;

typedef unsigned short USHORT;
typedef unsigned long long  ULL;

USHORT imgBytes = 0;
char* fb_buf = NULL;
camera_fb_t* fb = NULL;
```

```
ULL startTime = 0;
ULL passedTime = 0;
ULL t = 0;

char x = 0;

void capture(void* pvParameters){

    while(true){

        if(!activated){
            if( digitalRead(switchPin) == LOW ){
                startTime = millis();
                activated = true;
            }

        }else{

            if(!stat){
                Serial.read(&x, 1);

                if(x == 100){
                    stat = true;
                }
            }else{

            fb = esp_camera_fb_get();
            passedTime = millis();
            t = passedTime - startTime;
            imgBytes = fb->len;
            fb_buf = (char*) (fb->buf);

            // sending the time
            Serial.write((const char*) &t, 8);

            // sending the image size
            Serial.write((const char*) &imgBytes, 2);
            // sending the image content
            Serial.write((const char*) fb_buf, imgBytes);


            esp_camera_fb_return(fb);

            vTaskDelay(100 / portTICK_PERIOD_MS);
            }

        }


    }


}


void setup(){

    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

    // -----------------------------------  Camera configuration
    camera_config_t config;

    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;
    config.pin_d7 = Y9_GPIO_NUM;
    config.pin_xclk = XCLK_GPIO_NUM;
    config.pin_pclk = PCLK_GPIO_NUM;
    config.pin_vsync = VSYNC_GPIO_NUM;
    config.pin_href = HREF_GPIO_NUM;
    config.pin_sscb_sda = SIOD_GPIO_NUM;
    config.pin_sscb_scl = SIOC_GPIO_NUM;
    config.pin_pwdn = PWDN_GPIO_NUM;
    config.pin_reset = RESET_GPIO_NUM;
    config.xclk_freq_hz = 30000000;
    config.pixel_format = PIXFORMAT_JPEG;
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 8;
    config.fb_count = 1;

    esp_err_t err = esp_camera_init(&config);

    pinMode(switchPin, INPUT);

    Serial.begin(baud_rate);

    xTaskCreate(capture, "capture", 10000, NULL, 1, NULL);

}


void loop(){

}
```

Important notes:

1) The Serial.write() function is used to send the data from the ESP32 CAM board to the Python program.
2) Before sending each image, its size is sent to the Python program, because sizes of the images are variable and the Python program needs to know how many bytes it should receive for each image.
3) In order to prevent the corruption of the sent images, multithreading must be implemented. Here, the FreeRTOS library is used for multithreading.
4) The times are expressed in milliseconds and 8-byte integers are used to represent them. In C++, the unsigned long long data type provides 8-byte integers.

5) GPIO 13 is connected to the switch.

The following diagram illustrates how the switch is connected to our ESP32 CAM board:

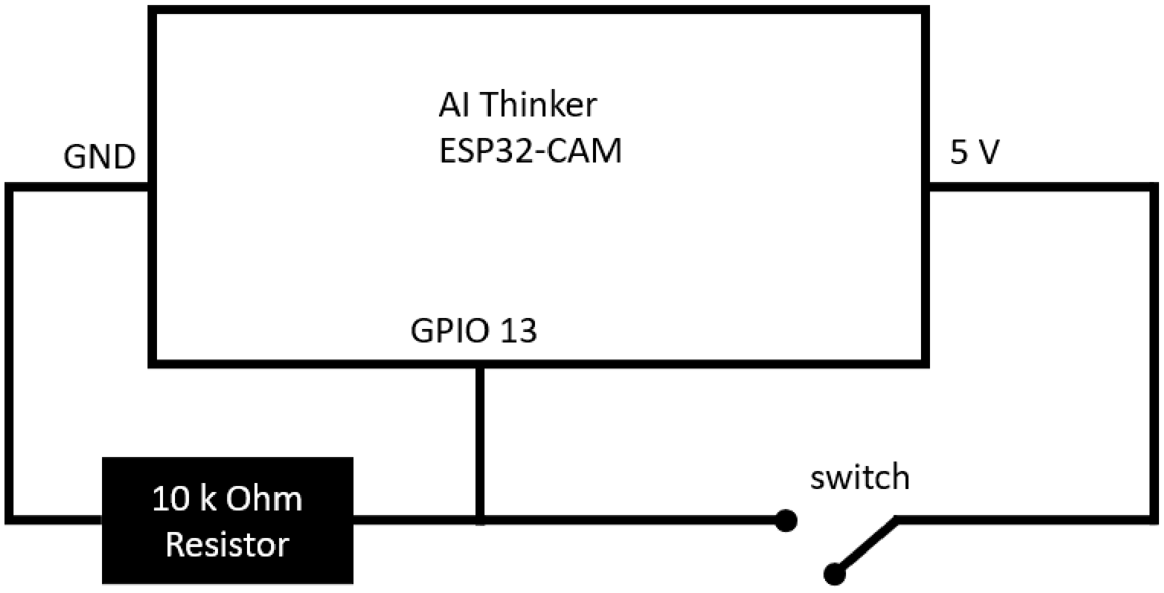


Fig. 4. Switch wiring

Training

In this phase, the first step is to load all of the captured images. Then all of these images are normalized. The images and their corresponding steering angles are matched together in this phase. The IMU sensor measurements are obtained from the CSV file using the Pandas library. Note that 90 percent of the data is used for training and the remaining 10 percent is used for validation.

A CNN model is designed and trained using the preprocessed data. MSE (Mean Squared Error) is used as the loss function and Adam is used as the optimizer for the training process. Then the loss function value and the mean absolute percentage error are plotted versus the epoch number. Since the elements of the output vector are in the interval (-1, +1), the hyperbolic tangent function can be a good choice for the activation function of the output layer. Finally, the best weights are stored in a file named ‘model’ for further uses. The following is the Python code for the implementation of the training phase:

training.py

```
import numpy as np
import cv2 as cv
import pandas as pd
import keras
from keras import Input
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from keras.optimizers import Adam
from keras.losses import MSE
from matplotlib import pyplot as plt
from math import cos
from math import sin
from math import pi
import math
from math import radians


inp = []
out = []

# -------------------------------------------   number of captured images
num = 931

# -------------------------------------------   reading the 'times.txt' file
# -------------------------------------------   times are expressed in milliseconds
file = open('times.txt', 'rt')
times = []

for line in file:
    times.append(int(line.strip()))


# -------------------------------------------   steering angles in degrees
# -------------------------------------------   times are expressed in milliseconds
# -------------------------------------------   'T' denotes the label for the times column
# --------------------------------------------   'A' denotes the label for the angles column
df = pd.read_csv('IMU.csv')
encoder_times  = list(df['T'])
encoder_angles = list(df['A'])




for i in range(num):
    img = cv.imread('images/' + str(i) + '.jpg', cv.IMREAD_COLOR)
    inp.append(img/255.0)

    time = times[i]

    diff = [abs(x - time) for x in encoder_times]
    diff = np.array(diff)
    j = np.argmin(diff)

    theta = radians(encoder_angles[j])
    vector = np.array([cos(theta), sin(theta)])
    out.append(vector)



inp = np.array(inp)
out = np.array(out)


#------------------------------------------------------  Model Checkpoint
checkpoint_filepath = 'model'
model_checkpoint_callback = keras.callbacks.ModelCheckpoint(
    filepath = checkpoint_filepath,
    monitor = 'val_loss',
    save_best_only = True,
    save_weights_only = True,
    mode = 'min')

#------------------------------------------------------  CNN model
input = Input(shape = (600, 800, 3))
```

```
layer1 = Conv2D(filters = 1, kernel_size = (5, 5), strides = (5, 5),
                padding = 'valid', activation = 'relu', use_bias = True)

layer2 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid')

layer3 = Conv2D(filters = 1, kernel_size = (3, 3), strides = (1, 1),
                padding = 'valid', activation = 'relu', use_bias = True)

layer4 = MaxPooling2D(pool_size = (2, 2), strides = (1, 1), padding = 'valid')

layer5 = Flatten()

layer6 = Dense(units = 10, activation = 'relu', use_bias = True)
layer7 = Dense(units = 2, activation = 'tanh', use_bias = True)

model = keras.Sequential( [input, layer1, layer2, layer3, layer4, layer5, layer6, layer7] )

model.summary()

model.compile(optimizer = Adam(learning_rate = 0.01), loss = MSE, metrics = ['mae'])
results = model.fit(x = inp, y = out, batch_size = 10, epochs = 50,
                    validation_split = 0.1, shuffle = True,
                     callbacks = [model_checkpoint_callback])


plt.figure()
plt.title('Train and Validation Set Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()
```

So why we used the vector $[\cos\theta\,,\ \sin\theta]^T$ as the output tensor of the CNN? Because if we chose the scalar θ as the output tensor, the CNN wouldn't learn properly, it would overfit and it would become extremely sensitive to the noises in the input image. The reason is explained below.

Suppose two input images, one corresponding to the angle 0° and the other to 359°. It's evident that these two images are very similar. Also, we know that the output of this CNN is a continuous function of its input, since all of its activation functions are continuous. Therefore, the close similarity in these input images must result in close output values, but that's not the case! They differ from each other by 359° ! Therefore, we concluded that it would be impossible to train a CNN in this way. We needed the output of the CNN to be a continuous and periodic function $F(\theta)$ such that $F(0^\circ) = F(360^\circ)$. The sine and cosine functions were good candidates for this purpose, but they had another important problem. These functions were not one-to-one in the interval $[0^\circ,\ 360^\circ]$. But we found that the vector-valued function $[\cos\theta\,,\ \sin\theta]^T$ is a great solution. It's continuous, periodic and invertible in that interval!

We first used the image of a steering wheel to generate a dataset by rotating the steering wheel around its center point for angles ranging from 0° to 360°. This dataset was composed of 1440 400x400 images of a steering wheel. The following code was used for training:

```
import numpy as np
import cv2 as cv
import keras
from keras import Input
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from keras.optimizers import Adam
from keras.losses import MSE
from matplotlib import pyplot as plt
from math import cos
from math import sin
from math import pi

inp = []
out = []

for i in range(1440):
   img = cv.imread('/kaggle/input/measurement-systems/' + str(i) + '.jpg', cv.IMREAD_COLOR)
   inp.append(img/255.0)
```

```
    theta = (i/1440)*2*pi
    vector = np.array([cos(theta), sin(theta)])
    out.append(vector)



inp = np.array(inp)
out = np.array(out)
input = Input(shape = (400, 400, 3))

layer1 = Conv2D(filters = 1, kernel_size = (5, 5), strides = (5, 5), padding = 'valid', activation =
'relu', use_bias = True)
layer2 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid')

layer3 = Conv2D(filters = 1, kernel_size = (3, 3), strides = (1, 1), padding = 'valid', activation =
'relu', use_bias = True)
layer4 = MaxPooling2D(pool_size = (2, 2), strides = (1, 1), padding = 'valid')

layer5 = Flatten()

layer6 = Dense(units = 3, activation = 'relu', use_bias = True)
layer7 = Dense(units = 2, activation = 'tanh', use_bias = True)

model = keras.Sequential( [input, layer1, layer2, layer3, layer4, layer5, layer6, layer7] )

model.summary()

model.compile(optimizer = Adam(learning_rate = 0.01), loss = MSE, metrics = ['mae'])
results = model.fit(x = inp, y = out, batch_size = 10, epochs = 200, validation_split = 0.1, shuffle
= True)

plt.figure()
plt.title('Train and Validation Set Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()
```

The results showed that this model has a good learning capacity for this specific kind of dataset.

The figure below shows the variation of the training and validation losses versus the epoch number.

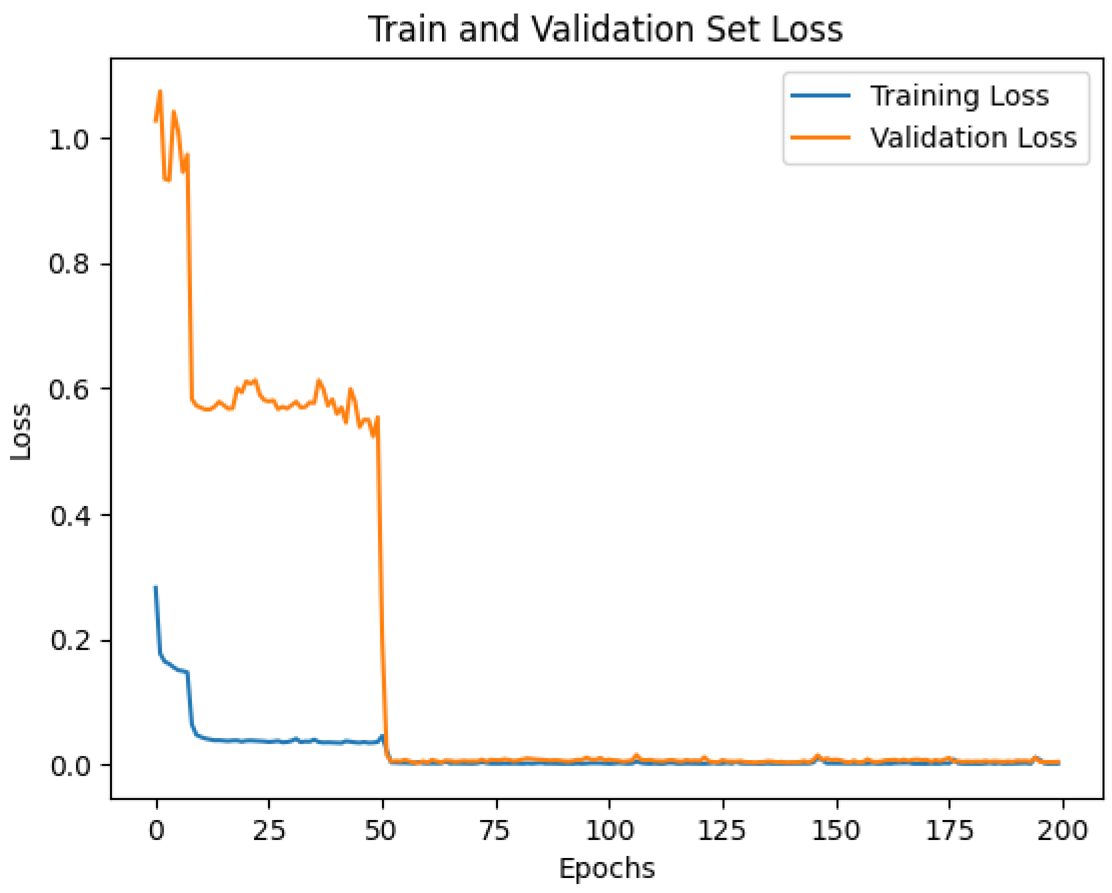


Fig. 5. Train and validation set loss versus the epoch number for the generated dataset

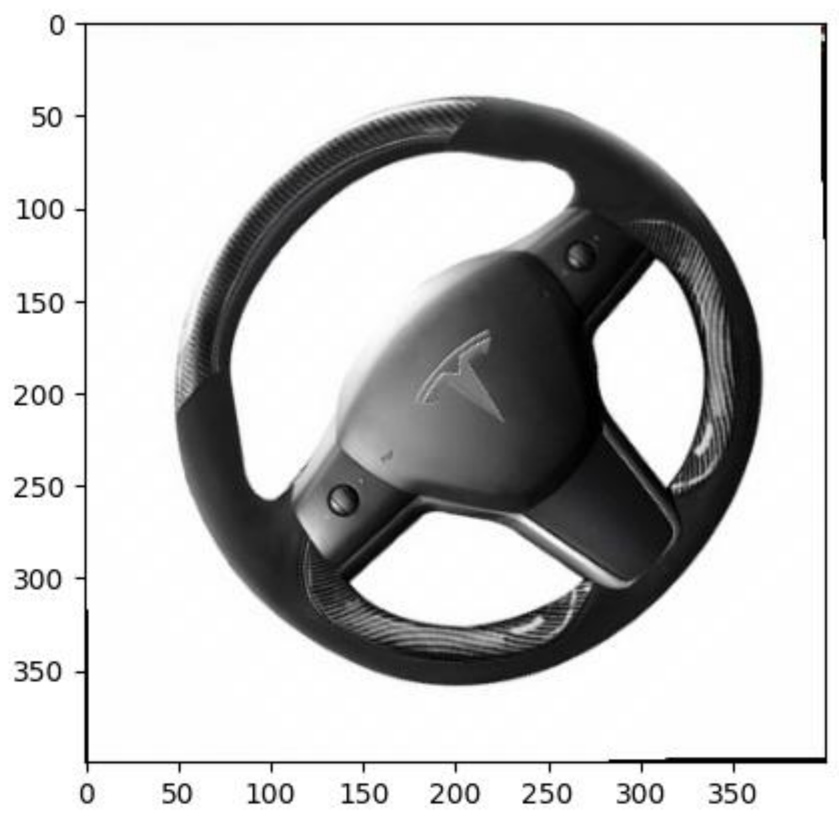


Fig. 6. Image of the aforementioned steering wheel corresponding to the angle 45 degrees. The prediction of the CNN model is 46.33 degrees.

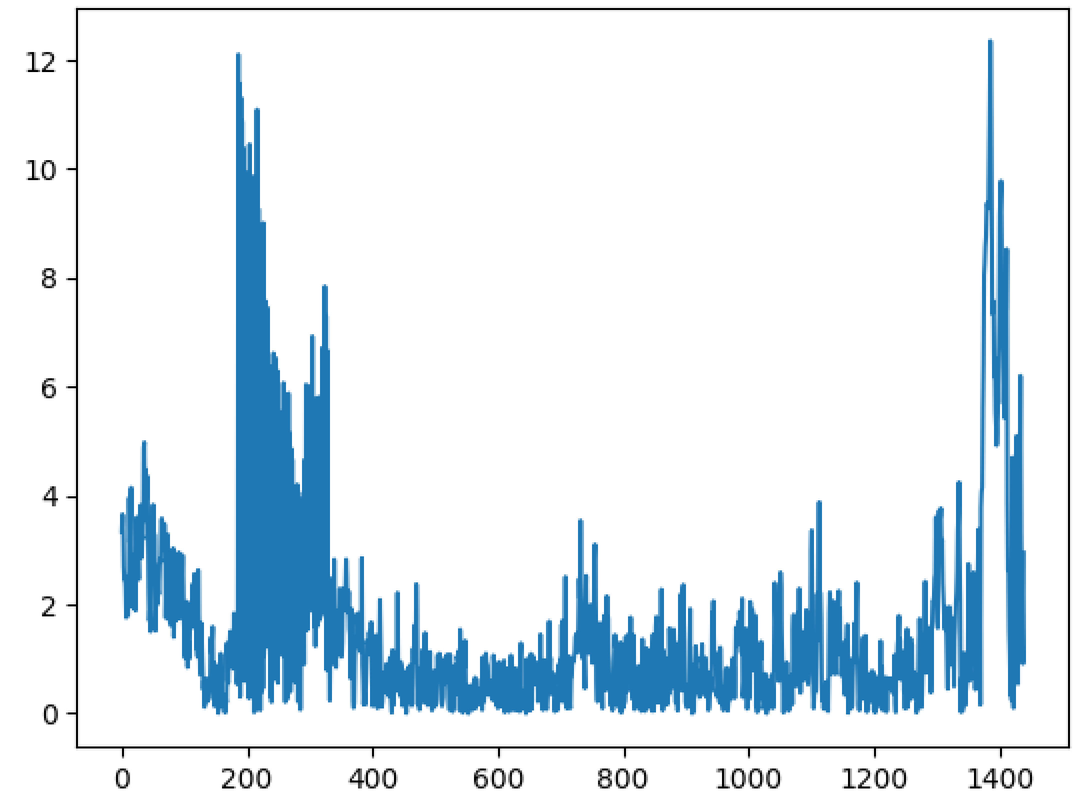


Fig. 7. Distribution of the difference between the true value and the predicted value of the steering angle over a full rotation.

The following results are also obtained from the real dataset collected by the group members.

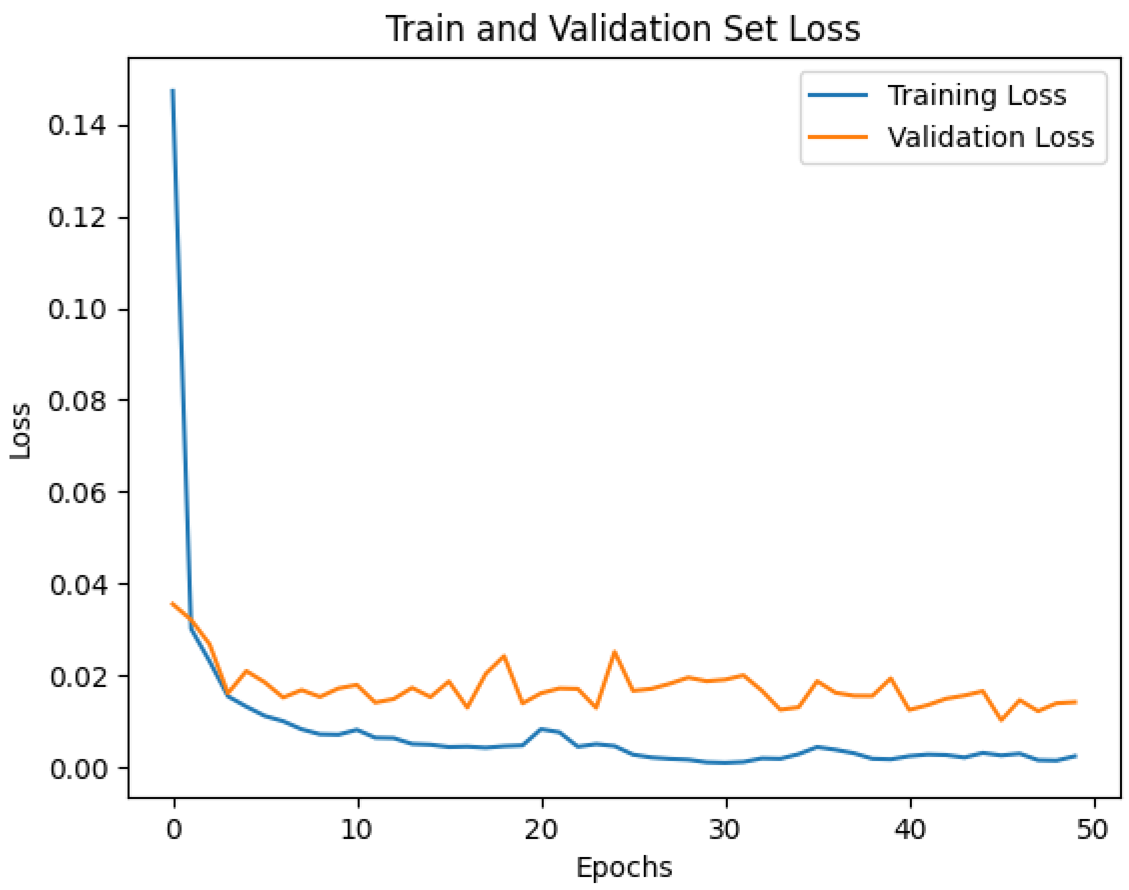


Fig. 8. Train and validation set loss versus the epoch number for the gathered dataset

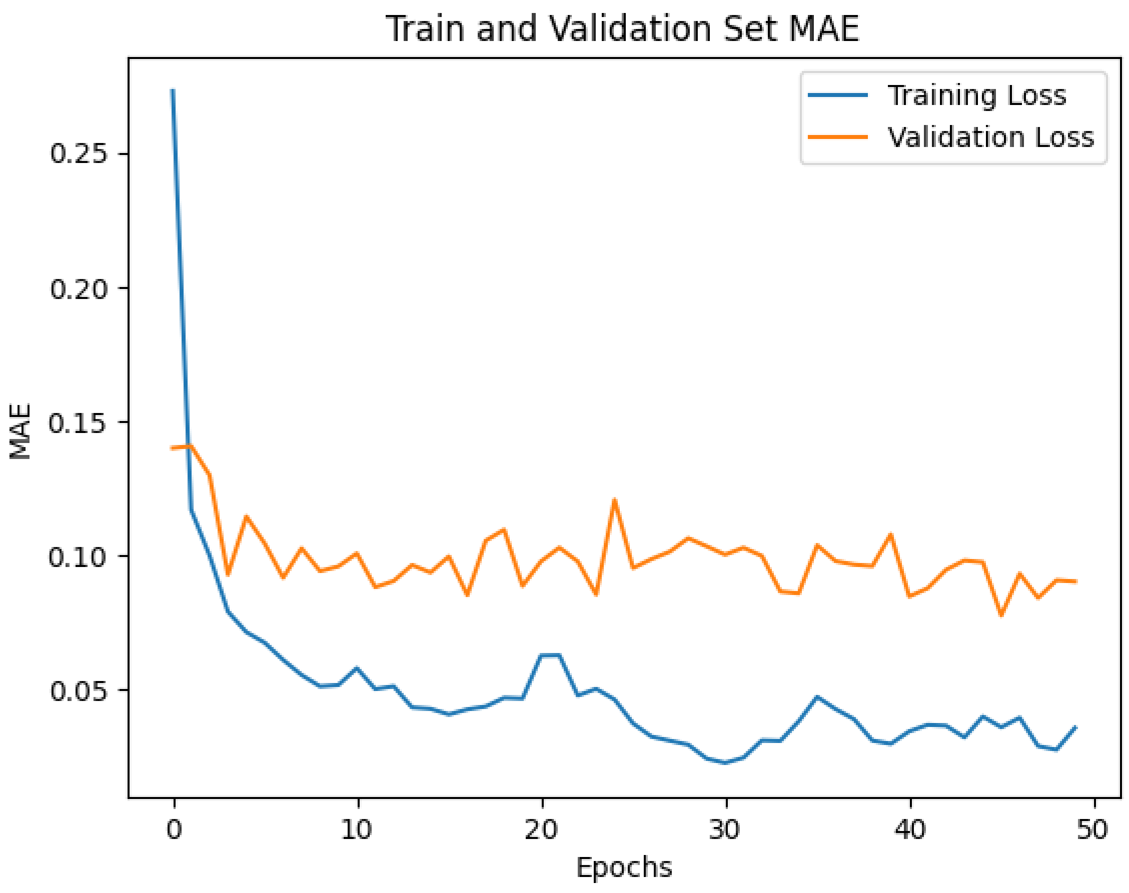


Fig. 9. Train and validation set loss versus the epoch number for the gathered dataset

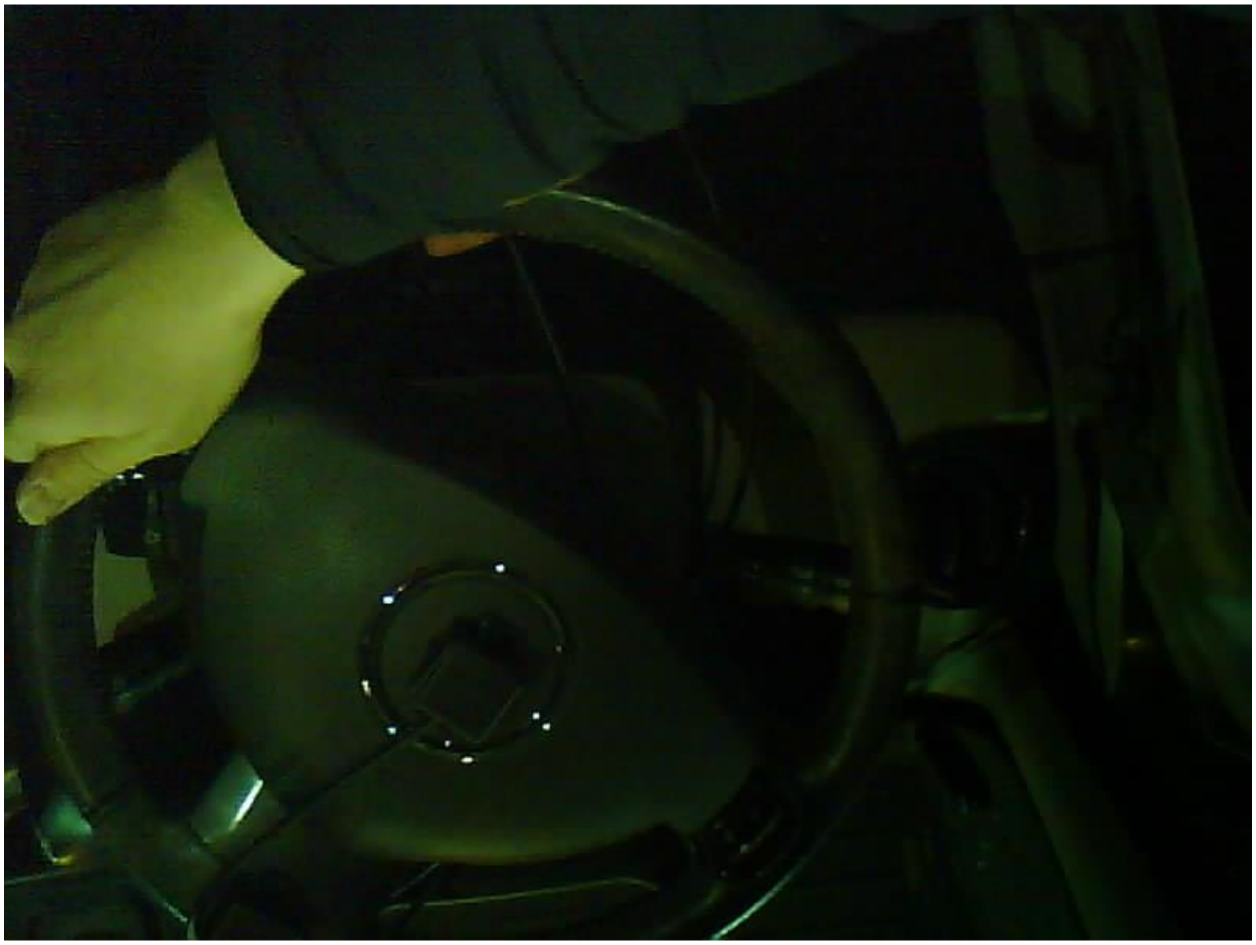


Fig. 10. An image from the real dataset gathered by the group members

IMU sensor

Measuring steering angle with IMU (inertia measurement unit) sensor

In this section of project, we used two IMU sensors (MPU6050) to measure the steering angle of the car.

Theory of measuring Roll, Pitch and yaw angle with IMU

Gyroscope is a sensor which measures angular velocity around x, y and z axis. As we know angle can be measured with Integration of speed with respect to time we can estimate the angle.

$$\theta = \int_{t}^{t+\Delta t} \dot{\theta} dt \cong \sum_{i=0}^{t} \dot{\theta}_i T_s$$

T_s is the time step.

Accelerometer sensor measures the difference between any linear acceleration in the accelerometer's reference frame and the earth's gravitational field vector. Consider $\boldsymbol{G_p}$ be the output of the sensor.

$$G_p = \begin{bmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{bmatrix} = \boldsymbol{R}(g - a_r)$$

R is Rotation matrix.

$$R = R_{xyz} = R_x(\phi)R_y(\theta)R_z(\psi)$$

If we assume that accelerometer has no acceleration and $a_r = 0$ then we would have:

$$G_p = R_{xyz}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = R_x(\phi)R_y(\theta)R_z(\psi)\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix}$$

So

$$\tan\phi = \frac{G_{py}}{G_{pz}}$$

$$\tan\theta = \frac{-G_{px}}{\sqrt{G_{py}^2 + G_{pz}^2}}$$

By these equations we can estimate Roll and Pitch angle.

As you noticed we cannot measure the yaw angle with accelerometer and the yaw angle can be just estimated by gyroscope and therefore we cannot use fusion algorithms like Kalman filter to combine gyroscope and accelerometer data to get more accurate estimation of the angle with less noises so the yaw angle is not estimated very well by just 6-Axis IMU and it is recommended to fuse gyro and accelerometer by other sensors like magnetometer. But in case of Roll and Pitch angle fusion like Kalman filter yields to less error in estimating these angles.

Code description:

the code below is written for Arduino Uno in Arduino IDE and the hardware is two MPU6050 connected to Arduino with I2C communication protocol. next I will explain most important parts of the

The first section of the code is related to importing libraries.

```
//include libraries
#include <Wire.h>
#include <MPU6050.h>
#include <KalmanFilter.h>
```

Next we define two objects from the MPU6050 class for the two IMU's we have used.

```
//define objects of IMU's
MPU6050 mpu;
MPU6050 mpu2;
```

Next is definition of initial values.

```
//setting time for gyroscope calculation
unsigned long timer = 0;
float timeStep = 0.01;
// Pitch, Roll and Yaw values
float gyroPitch = 0;
float gyroRoll = 0;
float gyroYaw = 0;

float gyroPitch2 = 0;
float gyroRoll2 = 0;
float gyroYaw2 = 0;
float accPitch = 0;
float accRoll = 0;

float accPitch2 = 0;
float accRoll2 = 0;

float kalPitch = 0;
float kalRoll = 0;

float kalPitch2 = 0;
float kalRoll2 = 0;

KalmanFilter kalmanX(0.001, 0.003, 0.03);
KalmanFilter kalmanY(0.001, 0.003, 0.03);

KalmanFilter kalmanX2(0.001, 0.003, 0.03);
KalmanFilter kalmanY2(0.001, 0.003, 0.03);

typedef unsigned long long ULL;
ULL startTime = 0;
ULL passedTime = 0;
ULL t = 0;


bool activated = false;
```

In the setup() function we first begin Serial communication with baud rate equal to 115200, then we configure two IMU's with begin function that take arguments for scale and range of accelerometer and the most important the address of the IMU's. as I mentioned before we have two IMU's that first one's I2C address is 0x68 and second one's is 0x69.

```
void setup()
{
  Serial.begin(115200);

  // Initialize MPU6050
  while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G,0x68))
  {
    delay(500);
  }

  while(!mpu2.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G,0x69))
```

```
 {
   delay(500);
 }
```

Then we have the main loop code. First I should explain that we have a micro switch attached to the GPIO pin 11 of the Arduino and when this bottom is pressed the board starts to read the sensor data.

We define a variable called activated which its value is set by the state of bottom. when we press the switch value of activated variable would be changed to true and the code would go to the else section in which IMU data would be calculated. And then first the time is the first thing that would be printed on serial monitor and would be the first column of the Excel data.

```
void loop()
{
// micro switch bottom
  if(!activated){
    if(digitalRead(11) == LOW){
      startTime = millis();
      activated = true;
    }
  }else{
  passedTime = millis();
  t = passedTime - startTime;

  Serial.print("\n");
  Serial.print((t/1000.0));
  Serial.print(",");
```

Then we update the accelerometer and gyroscope data. Next the Roll, Pitch and yaw angle are measured from gyroscope data by numerically integrating the angular velocity which gyroscope measures. The method for numerically integration is simple it is a recursive function that summarizes the angle of last increment with $angular\_velocity \times time$ of the present increment. Next step is calculating Roll and Pitch data from accelerometer. This is done by the equations driven from the kinematic relation of the acceleration vector of body frame with respect to the inertia frame that can be achieved with rotation matrix. Here only two angle Roll and Pitch of the three angles can be measured with accelerometer because the acceleration vector is a vector with magnitude equal to 1 and hence can only move on the surface of a sphere with radius of 1 therefor it has two degrees and freedom and equation can be derived only for Roll and Pitch.

The last part is calculating Roll and Pitch angle by applying a Kalman filter to gyroscope and accelerometer data. Kalman filter is a sort of sensor fusion and combines data from accelerometer and gyroscope to get more accurate results. The reason that we have only Kalman Roll and Pitch angles is that we can just estimate Yaw angle by gyroscope and there is no data from accelerometer to combine them. It is the reason that 6-Axis IMU has limitation in estimating yaw angle and it is better to fuse it with another sensors like magnetometer sensor (9_Axis IMUs like MPU9250) or camera for better position estimating.

```
Vector acc = mpu.readNormalizeAccel();
 Vector gyr = mpu.readNormalizeGyro();

 Vector acc2 = mpu2.readNormalizeAccel();
 Vector gyr2 = mpu2.readNormalizeGyro();

 gyroPitch = gyroPitch + gyr.YAxis * timeStep;
 gyroRoll = gyroRoll + gyr.XAxis * timeStep;
 gyroYaw = gyroYaw + gyr.ZAxis * timeStep;

 gyroPitch2 = gyroPitch2 + gyr2.YAxis * timeStep;
 gyroRoll2 = gyroRoll2 + gyr2.XAxis * timeStep;
 gyroYaw2 = gyroYaw2 + gyr2.ZAxis * timeStep;


 // Calculate Pitch & Roll from accelerometer (deg)
 accPitch = -(atan2(acc.XAxis, sqrt(acc.YAxis*acc.YAxis + acc.ZAxis*acc.ZAxis))*180.0)/M_PI;
 accRoll  = (atan2(acc.YAxis, acc.ZAxis)*180.0)/M_PI;

 accPitch2 = -(atan2(acc2.XAxis, sqrt(acc2.YAxis*acc2.YAxis + acc2.ZAxis*acc2.ZAxis))*180.0)/M_PI;
 accRoll2  = (atan2(acc2.YAxis, acc2.ZAxis)*180.0)/M_PI;

 // Kalman filter
 kalPitch = kalmanY.update(accPitch, gyr.YAxis);
 kalRoll = kalmanX.update(accRoll, gyr.XAxis);

 kalPitch2 = kalmanY2.update(accPitch2, gyr2.YAxis);
 kalRoll2 = kalmanX2.update(accRoll2, gyr2.XAxis);
```

In the next section we print the outputs on the Serial monitor. The printed outputs are first the Roll angle calculated by Kalman filter of two IMUs which are the most important outputs and the steering angle we want to measure is actually the Roll angle of two IMUs.

```
Serial.print(kalRoll);
Serial.print(",");
Serial.print(kalRoll2);
Serial.print(",");
//=============== imu1 ================
Serial.print(acc.XAxis);
Serial.print(",");
Serial.print(acc.YAxis);
Serial.print(",");
Serial.print(acc.ZAxis);
Serial.print(",");
Serial.print(gyr.XAxis);
Serial.print(",");
Serial.print(gyr.YAxis);
Serial.print(",");
Serial.print(gyr.ZAxis);
Serial.print(",");
Serial.print(accPitch);
Serial.print(",");
Serial.print(accRoll);
Serial.print(",");
Serial.print(gyroPitch);
Serial.print(",");
Serial.print(gyroRoll);
Serial.print(",");
Serial.println(gyroYaw);
Serial.print(",");
Serial.print(kalPitch);
Serial.print(",");
//=========== imu 2 ==============
Serial.print(acc2.XAxis);
Serial.print(",");
Serial.print(acc2.YAxis);
Serial.print(",");
Serial.print(acc2.ZAxis);
Serial.print(",");
Serial.print(gyr2.XAxis);
Serial.print(",");
Serial.print(gyr2.YAxis);
Serial.print(",");
Serial.print(gyr2.ZAxis);
Serial.print(",");
Serial.print(accPitch2);
Serial.print(",");
Serial.print(accRoll2);
Serial.print(",");
```

```
 Serial.print(gyroPitch2);
 Serial.print(",");
 Serial.print(gyroRoll2);
 Serial.print(",");
 Serial.println(gyroYaw2);
 Serial.print(",");
 Serial.print(kalPitch2);

 }
```

Explaining about the MPU6050 library source code

There are three functions in this library which read raw data from sensor. The raw accelerometer and gyroscope data are the data directly read from the sensor, typically in terms of the sensor's digital counts. For example, for the accelerometer this raw data represents the instantaneous acceleration along each axis as measured by the accelerometer.

And also, there are functions the normalize the gyroscope and accelerometer data. In the case of accelerometer for example, normalized accelerometer data is the raw accelerometer data that has been converted into a more human-readable and interpretable form. The raw data is often converted into units such as meters per second squared (m/s^2) or gravitational acceleration (g). Normalizing the data allows you to compare it more easily to real-world values and standards.

In the code which is for reading raw accelerometer data it first begins the communication with sensor and requests 6 byte of data from sensor. 2 bytes for each axis accelerometer data therefor the output of the sensor is a 16-bit data.

```
Vector MPU6050::readRawAccel(void) {
 Wire.beginTransmission(mpuAddress);
#if ARDUINO >= 100
 Wire.write(MPU6050_REG_ACCEL_XOUT_H);
#else
 Wire.send(MPU6050_REG_ACCEL_XOUT_H);
#endif
 Wire.endTransmission();

 Wire.beginTransmission(mpuAddress);
 Wire.requestFrom(mpuAddress, 6);
```

The it begins reading the data byte by byte which is 6 byte of data. For each axis it receives two bytes a high byte and low byte. The it shifts high byte by 8 position to left and combines it with low byte by bitwise OR operator to get a 16-bit value.

```
  while (Wire.available() < 6)
    ;

#if ARDUINO >= 100
  uint8_t xha = Wire.read();
  uint8_t xla = Wire.read();
  uint8_t yha = Wire.read();
  uint8_t yla = Wire.read();
  uint8_t zha = Wire.read();
  uint8_t zla = Wire.read();
#else
  uint8_t xha = Wire.receive();
  uint8_t xla = Wire.receive();
  uint8_t yha = Wire.receive();
  uint8_t yla = Wire.receive();
  uint8_t zha = Wire.receive();
  uint8_t zla = Wire.receive();
#endif

  ra.XAxis = (int16_t)(xha << 8 | xla);
  ra.YAxis = (int16_t)(yha << 8 | yla);
  ra.ZAxis = (int16_t)(zha << 8 | zla);

  return ra;
}

Vector MPU6050::readNormalizeAccel(void) {
  readRawAccel();

  na.XAxis = ra.XAxis * rangePerDigit * 9.80665f;
  na.YAxis = ra.YAxis * rangePerDigit * 9.80665f;
  na.ZAxis = ra.ZAxis * rangePerDigit * 9.80665f;

  return na;
}
```

The next function which normalizes the raw acceleration data with multiplying raw data by rangeperdigit first and then earth’s gravitational acceleration to calculate a number in (m/s^2) unit.

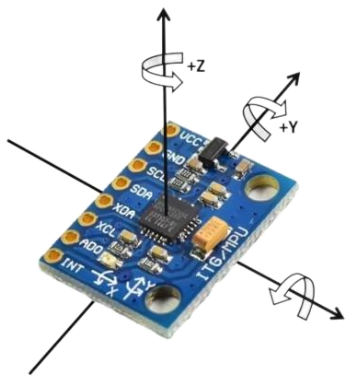
the term "range per digit" refers to the sensitivity or resolution of the sensor, and it indicates how the raw digital counts from the sensor should be interpreted in terms of physical units.

```
Vector MPU6050::readScaledAccel(void) {
  readRawAccel();

  na.XAxis = ra.XAxis * rangePerDigit;
  na.YAxis = ra.YAxis * rangePerDigit;
  na.ZAxis = ra.ZAxis * rangePerDigit;

  return na;
}
```

This two functions do almost the similar things in term of gyroscope also.

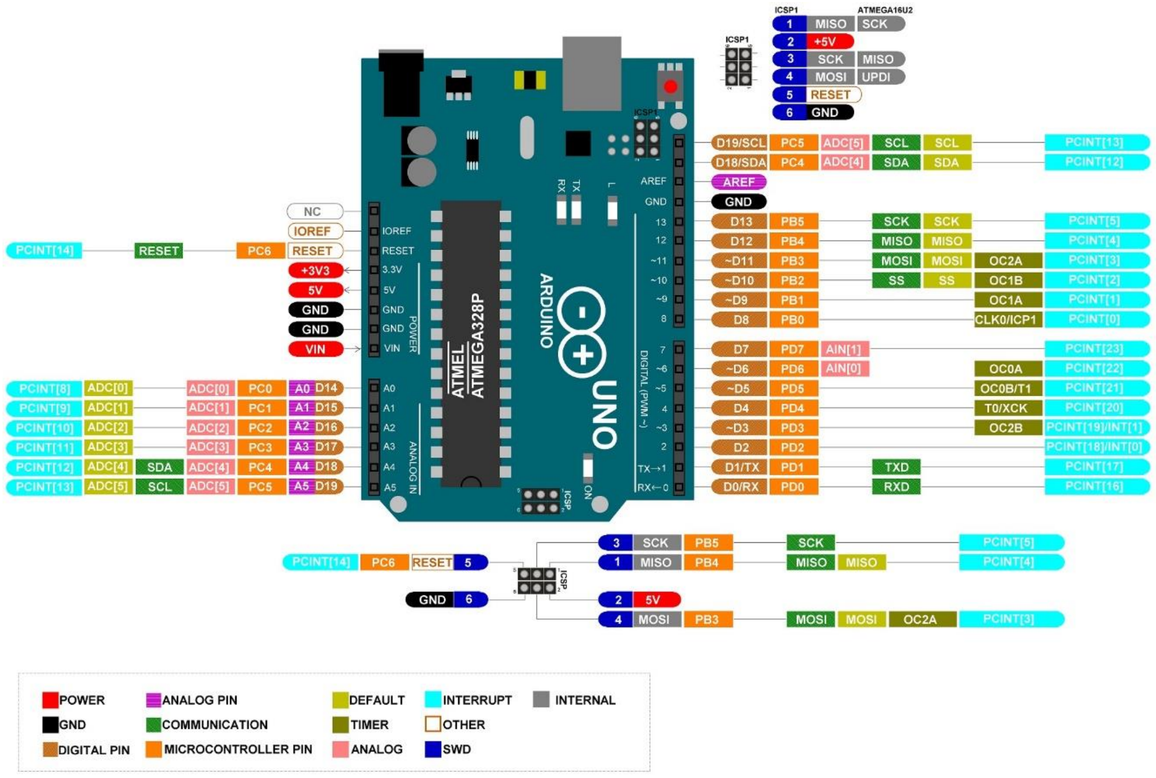
Hardware of IMU data acquisition system

We made a setup for data gathering to train our neural network with. The setup contains an Arduino Uno board and two MPU6050 IMU sensors.

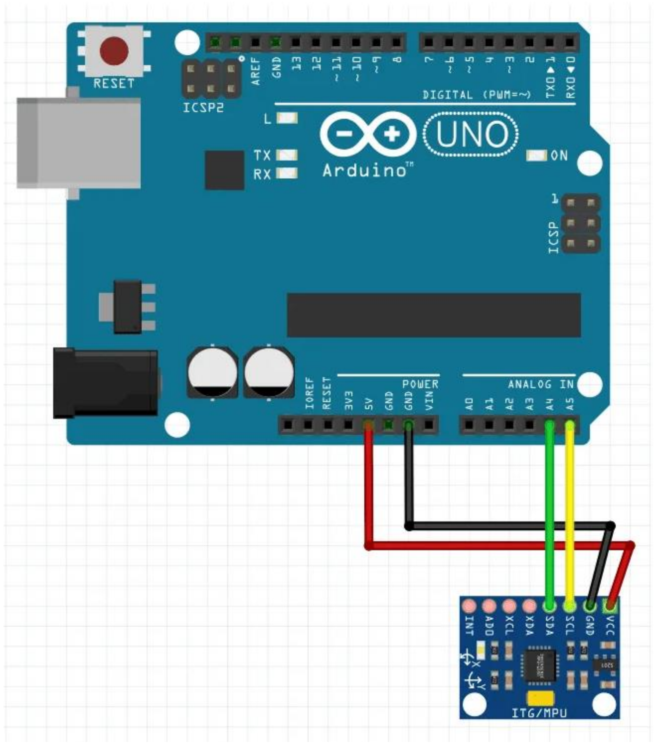


MPU6050 has 4 main pins containing VCC, GND, SCL(clock) and SDA(data) that should be connected to Uno.

The communication protocol between IMU and Arduino is I2C hence according to pinout diagram of Arduino Uno we should connect SCL to A5 pin and SDA to A4 pin.



The wiring would be like this.



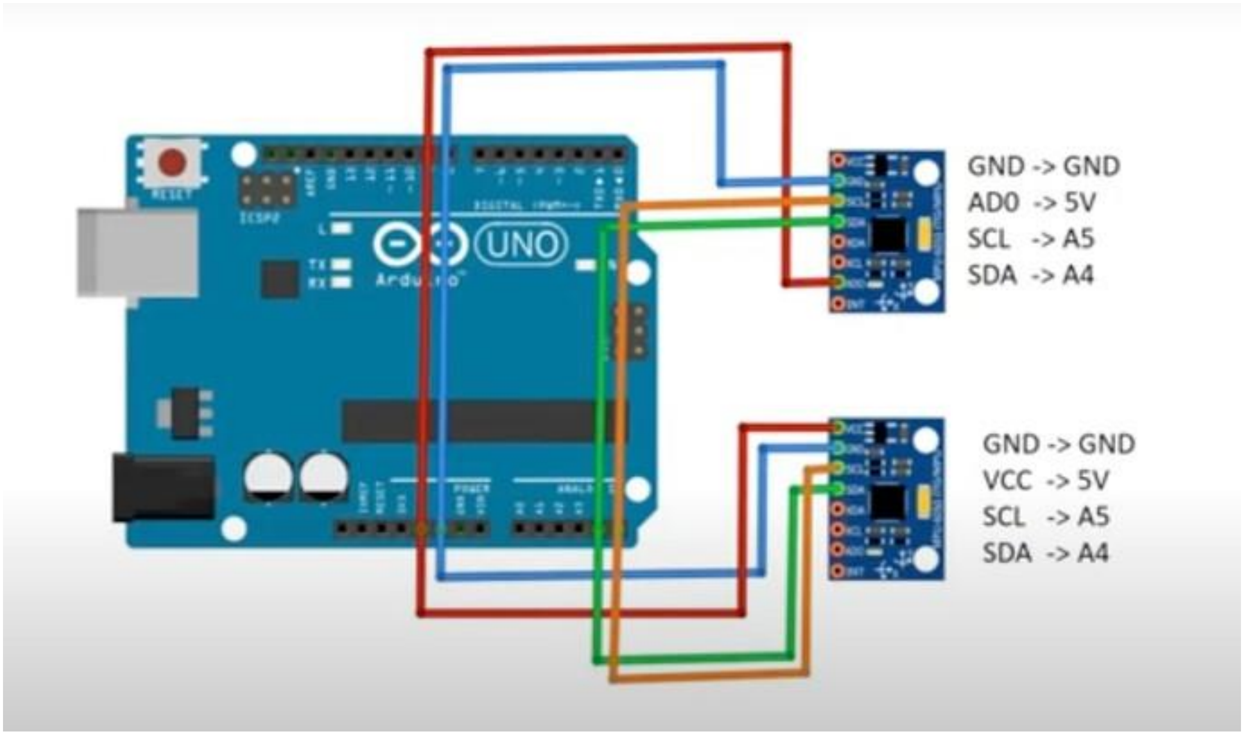
But as I mentioned we have two IMUs and for connecting two of them to Arduino board we should connect second IMU also to A4 and A5 pin but with this difference that AD0 pin of the second IMU should be pulled up to VCC and its state should be HIGH. This is done to make the I2C address of the two IMUs different.

When the AD0 pin is LOW the address would be 0x68 that is the default address but for using the second IMU its address must be different and by making the AD0 pin HIGH the address would be 0x69. So then in the code the in the board would first make communication with first IMU and reads its data then begin communication with second IMU and reads its data too.

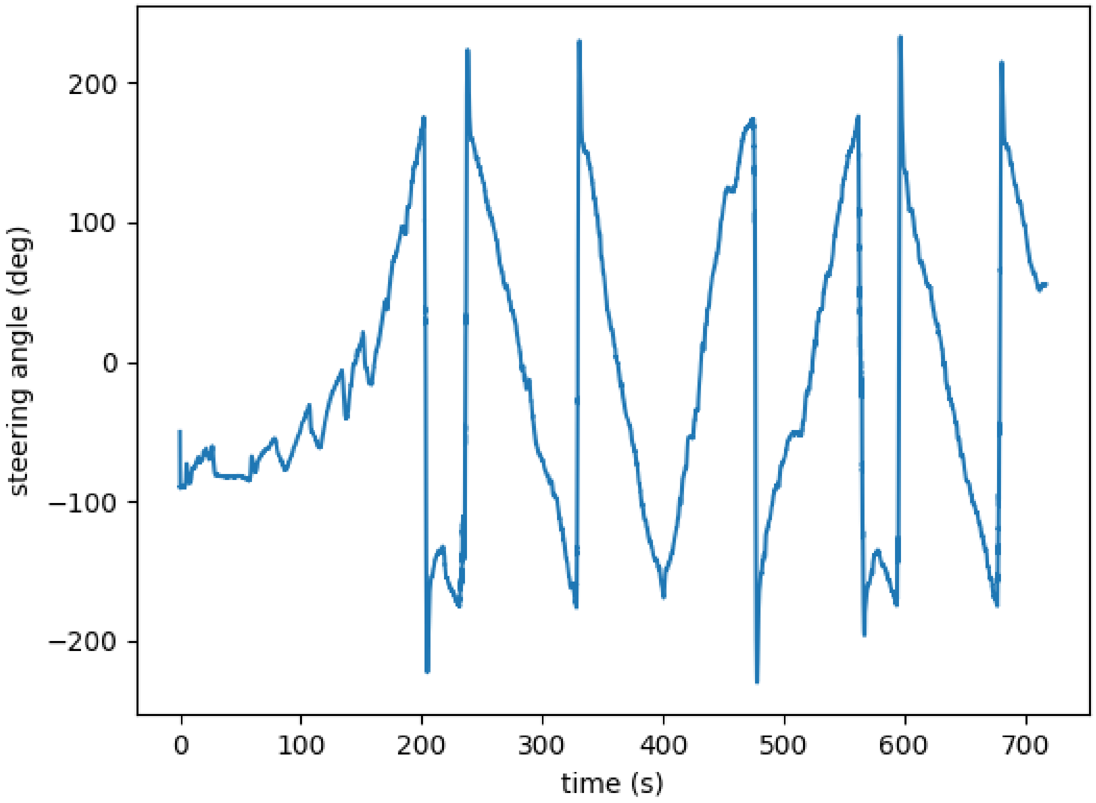
In this setup we had a micro switch that its function was to initiate the data gathering process as it pushed. There is this switch also in camera setup and its function is too start camera system and IMU system by pushing this two switch at the same time to synchronize IMU data and image data. There would absolutely be a slight difference between pushing time of the switches but actually Because we turn steering wheel with low speed it would not be a big trouble.

The circuit diagram of the micro switch is provided in camera setup section, it is just mentionable that it would be connected to pin number 11 of the Arduino.

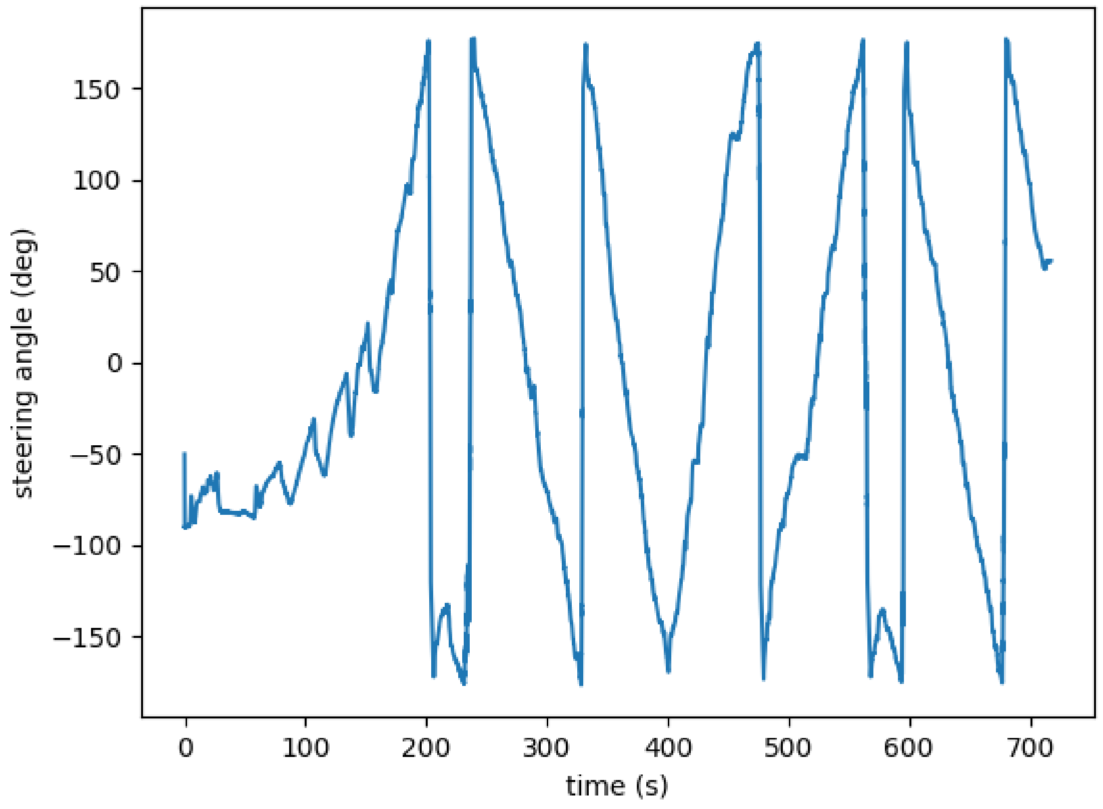
The final wiring is like this.



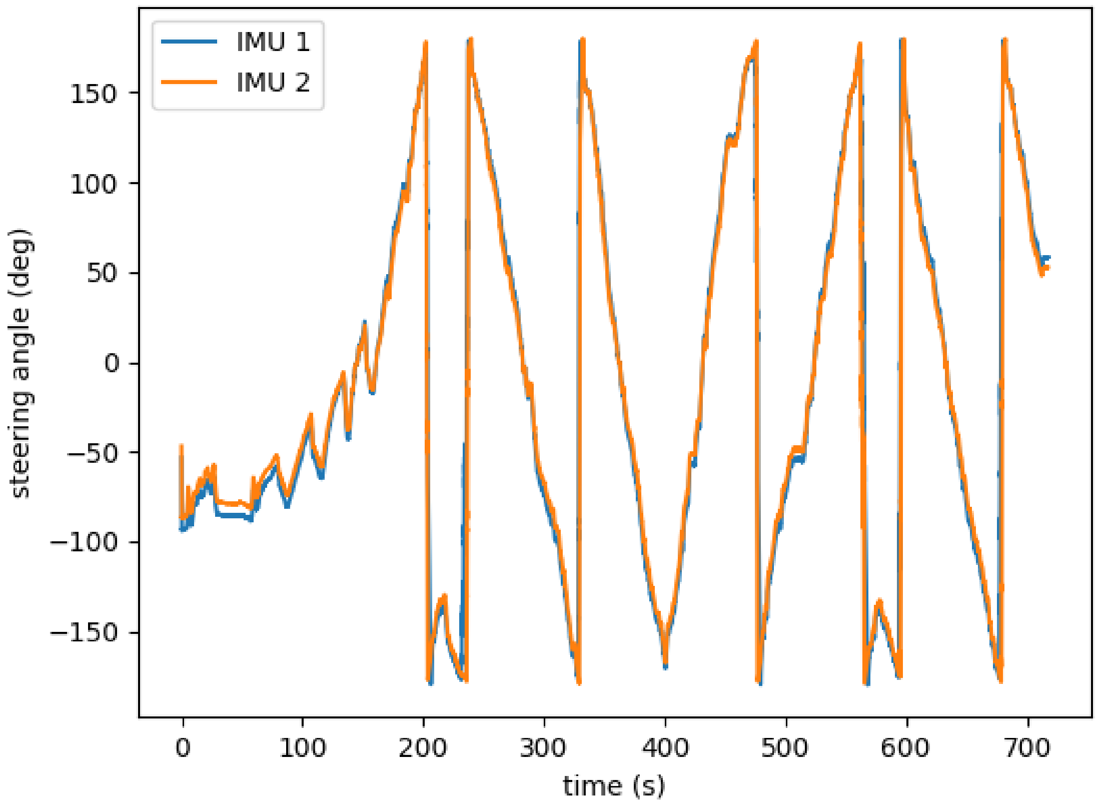
Here are some plots which are obtained based on the data acquired by the IMU sensors.



The average of the output angles of the two IMU sensors. Some of the data points are outliers (less than -180 deg or greater than +180 deg).



The average of the output angle of the two IMU sensors, with outliers removed.



Comparison of the output angles of the two IMU sensors with outliers removed.

What is Arduino?

Arduino is an open-source platform that helps circuit developers build electronic projects. It consists of both hardware and software. Arduino hardware is a programmable circuit board called a microcontroller. Arduino software is an IDE (integrated development environment) through which developers write and upload the code to the microcontroller.
We can feed a program with a set of instructions to the Arduino board that can carry out simple to complex tasks. Traditional programmable circuit boards require separate hardware to load the code onto the board. But Arduino eliminates the need for hardware; instead, it uses a simple USB cable to load code onto the Arduino board.
The Arduino board enables developers to feed the program in the simplified version of the C++ language, making it easier for them to learn and code.
Arduino Hardware
The hardware part of the Arduino is its programmable circuit board. You might have come across various Arduino boards in your work, but the most commonly used Arduino board is Arduino UNO. All Arduino boards have a microcontroller known as a small computer, which is the heart of Arduino.
So, when learning about Arduino, it’s essential to learn about microcontrollers and how to use them. The Arduino microcontroller is responsible for reading different inputs and controlling the outputs.
Arduino Software
Arduino’s software is called Arduino IDE. You can download the software on your computer and program the Arduino boards to perform various tasks accordingly.
IDE is similar to a text editor, where you write instructions for the Arduino board.

Arduino Code
Apart from hardware and software, the third most important aspect of Arduino is its code, also known as a sketch. You can write the code in the Arduino IDE and load it onto the board.

You might be wondering what language Arduino uses for scripting the code. Arduino has its native language analogous to C++, called Arduino Programming Language. Any program developed using Arduino Programming Language is called sketch and saved in a file with the .ino extension.

To write instructions for Arduino boards, you must possess a basic understanding of C and C++ programming languages.

Together, hardware, software, and code make up Arduino.

Types of Arduino Boards

You can create different boards, each with additional capabilities using Arduino. It's an open-source hardware, allowing anyone to make changes and create various derivatives of Arduino boards. These changes let you add multiple functionalities.

Below are some common types of Arduino even computer novices can use:

1. Arduino UNO (R3)

The UNO is one of the most accessible options to get your feet wet in the electronics field. It depends on an ATmega328P-based microcontroller, and features the following:

- 14 digital I/O pins
- 6 pins for PWM
- 6 pins for analog inputs
- a reset button
- a USB connection

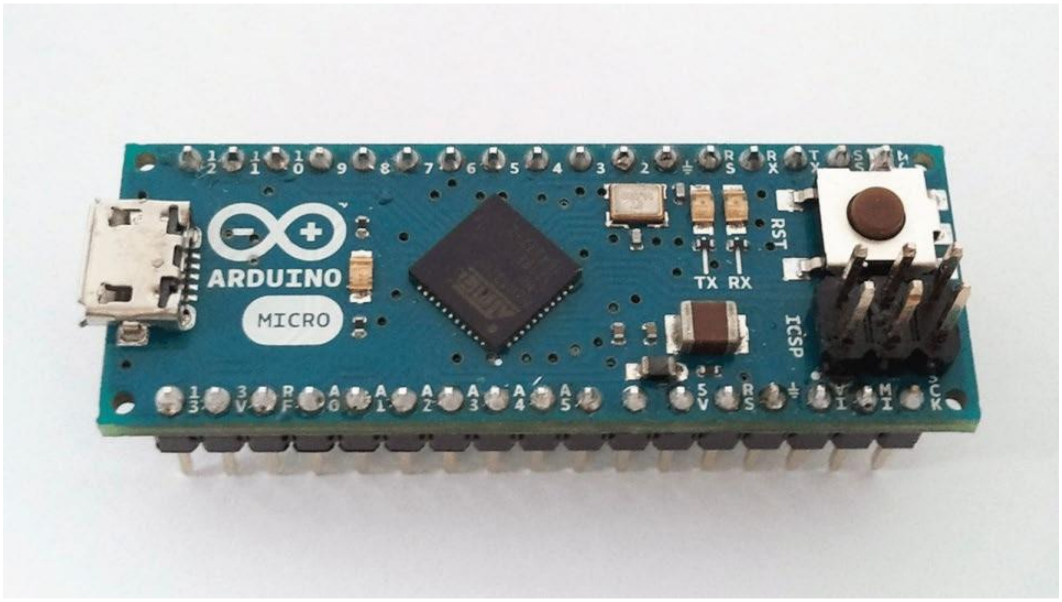
Arduino UNO (R3) has everything it needs to hold up the microcontroller. All you need to do is attach it to your computer via USB cable and provide the supply with an AC-to-DC adapter or battery to make it work.

2. Arduino Nano

This board has connections similar to UNO but is based on ATmega328P and ATmega628 microcontrollers. This small, flexible, and reliable board with a mini USB and is great for creating projects.

This board comes with 8 analog pins, 14 digital pins with an I/O pin, 6 power pins & 2 RST (reset) pins.

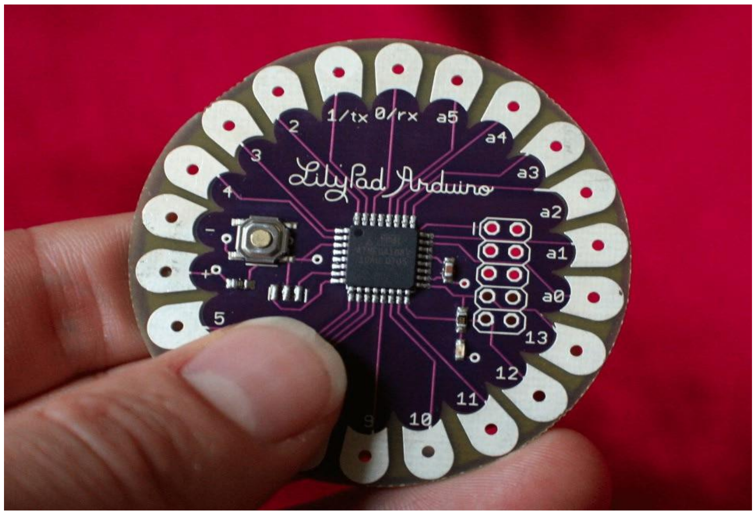
3. Arduino Micro



This type of Arduino has an ATmega32U4-based microcontroller with 20 sets of pins, 7 of which are for PWM, and 12 analog input pins. The Arduino Micro also comes with an ICSP header, RST button, small USB connection, and 16-MHz crystal oscillator.

The miniature of the Leonardo board, this Arduino Micro has a built-in USB connection.

4. Arduino Lilypad

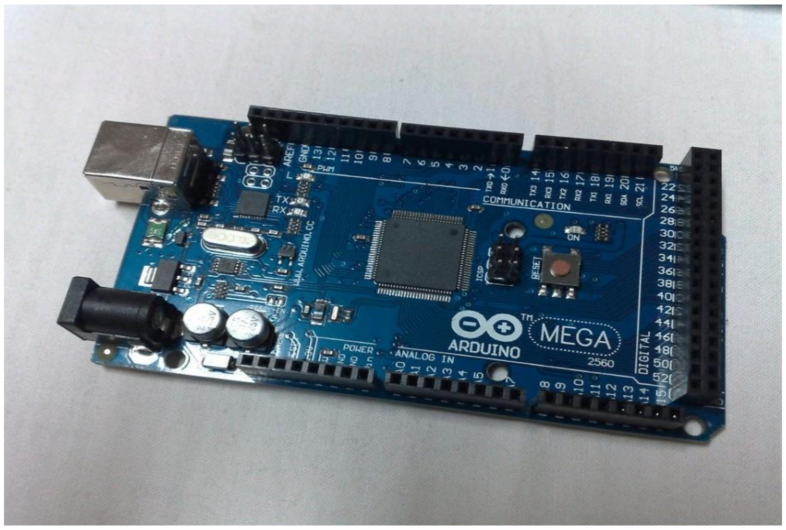


The Lily Pad Arduino board is designed as an e-textile technology. This hardware is expanded by Leah “Buechley” and designed by “Leah and SparkFun.” Each board is designed with substantial connecting pads, having a smooth back that you can sew into clothing using conductive thread. This Arduino has I/O, power, and sensor boards. A nice bonus is that this Arduino board is even washable!

5. Arduino RedBoard

You can use the mini-b USB to program this board. This board is flat on the back like the previous Lilypad, making placement easy. You can efficiently use this board with Windows 8 without having to change the settings. Further, you just need to plug this board into your system and write code to upload on the board. You can control this board using a barrel jack along with the USB cable.

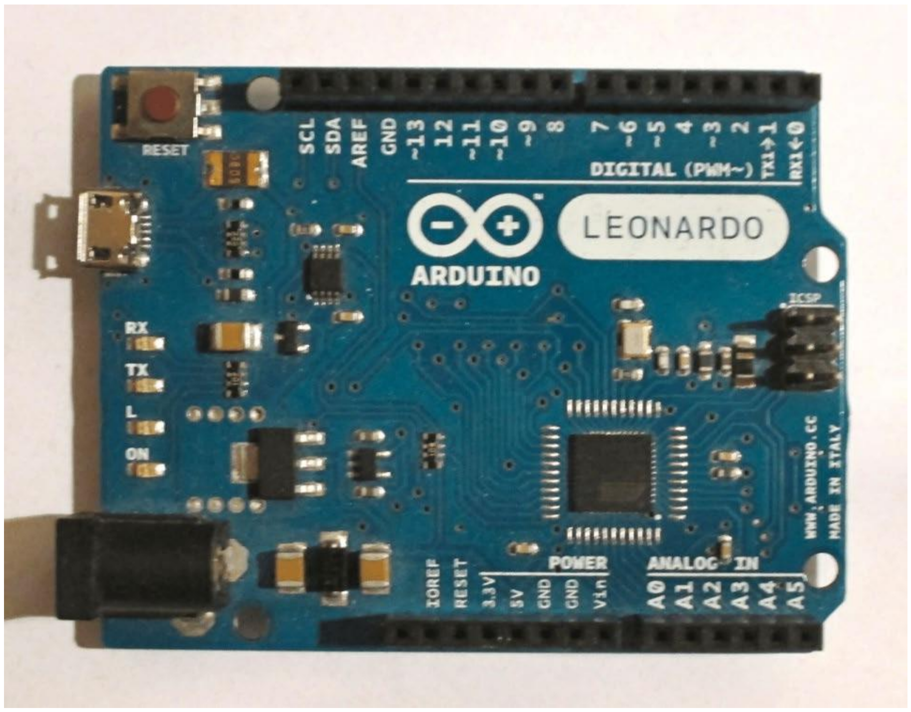
6. Arduino Mega R3



The Arduino Mega R3 is an expanded form of Arduino UNO and comes with a digital I/O pin. It also has 14 pins that work as PWM o/ps, 6 pins for analog inputs, a reset button, a power jack, and a USB connection.

You can use a USB cable to communicate with a computer. This Arduino board is ideal for designing projects that require substantial digital inputs and outputs.

7. Arduino Leonardo



This Arduino was introduced as the first development board with one microcontroller and a USB. It is the simplest and cheapest type of Arduino, making it extremely suitable for novices.

What is on the Arduino Board?

Among all Arduino boards available on the market, the easiest and commonly used board is Arduino UNO. Some of these boards may have a different look and feel, but all have some standard components. So, let us discuss its fundamental components here.

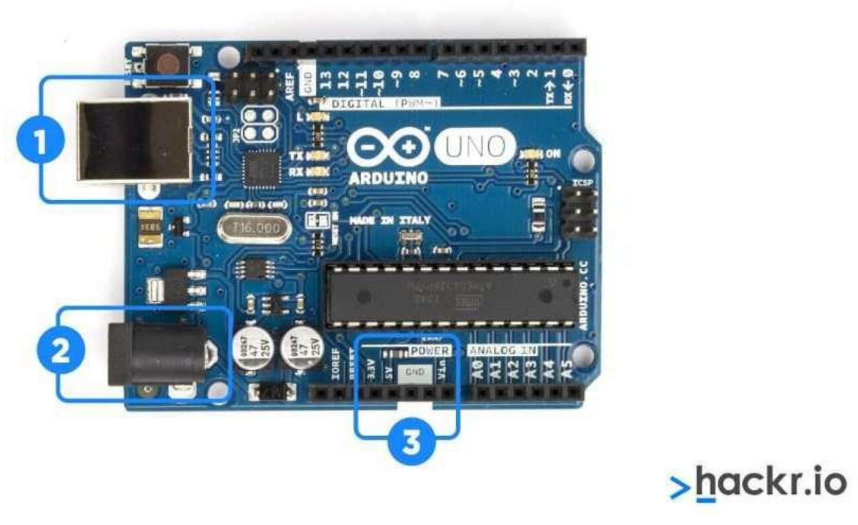
Arduino UNO

It is a simple and commonly used prototyping board suitable even for beginners to get along with electronics. Being the basic one, it is essential for every electronics developer to know its different components.

UNO comes with an ATmega328P microcontroller. It has two variants: one has a through-hole microcontroller connection and the other has a surface mount type. In the through-hole model, you can replace its chip with a new one in case of any error.

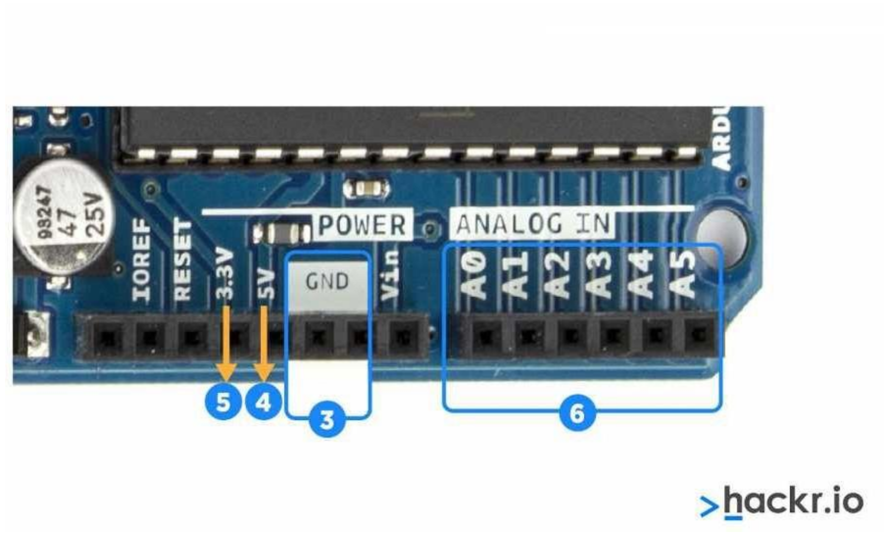
Arduino UNO is an 8-bit microcontroller with the AVR architecture, and it offers different features and capabilities.

UNO comes with a total of 14 digital input-output (I/O) pins that you can use as input or output. Out of these 14 pins, you can use the six pins for producing PWM signals. Each pin on this board works at 5V and has a current of 20mA.



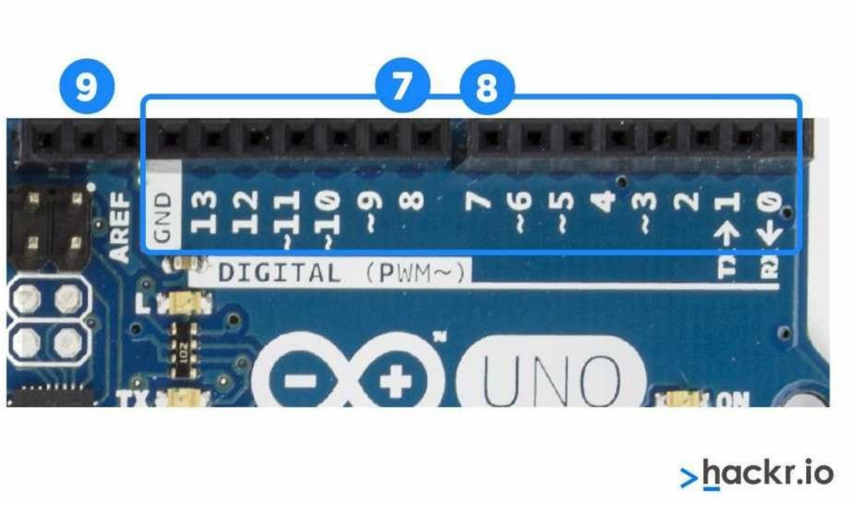
- We always need a power source to make txhe board work. You can power this board using a USB connection to your computer, and you can either use a wall power supply that will terminate in a barrel jack. In the above image, (1) specifies the USB and (2) specifies the barrel jack.

You can even load the code using a USB connection onto your Arduino board.



The above image shows the following:

- **GND (3):** GND stands for 'Ground," which is used to ground your circuit.
- **5V (4) & 3.3V (5):** The 5V pin can be used to supply five volts of power, and the 3.3V pin can be used to supply 3.3 volts of power.
- **Analog (6):** These pins labeled from (A0 to A5) are known as Analog pins. They will convert an analog sensor to a digital one.



The top right of the above image highlights the 14 I/O pins that can perform specific functions, as stated below:

- With pins 0 and 1, you can carry out serial communication to receive and transmit serial data. You can use them to program the Arduino board and communicate with a user via the serial monitor.

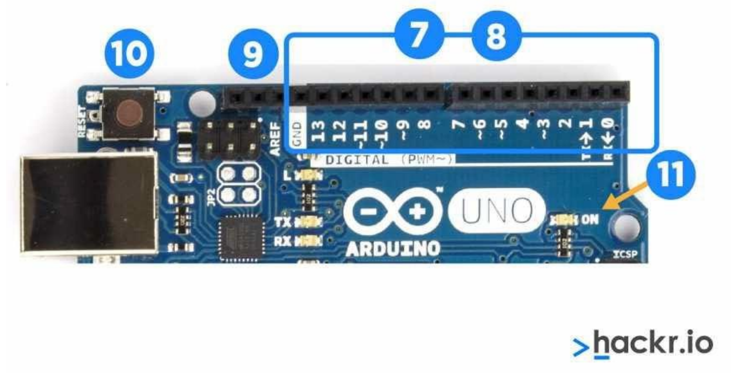
- With pins 2 and 3, you can provide external interrupts. These pins trigger an external event .

- Six pins (3-11) are used for 8-bit PWM output.

- Pins 10, 11, 12, and 13 are for SS, MOSI, MISO, and SCK, respectively, especially for SPI communication.

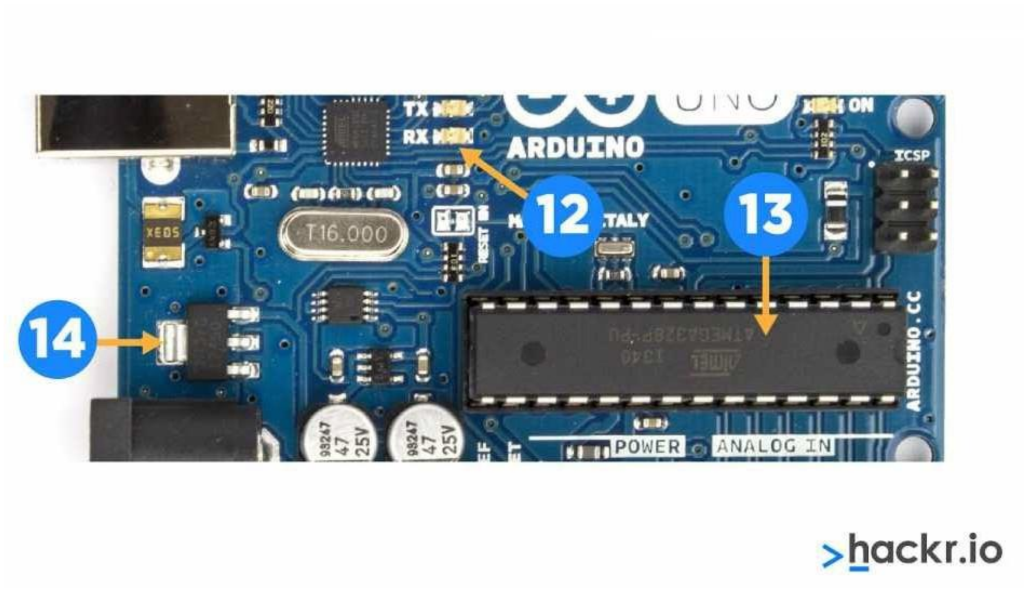
- Pin 13 comes with a built-in LED connection. When this pin is set to HIGH, the LED is turned on, and when it is LOW, the LED is turned off.

- AREF stands for Analog Reference, used to set an external reference voltage (0 -5 Volts).



- At the top left of the above image, (10) specifies the reset button. This button connects the reset pin to the ground and restarts the uploaded code. Pressing the reset button in case of failure will allow you to test your code multiple times.

- The number (11) specifies the Power LED Indicator, which will light up when powering your Arduino to a source.



- In the above image, (12) specifies the TX RX LEDs, where TX stands for *transmit* and RX for *receive*. These are used for serial communication. These LEDs provide visual indications while receiving or transmitting data using Arduino.

- (13) specifies the Integrated circuit, otherwise known as the brain of Arduino. You can see the IC type mentioned on the top of the IC.

- (14) specifies the voltage regulator, which helps control the amount of voltage supplied to the Arduino board. It acts as a gatekeeper, preventing an extra voltage from entering the circuit. Also, it comes with some limits, so do not connect the Arduino to more than 20 volts.

Why Should You Use Arduino?

Today, many people use Arduino. It’s easy to use and program, thus making it more popular among beginners and advanced users. You can connect Arduino to multiple platforms such as Mac, Windows, and Linux. Moreover, you can use it to create low-cost scientific instruments.

Arduino provides you with an opportunity to play around with microcontrollers. Below are some significant reasons to use Arduino.

Cheap: Arduino boards are affordable. If you know, you can even assemble Arduino by hand or use the pre-assembled Arduino modules that cost less than $50.

Cross-platform: You can plug your Arduino board on any platform such as Windows, Mac, and Linux operating systems.

Simple programming environment: The Arduino Software’s IDE is simple and easy to learn, as it comes with a simple version of C++.

Constraints of Arduino

Despite various reasons to use Arduino, you need to understand its limitations before working with it.

- **Memory**: Arduino does not have enough memory for storing programs and variables. Also, you cannot add external memory to it. ATmega32 and ATmega128 can be used for external memory, but you cannot utilize the I/O functions for those pins.

Arduino boards cannot accommodate external memory because of their basic design assumptions. It is important to remember that Arduino is an inexpensive Intel-based single-board computer and was not designed to replace a full-on computer system with high system requirements.

- **Speed**: The Arduino CPU clock rate is between 8 and 20 MHz — that is way slower than most platforms. You can execute several instructions in each clock cycle, and that's a lot of available CPU activity to be handled in between each pulse.

- **Electrical power**: When working with Arduino hardware, you need to consider voltage parameters since some devices have 3.3V I/O while others are 5V tolerant. If you connect a 5V transistor-transistor logic to a 3.3V device, it will impact the hardware and can harm your Arduino.

Applications of Arduino

The following are some typical applications of Arduino:

Robotics: Arduino is suitable for both entry- and intermediate-level robotics projects. You can give it basic commands to make a robot function, even with limited resources at hand. Some well-known examples of robots developed using Arduino are K'Nex Wall-Following Robot and SCARA Robot Arm.

Audio: From Hi-Fi to headphones, everything depends on sound quality. Unfortunately, Arduinos are not suitable for audio, but you can use them to add an audio element to your projects.

Tools: You can design devices like print farms, 3D printers, CNC machines, laser etchers, etc., using Arduino.

Networking: Most Arduinos offer built-in networking capabilities, usually in the form of an Ethernet port. You can utilize this facility in various projects, such as IoT and data-logging projects.

GPS: You can use Arduino to track devices and vehicles and create one of the most impressive applications involving GPS.

Well, there is no limit to its application, but we have only mentioned a few.

Conclusion

What is Arduino? It's your first step in developing creative new applications and electronic projects. That sums up the basics of Arduino. Whether you're a beginner or experienced circuit designer, opt for Arduino to create unique applications with this easy-to-install program. You can download Arduino IDE for coding and start working on your versions of Arduino, as it is available freely and is open-source.

Getting Started With ESP32-CAM

If you were asked a few years ago how much a digital camera with WiFi would cost, you probably wouldn't have said $10. But that is no longer the case.

The ESP32-CAM, a board that hit the market in early 2019, has changed the game. Amazingly, for less than $10, you get an ESP32 with support for a camera and an SD card.

These modules are seriously nifty. Whether you need to detect motion in your Halloween project, detect faces, decode license plates, or perhaps merely a security camera, it's worth having one in your DIY toolbox.

ESP32-CAM Hardware Overview

The heart of the ESP32-CAM is an ESP32-S System-on-Chip (SoC) from Ai-Thinker. Being an SoC, the ESP32-S chip contains an entire computer—the microprocessor, RAM, storage, and peripherals—on a single chip. While the chip's capabilities are quite impressive, the ESP32-CAM development board adds even more features to the mix. Let's take a look at each component one by one.

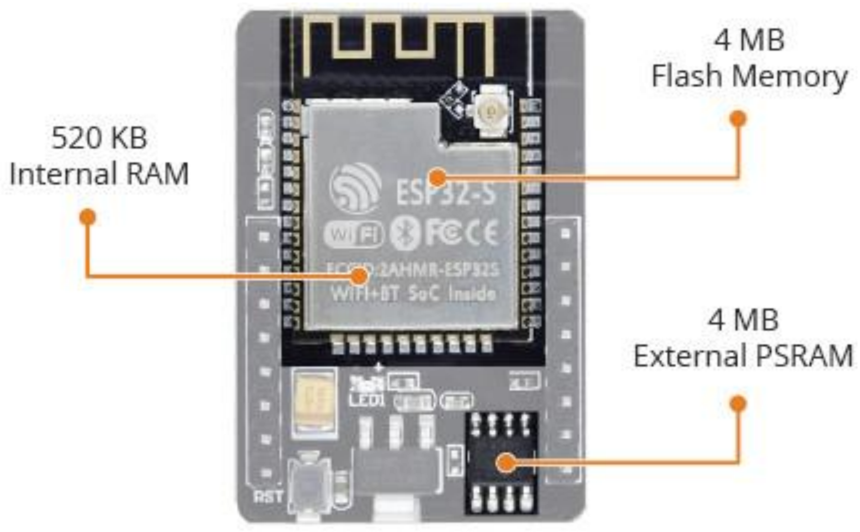
The ESP32-S Processor

The ESP32-CAM equips the ESP32-S surface-mount printed circuit board module from Ai-Thinker. It is equivalent to Espressif's ESP-WROOM-32 module (same form factor and general specifications).

The ESP32-S contains a Tensilica Xtensa® LX6 microprocessor with two 32-bit cores operating at a staggering 240 MHz! This is what makes the ESP32-S suitable for intensive tasks like video processing, facial recognition, and even artificial intelligence.

The Memory

Memory is paramount for complex tasks, so the ESP32-S has a full 520 kilobytes of internal RAM, which resides on the same die as the rest of the chip's components.



It may be inadequate for RAM-intensive tasks, so ESP32-CAM includes 4 MB of external PSRAM (Pseudo-Static RAM) to expand the memory capacity. This is plenty of RAM, especially for intensive audio or graphics processing.

All these features amount to nothing if you don't have enough storage for your programs and data. The ESP32-S chip shines here as well, as it contains 4 MB of on-chip flash memory.

The Camera

The OV2640 camera sensor on the ESP32-CAM is what sets it apart from other ESP32 development boards and makes it ideal for use in video projects like a video doorbell or nanny cam.

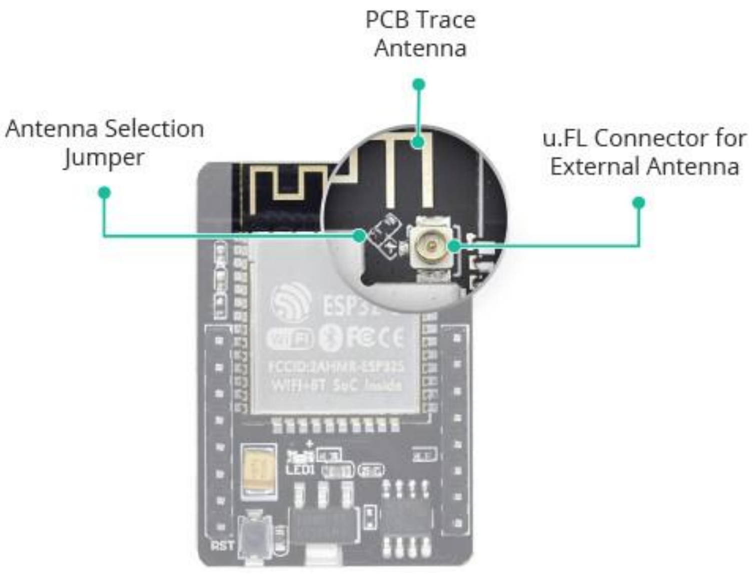
The OV2640 camera has a resolution of 2 megapixels, which translates to a maximum of 1600×1200 pixels, which is sufficient for many surveillance applications.

The Storage

The addition of a microSD card slot on the ESP32-CAM is a nice bonus. This allows for limitless expansion, making it a great little board for data loggers or image capture.

The Antenna

The ESP32-CAM comes with an on-board PCB trace antenna as well as a u.FL connector for connecting an external antenna. An Antenna Selection jumper (zero-ohm resistor) allows you to choose between the two options.

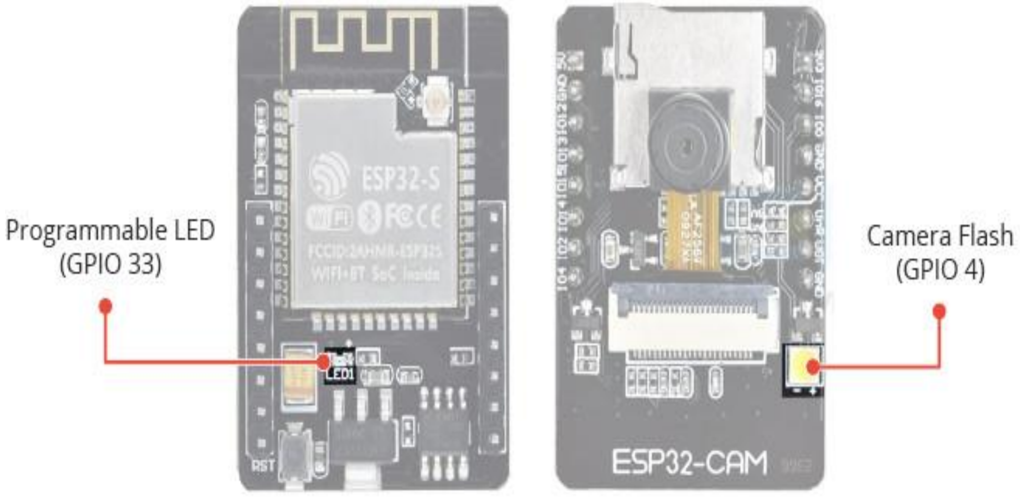


The Antenna

The ESP32-CAM comes with an on-board PCB trace antenna as well as a u.FL connector for connecting an external antenna. An Antenna Selection jumper (zero-ohm resistor) allows you to choose between the two options.

LEDs

The ESP32-CAM has a white square LED. It is intended to be used as a camera flash, but it can also be used for general illumination.



There is a small red LED on the back that can be used as a status indicator. It is user-programmable and connected to GPIO33.

Technical Specifications

To summarize, the ESP32-CAM has the following specifications.

- Processors:
  - CPU: Xtensa dual-core 32-bit LX6 microprocessor, operating at 240 MHz and performing at up to 600 DMIPS
  - Ultra low power (ULP) co-processor
- Memory:
  - 520 KB SRAM
  - 4MB External PSRAM
  - 4MB internal flash memory
- Wireless connectivity:
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
- Camera:
  - 2 Megapixel OV2640 sensor
  - Array size UXGA 1622×1200
  - Output formats include YUV422, YUV420, RGB565, RGB555 and 8-bit compressed data
  - Image transfer rate of 15 to 60 fps
  - Built-in flash LED
  - Support many camera sensors
- Supports microSD card
- Security:

    - IEEE 802.11 standard security features all supported, including WFA, WPA/WPA2 and WAPI
    - Secure boot
    - Flash encryption
    - 1024-bit OTP, up to 768-bit for customers
    - Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)
- Power management:
    - Internal low-dropout regulator
    - Individual power domain for RTC
    - 5μA deep sleep current
    - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

Schematic and Datasheets

For more information on ESP32-CAM, please refer to:

ESP32-CAM Datasheet

ESP32-CAM schematic diagram

OV2640 Camera Datasheet

ESP32-CAM Power Consumption

The power consumption of the ESP32-CAM varies depending on what you're using it for.

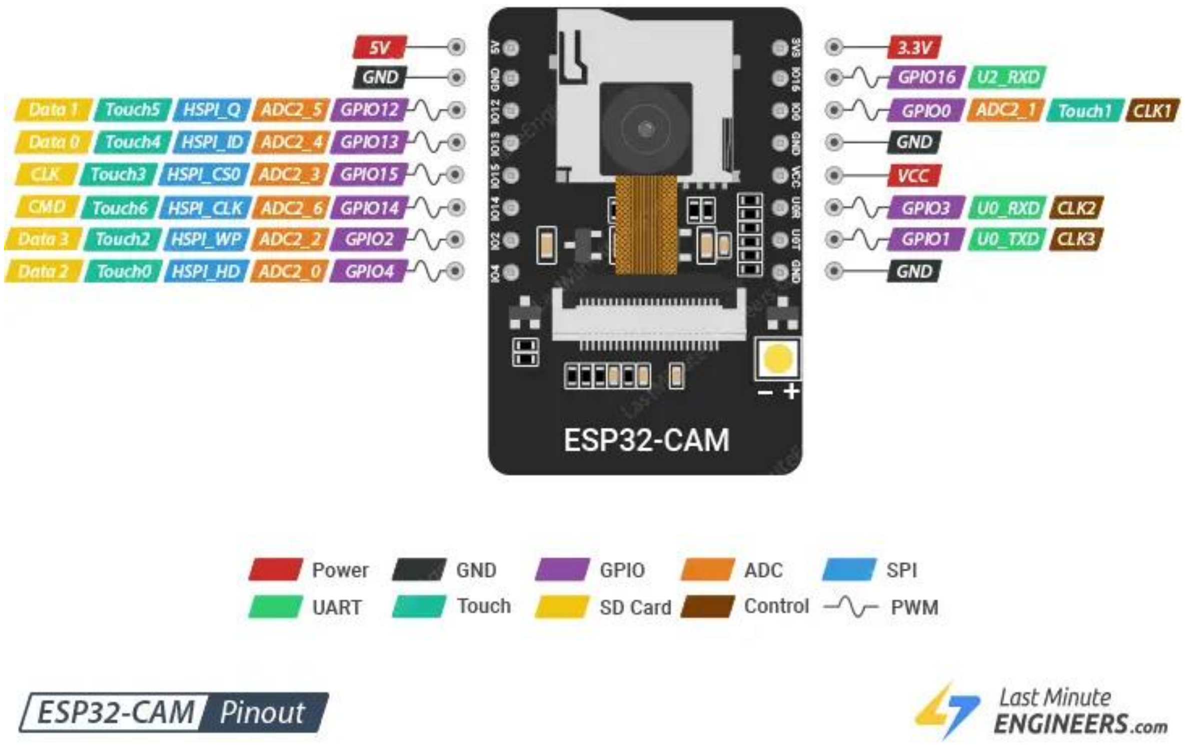
It ranges from 80 mAh when not streaming video to around 100~160 mAh when streaming video; with the flash on, it can reach 270 mAh.

Operation mode	Power Consumption
Stand by	80 mAh

In streaming	100~160 mAh
In streaming with flash	270 mAh

ESP32-CAM Pinout

The ESP32-CAM has 16 pins in total. For convenience, pins with similar functionality are grouped together. The pinout is as follows:



5V
GND
Data 1 Touch5 HSPI_Q ADC2_5 GPIO12
Data 0 Touch4 HSPI_ID ADC2_4 GPIO13
CLK Touch3 HSPI_CS0 ADC2_3 GPIO15
CMD Touch6 HSPI_CLK ADC2_6 GPIO14
Data 3 Touch2 HSPI_WP ADC2_2 GPIO2
Data 2 Touch0 HSPI_HD ADC2_0 GPIO4
3.3V
GPIO16 U2_RXD
GPIO0 ADC2_1 Touch1 CLK1
GND
VCC
GPIO3 U0_RXD CLK2
GPIO1 U0_TXD CLK3
GND
ESP32-CAM
Power
GND
GPIO
ADC
SPI
UART
Touch
SD Card
Control
PWM
ESP32-CAM Pinout
Last Minute ENGINEERS.com

Power Pins There are two power pins: 5V and 3V3. The ESP32-CAM can be powered via the 3.3V or 5V pins. Since many users have reported problems when powering the device with 3.3V, it is advised that the ESP32-CAM always be powered via the 5V pin. The VCC pin normally outputs 3.3V from the on-board voltage regulator. It can, however, be configured to output 5V by using the Zero-ohm link near the VCC pin.

GND is the ground pin.

GPIO Pins The ESP32-S chip has 32 GPIO pins in total, but because many of them are used internally for the camera and the PSRAM, the ESP32-CAM only has 10 GPIO pins available. These pins can be assigned a variety of peripheral duties, such as UART, SPI, ADC, and Touch.

UART Pins The ESP32-S chip actually has two UART interfaces, UART0 and UART2. However, only the RX pin (GPIO 16) of UART2 is broken out, making UART0 the only usable UART on the ESP32-CAM (GPIO 1 and GPIO 3). Also, because the ESP32-CAM lacks a USB port, these pins must be used for flashing as well as connecting to UART-devices such as GPS, fingerprint sensors, distance sensors, and so on.

MicroSD Card Pins are used for interfacing the microSD card. If you aren't using a microSD card, you can use these pins as regular inputs and outputs.

ADC Pins On the ESP32-CAM, only ADC2 pins are broken out. However, because ADC2 pins are used internally by the WiFi driver, they cannot be used when Wi-Fi is enabled.

Touch Pins The ESP32-CAM has 7 capacitive touch-sensing GPIOs. When a capacitive load (such as a human finger) is in close proximity to the GPIO, the ESP32 detects the change in capacitance.

SPI Pins The ESP32-CAM features only one SPI (VSPI) in slave and master modes.

PWM Pins The ESP32-CAM has 10 channels (all GPIO pins) of PWM pins controlled by a PWM controller. The PWM output can be used for driving digital motors and LEDs.

For more information, refer to our comprehensive ESP32-CAM pinout reference guide. This guide also explains which ESP32-CAM GPIO pins are safe to use and which pins should be used with caution.

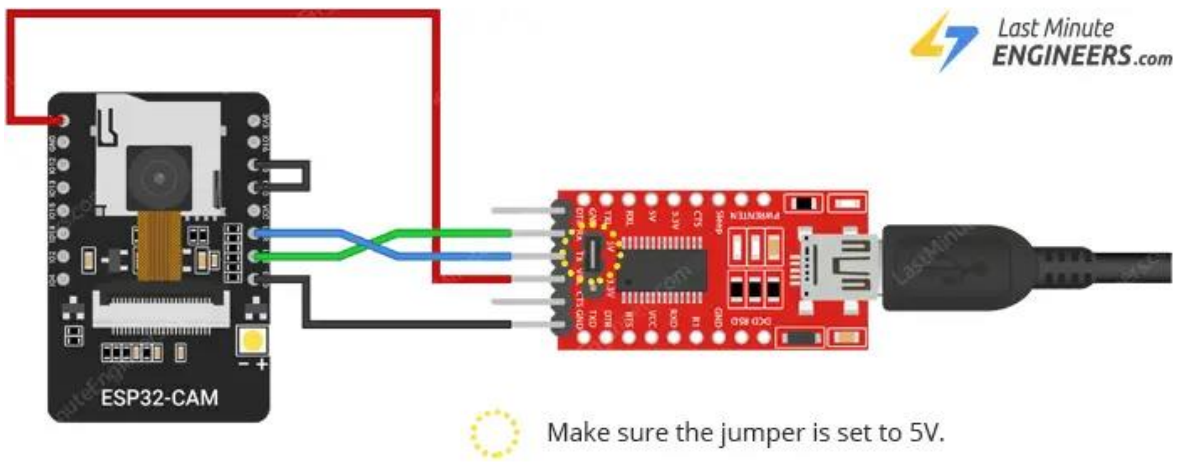
Programming the ESP32-CAM

Programming the ESP32-CAM can be a bit of a pain as it lacks a built-in USB port. Because of that design decision, users require additional hardware in order to upload programs from the Arduino IDE. None of that is terribly complex, but it is inconvenient.

To program this device, you'll need either a USB-to-serial adapter (an FTDI adapter) or an ESP32-CAM-MB programmer adapter.

Using the FTDI Adapter

If you've decided to use the FTDI adapter, here's how you connect it to the ESP32-CAM module.

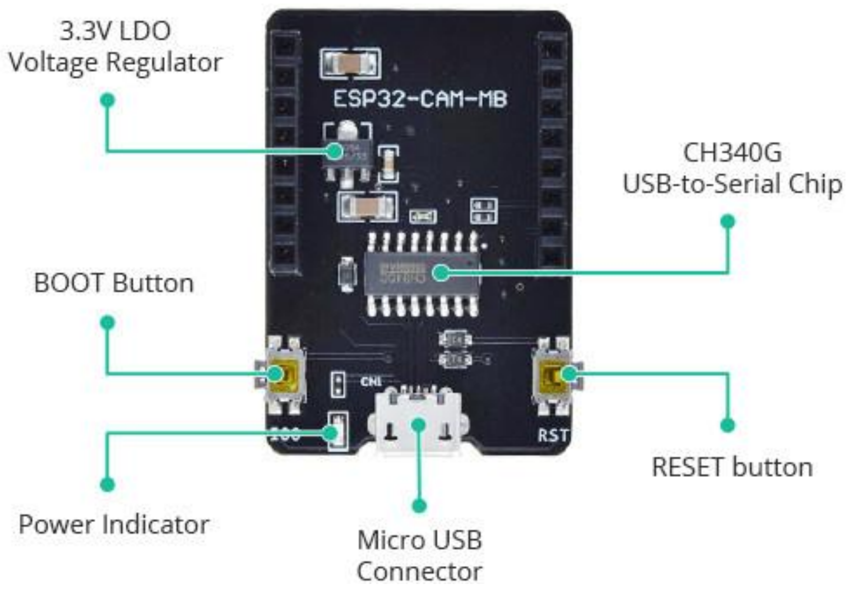


Many FTDI programmers have a jumper that lets you choose between 3.3V and 5V. As we are powering the ESP32-CAM with 5V, make sure the jumper is set to 5V.

Using the ESP32-CAM-MB Adapter (Recommended)

Using the FTDI Adapter to program the ESP32-CAM is a bit of a hassle. This is why many vendors now sell the ESP32-CAM board along with a small add-on daughterboard called the ESP32-CAM-MB.

You stack the ESP32-CAM on the daughterboard, attach a micro USB cable, and click the Upload button to program your board. It's that simple.



The highlight of this board is the CH340G USB-to-Serial converter. That's what translates data between your computer and the ESP32-CAM. There's also a RESET button, a BOOT button, a power indicator LED, and a voltage regulator to supply the ESP32-CAM with plenty of power.

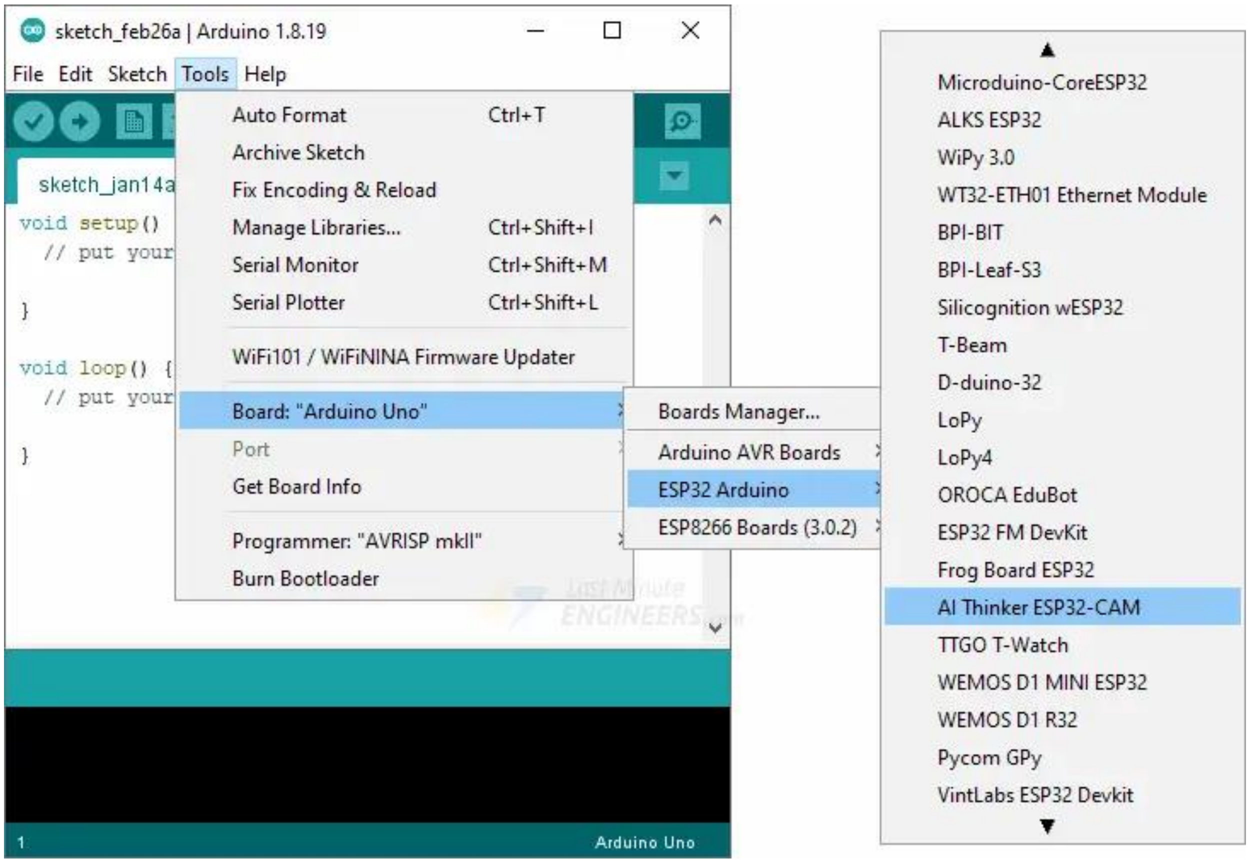
Setting Up the Arduino IDE

Installing the ESP32 Board

To use the ESP32-CAM, or any ESP32, with the Arduino IDE, you must first install the ESP32 board (also known as the ESP32 Arduino Core) via the Arduino Board Manager.

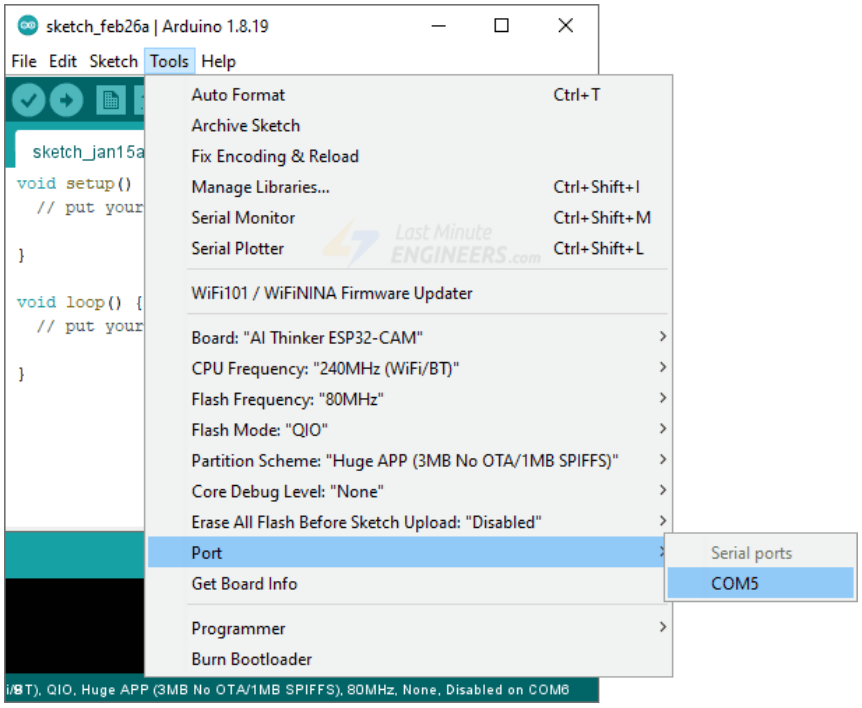
Selecting the Board and Port

After installing the ESP32 Arduino Core, restart your Arduino IDE and navigate to Tools > Board > ESP32 Arduino and select AI-Thinker ESP32-CAM.



```
void setup()
  // put your

}

void loop() {
  // put your

}
```

Now connect the ESP32-CAM to your computer using a USB cable. Then, navigate to Tools > Port and choose the COM port to which the ESP32-CAM is connected.


sketch_feb26a | Arduino 1.8.19
File Edit Sketch Tools Help
sketch_jan15a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Port
Serial ports
COM5
Get Board Info
Programmer
Burn Bootloader

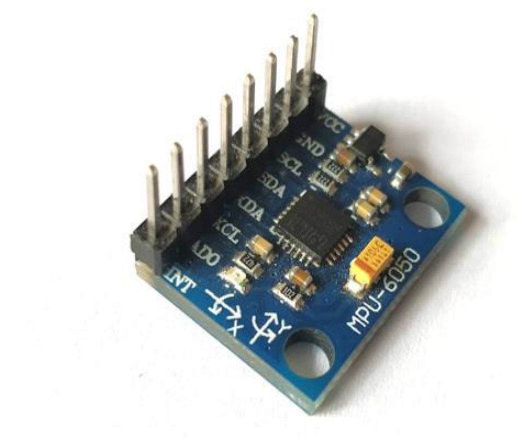
```
void setup() {
  // put your
}

void loop() {
  // put your
}
```

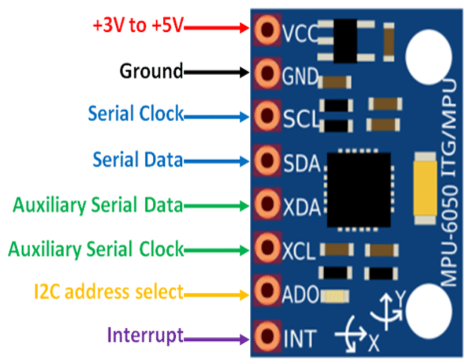
Board: "AI Thinker ESP32-CAM"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Partition Scheme: "Huge APP (3MB No OTA/1MB SPIFFS)"
Core Debug Level: "None"
Erase All Flash Before Sketch Upload: "Disabled"

i/BT), QIO, Huge APP (3MB No OTA/1MB SPIFFS), 80MHz, None, Disabled on COM6

That’s it; the Arduino IDE is now set up for the ESP32-CAM!

MPU6050 Accelerometer and Gyroscope Module





The **MPU6050 module** is a Micro Electro-Mechanical Systems (**MEMS**) which consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion related parameter of a system or object.

MPU6050 Pinout Configuration

Pin Number	**Pin Name**	**Description**
1	Vcc	Provides power for the module, can be +3V to +5V. Typically +5V is used
2	Ground	Connected to Ground of system
3	Serial Clock (SCL)	Used for providing clock pulse for I2C Communication

4	Serial Data (SDA)	Used for transferring Data through I2C communication
5	Auxiliary Serial Data (XDA)	Can be used to interface other I2C modules with MPU6050. It is optional
6	Auxiliary Serial Clock (XCL)	Can be used to interface other I2C modules with MPU6050. It is optional
7	AD0	If more than one MPU6050 is used a single MCU, then this pin can be used to vary the address
8	Interrupt (INT)	Interrupt pin to indicate that data is available for MCU to read.

MPU6050 Features

- MEMS 3-aixs accelerometer and 3-axis gyroscope values combined
- Power Supply: 3-5V
- Communication : I2C protocol
- Built-in 16-bit ADC provides high accuracy
- Built-in DMP provides high computational power
- Can be used to interface with other IIC devices like magnetometer
- Configurable IIC Address
- In-built Temperature sensor

More features and technical specifications be found in the **MPU6050 datasheet** attached at the bottom of the article.

Alternative for MPU6050

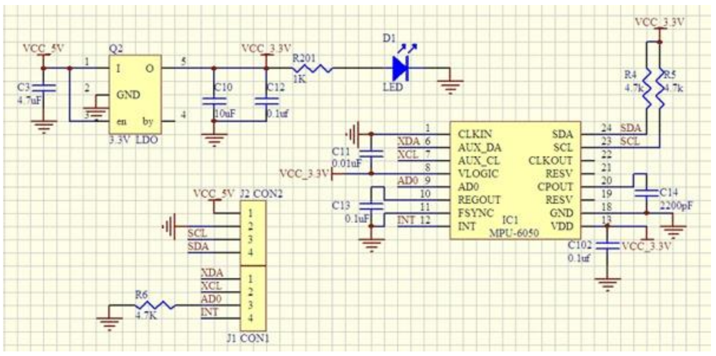
ADXL335 (3-axis accelerometer), ADXL345 (3-axis accelerometer), MPU9250 (9-axis IMU)

Where to Use MPU6050

The MPU6050 is a Micro Electro-Mechanical Systems (**MEMS**) which consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion related parameter of a system or object. This module also has a (DMP) Digital Motion Processor inside it which is powerful enough to perform complex calculation and thus free up the work for Microcontroller.

The module also have two auxiliary pins which can be used to interface external IIC modules like an magnetometer, however it is optional. Since the IIC address of the module is configurable more than one **MPU6050 sensor** can be interfaced to a Microcontroller using the AD0 pin. This module also has well documented and revised libraries available hence it’s very easy to use with famous platforms like Arduino. So if you are looking for a sensor to control motion for your **RC Car**, **Drone**, **Self balancing Robot**, **Humanoid**, **Biped** or something like that then this sensor might be the right choice for you.

How to Use MPU6050 Sensor



The hardware of the module is very simple, it actually comprises of the **MPU6050** as the main components as shown above. Since the module works on 3.3V, a voltage regulator is also used. The IIC lines are pulled high using a 4.7k resistor and the interrupt pin is pulled down using another 4.7k resistor.

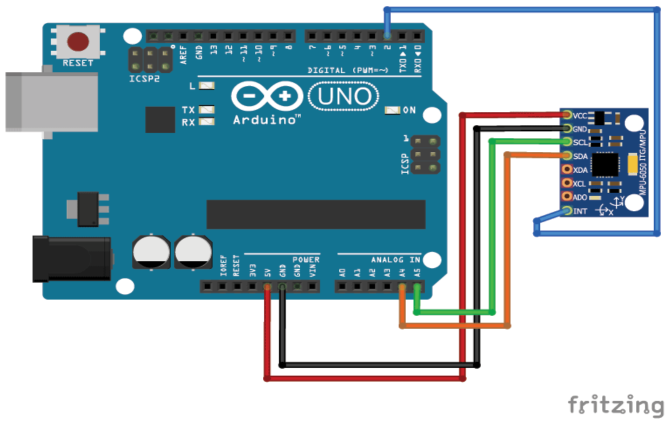
The MPU6050 module allows us to read data from it through the IIC bus. Any change in motion will be reflected on the mechanical system which will in turn vary the voltage. Then the IC has a 16-bit ADC which it uses to accurately read these changes in voltage and stores it in the FIFO buffer and makes the INT (interrupt) pin to go high. This means that the data is ready to be read, so we use a MCU to read the data from this FIFO buffer through IIC communication. As easy as it might sound, you may face some problem while actually trying to make sense of the data. However there are lots of platforms like Arduino using which you can start using this module in no time by utilizing the readily available libraries explained below.

Interfacing MPU6050 with Arduino

It is very easy to **interface the MPU6050 with Arudino**, thanks to the library developed by Jeff Rowberg. You can download the library from the below link

Jeff Rowberg MPU6050 Library for Arudino

Once you have added this library to you Arduino IDE, follow the below schematics to establish an IIC connection between your **Arduino and MPU6050**.



The library provides two example programs, which can be found at File -> Examples -> MPU6050. In these two examples one will give raw values while the other will give optimised values using the DMP. The following data values can be obtained using this example program.

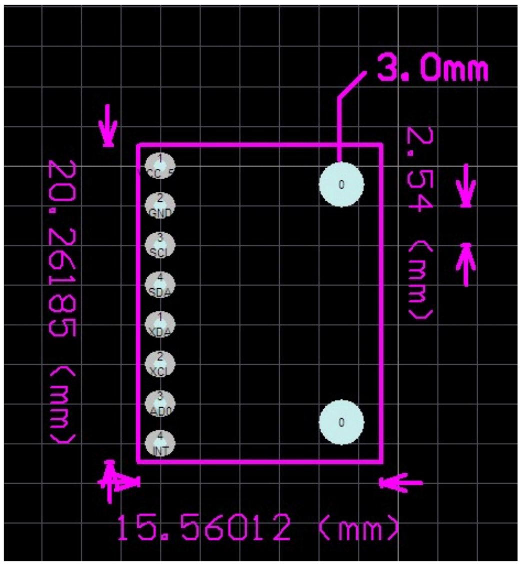
- Quaternion Components [w, x, y, z]
- Euler angles
- Yaw, Pitch, Roll
- Real world Acceleration
- World frame acceleration
- Teapot invent sense Values

Out of all these data, the Yaw, Pitch, Roll us commonly used. However the library is capable of performing more than that and can be used for different purposes. Once the program is uploaded, open serial monitor and set it to 115200 baud rate and you should see the data being printed on the screen.

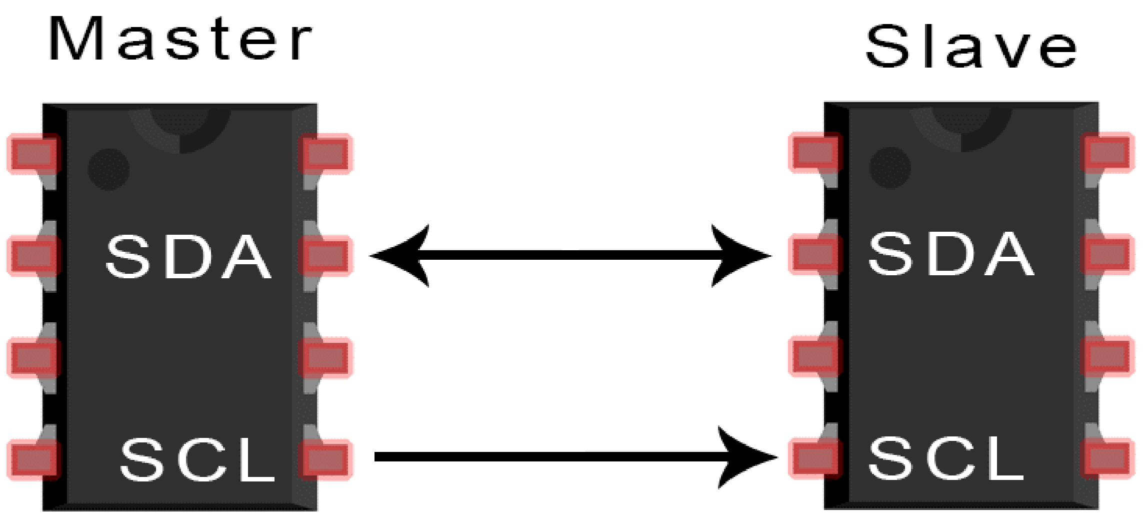
Applications

- Used for IMU measurement
- Drones / Quad copters
- Self balancing robots
- Robotic arm controls
- Humanoid robots
- Tilt sensor
- Orientation / Rotation Detector

2D Model of MPU6050



BASICS OF THE I2C COMMUNICATION PROTOCOL



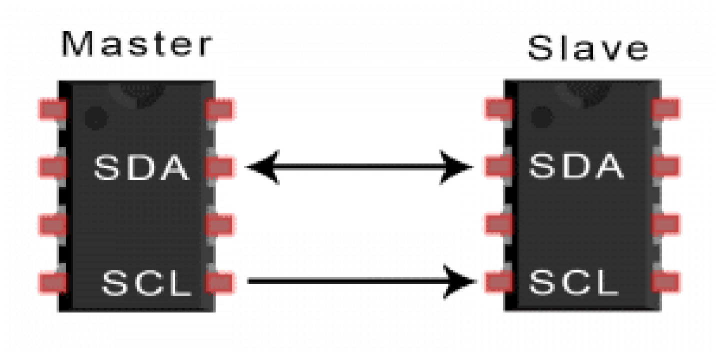
So far, we’ve talked about the basics of SPI communication and UART communication, so now let’s go into the final protocol of this series, the Inter-Integrated Circuit, or I2C.

You’ll probably find yourself using I2C if you ever build projects that use OLED displays, barometric pressure sensors, or gyroscope/accelerometer modules.

INTRODUCTION TO I2C COMMUNICATION

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

Like UART communication, I2C only uses two wires to transmit data between devices:

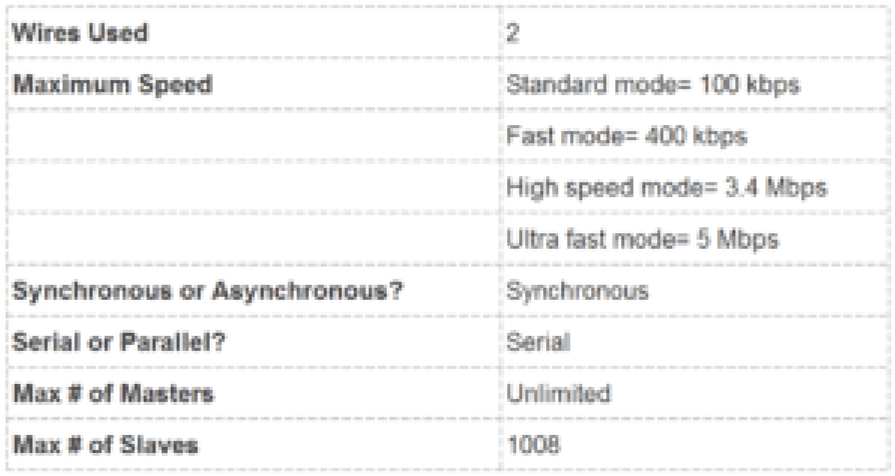


SDA (Serial Data) – The line for the master and slave to send and receive data.

SCL (Serial Clock) – The line that carries the clock signal.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

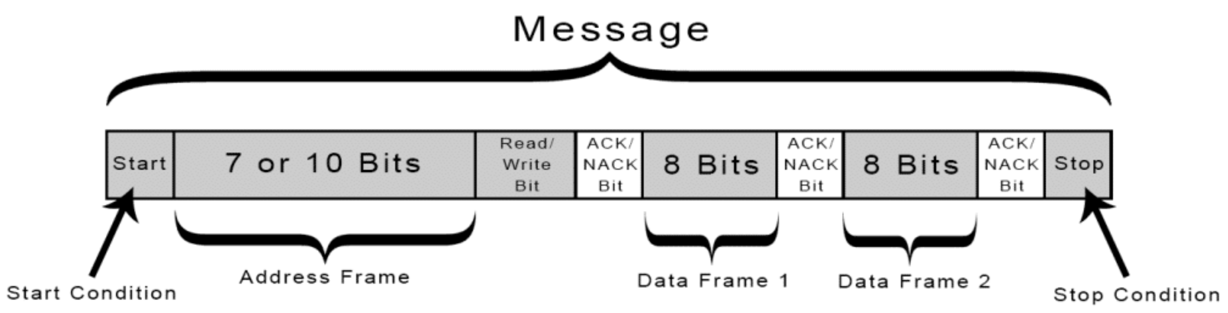
Like SPI, I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.



Wires Used	2
Maximum Speed	Standard mode= 100 kbps
	Fast mode= 400 kbps
	High speed mode= 3.4 Mbps
	Ultra fast mode= 5 Mbps
Synchronous or Asynchronous?	Synchronous
Serial or Parallel?	Serial
Max # of Masters	Unlimited
Max # of Slaves	1008

HOW I2C WORKS

With I2C, data is transferred in *messages.* Messages are broken up into *frames* of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:



Start Condition: The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.

Stop Condition: The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.

Address Frame: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

Read/Write Bit: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

ACK/NACK Bit: Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

ADDRESSING

I2C doesn't have slave select lines like SPI, so it needs another way to let the slave know that data is being sent to it, and not another slave. It does this by *addressing*. The address frame is always the first frame after the start bit in a new message.

The master sends the address of the slave it wants to communicate with to every slave connected to it. Each slave then compares the address sent from the master to its own address. If the address matches, it sends a low voltage ACK bit back to the master. If the address doesn't match, the slave does nothing and the SDA line remains high.

READ/WRITE BIT

The address frame includes a single bit at the end that informs the slave whether the master wants to write data to it or receive data from it. If the master wants to send data to the slave, the read/write bit is a low voltage level. If the master is requesting data from the slave, the bit is a high voltage level.

THE DATA FRAME

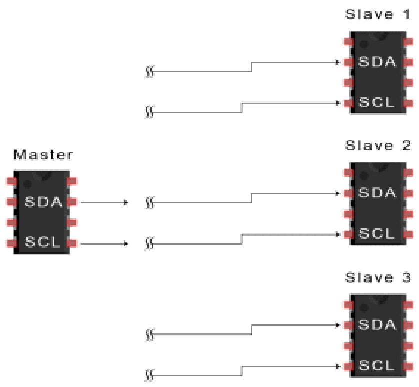
After the master detects the ACK bit from the slave, the first data frame is ready to be sent.

The data frame is always 8 bits long, and sent with the most significant bit first. Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully. The ACK bit must be received by either the master or the slave (depending on who is sending the data) before the next data frame can be sent.

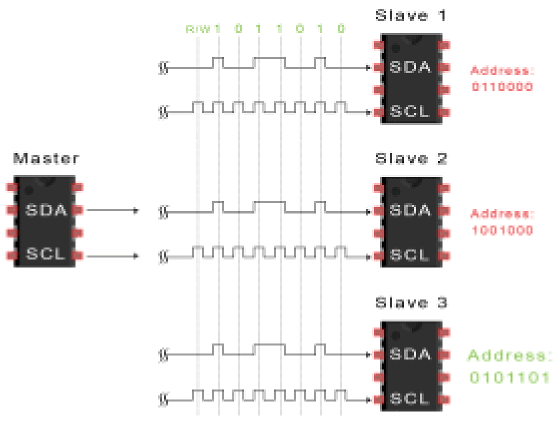
After all of the data frames have been sent, the master can send a stop condition to the slave to halt the transmission. The stop condition is a voltage transition from low to high on the SDA line after a low to high transition on the SCL line, with the SCL line remaining high.

STEPS OF I2C DATA TRANSMISSION

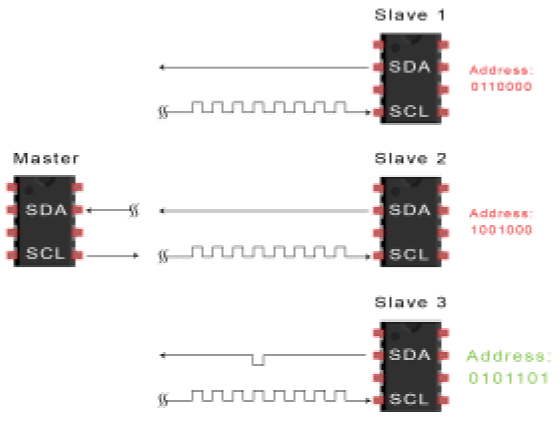
1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level *before* switching the SCL line from high to low:



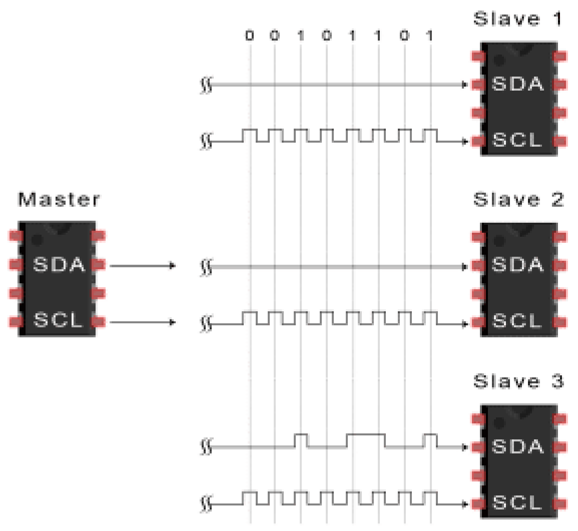
2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit:



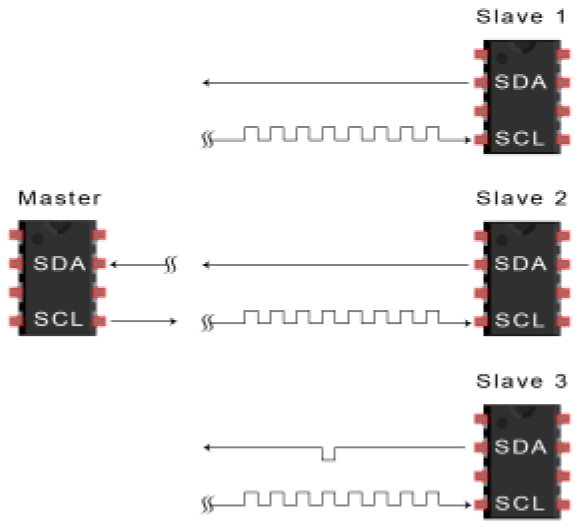
3. Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave’s own address, the slave leaves the SDA line high.



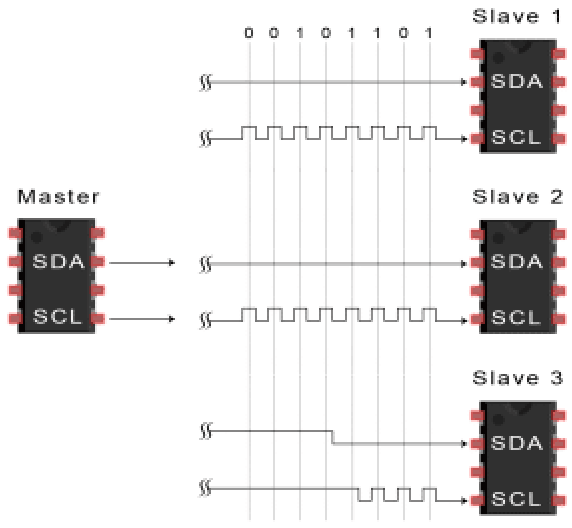
4. The master sends or receives the data frame:



5. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame:

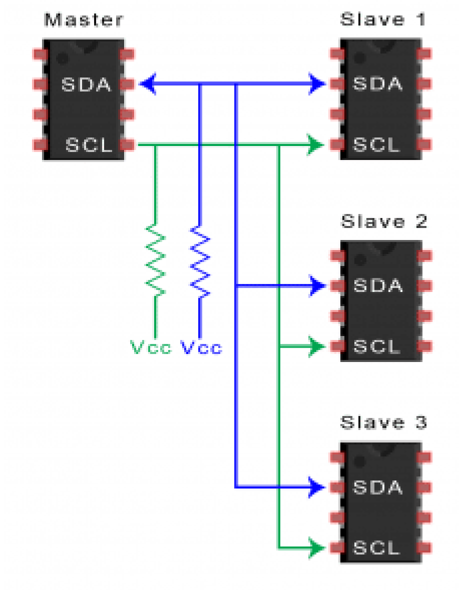


6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:



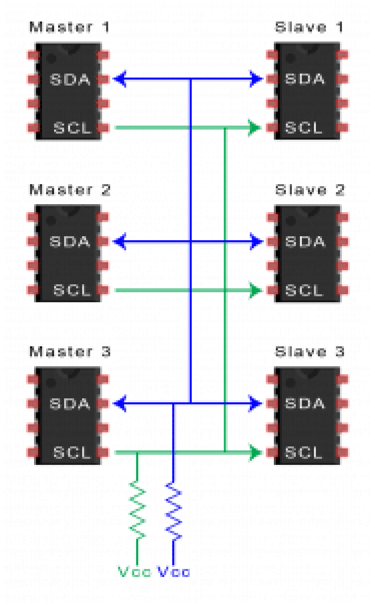
SINGLE MASTER WITH MULTIPLE SLAVES

Because I2C uses addressing, multiple slaves can be controlled from a single master. With a 7 bit address, 128 (2^7) unique address are available. Using 10 bit addresses is uncommon, but provides 1,024 (2^{10}) unique addresses. To connect multiple slaves to a single master, wire them like this, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:



MULTIPLE MASTERS WITH MULTIPLE SLAVES

Multiple masters can be connected to a single slave or multiple slaves. The problem with multiple masters in the same system comes when two masters try to send or receive data at the same time over the SDA line. To solve this problem, each master needs to detect if the SDA line is low or high before transmitting a message. If the SDA line is low, this means that another master has control of the bus, and the master should wait to send the message. If the SDA line is high, then it's safe to transmit the message. To connect multiple masters to multiple slaves, use the following diagram, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:



ADVANTAGES AND DISADVANTAGES OF I2C

There is a lot to I2C that might make it sound complicated compared to other protocols, but there are some good reasons why you may or may not want to use I2C to connect to a particular device:

ADVANTAGES

- Only uses two wires
- Supports multiple masters and multiple slaves
- ACK/NACK bit gives confirmation that each frame is transferred successfully
- Hardware is less complicated than with UARTs
- Well known and widely used protocol

DISADVANTAGES

- Slower data transfer rate than SPI
- The size of the data frame is limited to 8 bits
- More complicated hardware needed to implement than SPI

Thanks for reading! Hope you learned something from this series of articles on electronic communication protocols. In case you haven’t read them already, part one covers the SPI communication protocol, and part two covers UART driven communication.

If you have any questions or have anything to add, feel free to leave a comment below. And be sure to subscribe to get more articles like this in your inbox!

Future Work

What is a Kalman Filter?

A Kalman Filter is a mathematical algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. Named after Rudolf E. Kálmán, the primary developer of its theory, the Kalman Filter has become a fundamental tool in the field of control systems and time series analysis.

The Kalman Filter is an iterative process that estimates the state of a dynamic system from a series of incomplete and noisy measurements. It is recursive, meaning that it can run in real-time using only the current input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

How Does the Kalman Filter Work?

The Kalman Filter operates in two steps: the "predict" or "time update" phase and the "update" or "measurement update" phase.

Predict Phase

In the predict phase, the Kalman Filter uses the state from the previous time step to produce estimates of the current state. This prediction includes the estimation of the system's state variables and the uncertainty of the estimate. The uncertainty is often expressed as a covariance matrix, which is a measure of the "spread" or the expected accuracy of the prediction.

Update Phase

During the update phase, the current prediction is combined with the current observation to refine the state estimate. This step adjusts the predicted state by a factor proportional to the difference between the actual measurement and the prediction. The Kalman Filter uses the covariance matrix to weigh the accuracy of the prediction against the accuracy of the new measurement, thus updating the state estimate and its uncertainty.

These two phases are repeated in a loop, with each iteration refining the estimates. This process allows the filter to react to new measurements and improve the estimate over time, which is why it is particularly useful for systems where the measurements are uncertain or vary over time.

Applications of the Kalman Filter

The Kalman Filter has a wide range of applications in various fields:

- Navigation and Control Systems: It is extensively used in aerospace for trajectory estimation of aircraft and spacecraft, GPS navigation, and robotics.
- Economics and Finance: In econometrics, the Kalman Filter is used for signal extraction in time series analysis, such as separating a signal that evolves over time from "noise".
- Engineering: It is used for sensor fusion, where it combines data from various sensors to compute the best estimate of the state of interest.
- Computer Vision: The Kalman Filter can track moving objects in video streams or predict the position of a moving object.

Advantages and Limitations

The Kalman Filter is advantageous because it is a linear estimator that is optimal under the assumption that the errors are Gaussian. It is computationally efficient, which allows it to run in real-time applications, and it can handle cases where the noise statistics are not fully known.

However, the Kalman Filter has limitations. It assumes that the process and measurement noise are both Gaussian and white, and that the system dynamics are linear. For systems that do not meet these assumptions, extensions to the Kalman Filter, such as the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF), have been developed to handle non-linear systems.

Conclusion

The Kalman Filter is a powerful tool that has revolutionized the field of estimation and control. Its ability to provide optimal estimates in the presence of uncertain data has made it indispensable in many modern technological systems. Despite its limitations, the Kalman Filter remains a cornerstone of signal processing and control theory, with ongoing research extending its capabilities to a wider range of applications.

What is a Madgwick Filter?

A Madgwick filter is a type of sensor fusion algorithm that can estimate the orientation of a device using data from an inertial measurement unit (IMU) and optionally a magnetometer. It is based on an optimization method that minimizes the difference between the measured acceleration and magnetic field vectors and the expected vectors in a common reference frame. The result is a quaternion that represents the rotation from the reference frame to the device frame. The Madgwick filter can also integrate the angular velocity data from the gyroscope to improve the accuracy and stability of the orientation estimation. The Madgwick filter is popular for its simplicity, efficiency, and robustness. It can be used for applications such as head tracking, camera stabilization, and robot navigation

References

- https://hackr.io/blog/what-is-arduino
- **https://lastminuteengineers.com/getting-started-with-esp32-cam/**
- **https://components101.com/sensors/mpu6050-module**
- **https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/#google_vignette**
- **https://deepai.org/machine-learning-glossary-and-terms/kalman-filter**