**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.

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 be the output of the sensor.

R is Rotation matrix.

If we assume that accelerometer has no acceleration and then we would have:

So

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 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;

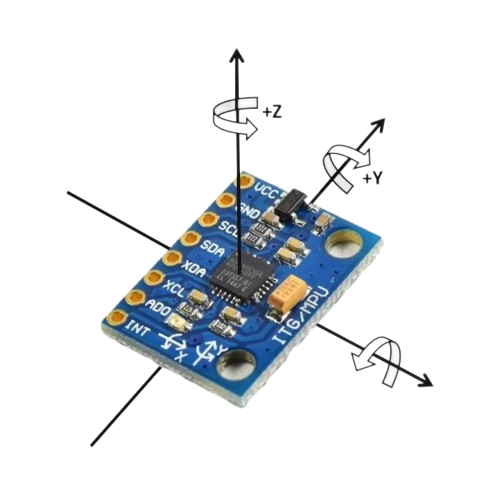
  return na;

}

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

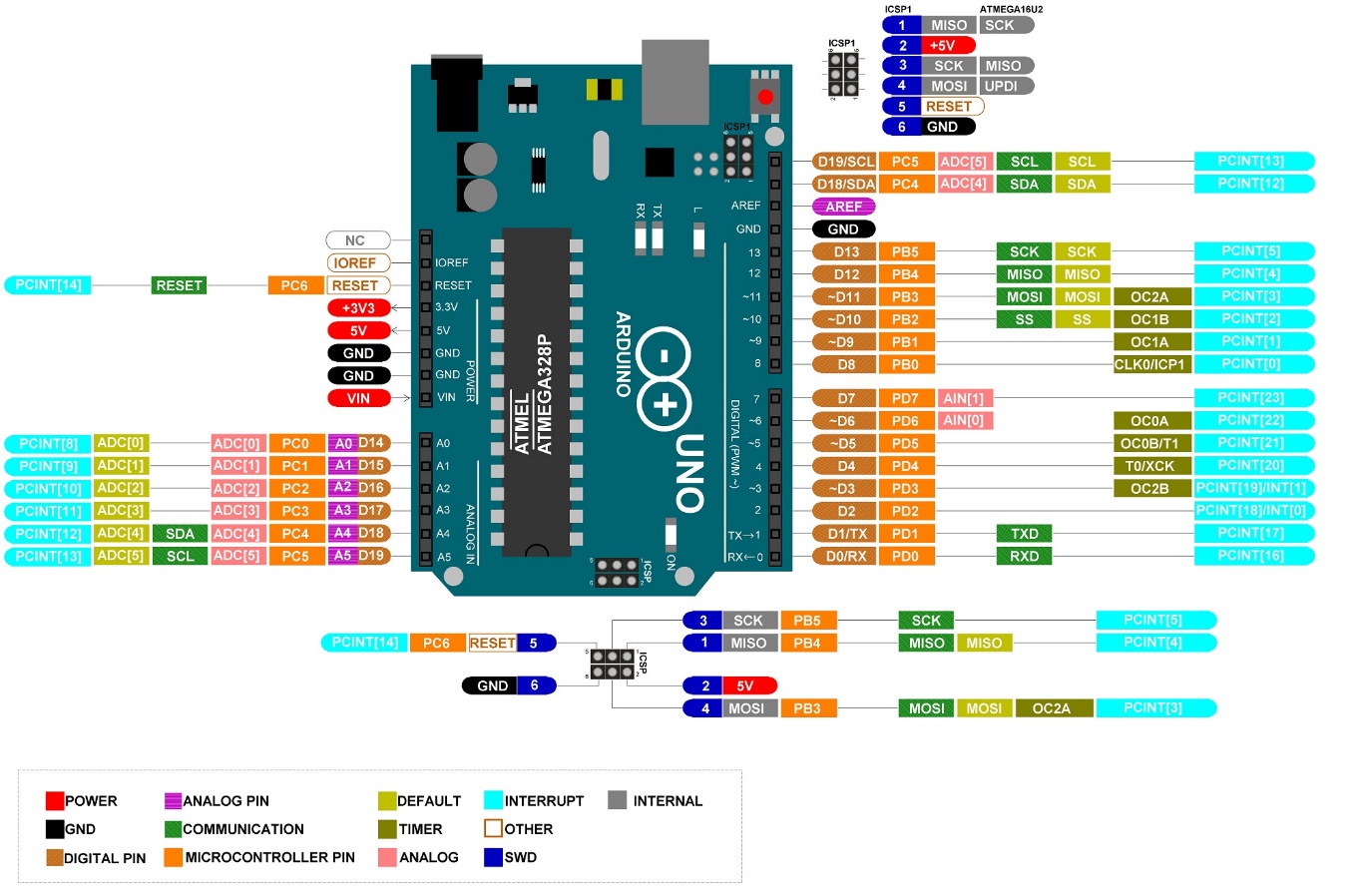
**Hardware of IMU data acquisition system**

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.

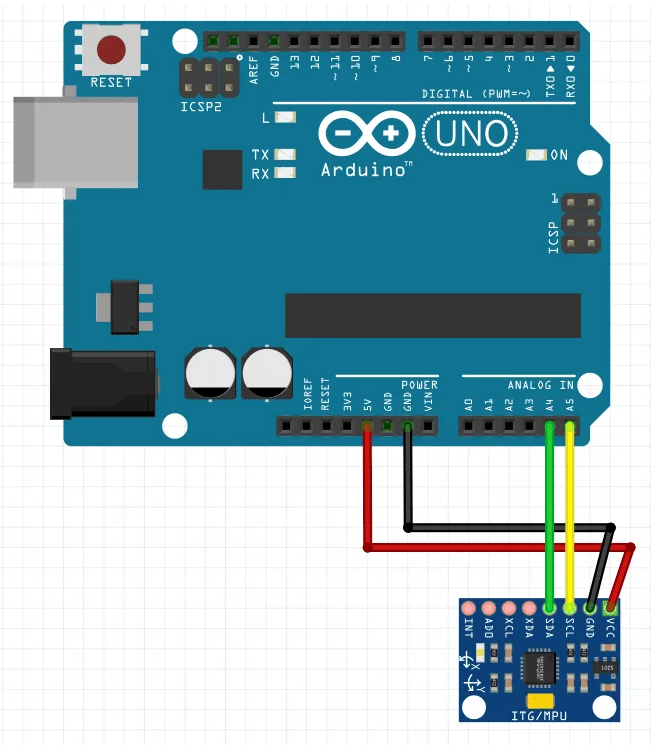
****

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.



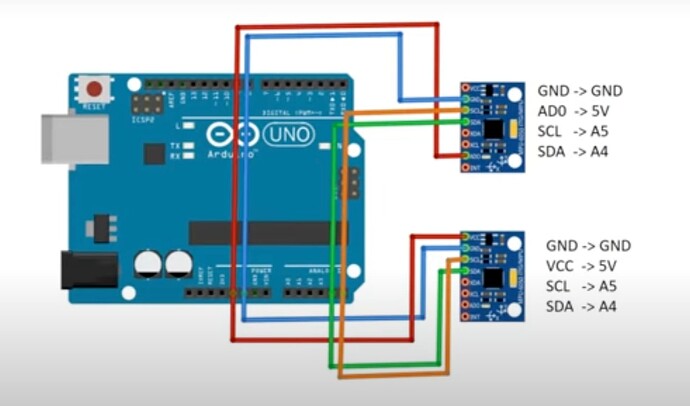
But as I mentioned we have two IMUs and for connecting two of them to Arduino board we should connect second IMU also to A4 and A5 pin but with this difference that AD0 pin of the second IMU should be pulled up to VCC and its state should be HIGH. This is done to make the I2C address of the two IMUs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

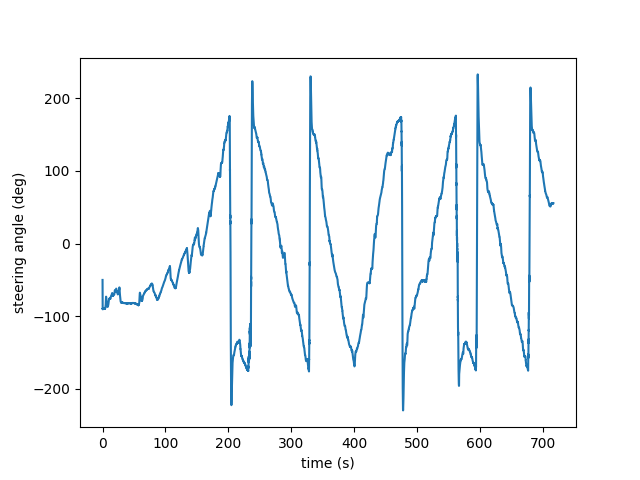
The circuit diagram of the micro switch is provided in camera setup section, it is just mentionable that it would be connected to pin number 11 of the Arduino.

The final wiring is like this.

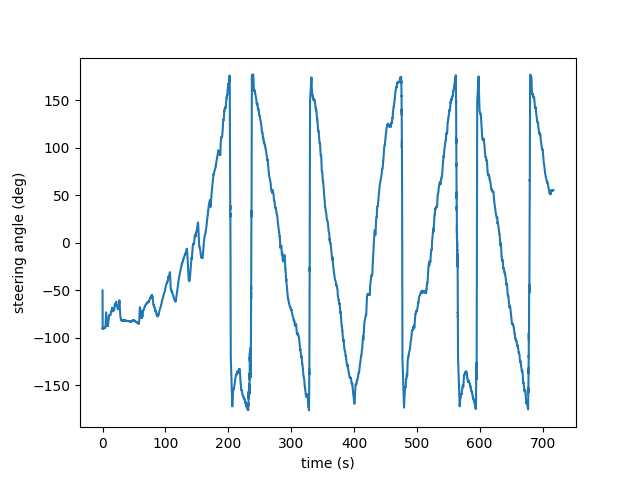


**Results**:

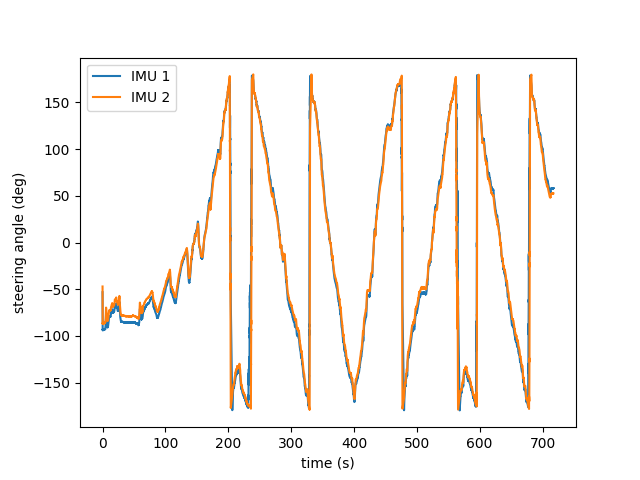
Here are some plots which are obtained based on the data acquired by the IMU sensors.



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).

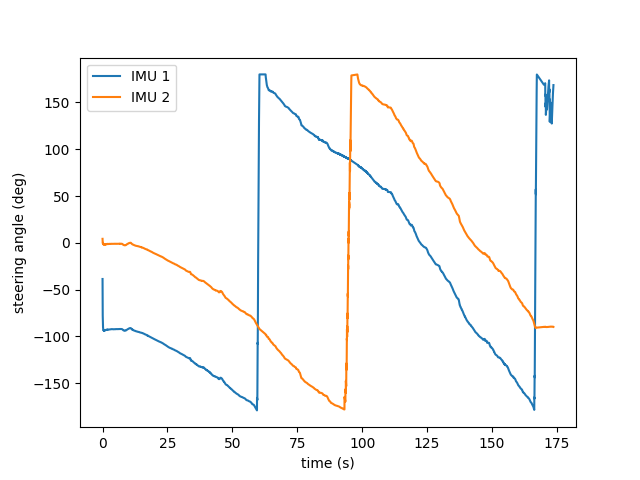


The average of the output angle of the two IMU sensors, with outliers removed.

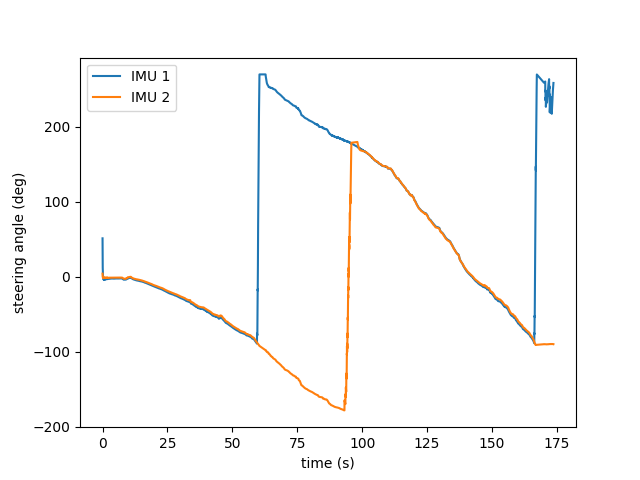


Comparison of the output angles of the two IMU sensors with outliers removed.

As you can see from plots especially the first plot when IMU gets near (it normally starts from for example 170 deg) its output data is not like a continuous function anymore and jumps from +180 to -180 deg. For having a valid data in the range of we can install two IMUs with 90 degree phase difference and use the other IMU’s data with consideration of the known offset between them. The output of two IMUs with 90 degree phase difference would be like this.



And then if we Add all the data to ninety degrees we can see that two graphs would almost overlap with this difference that they experience discontinuity it different time and positions. Therefore, we can use from the data of each of them at the point where the other is discontinuous. Here is the plot of two IMUs that one is added 90 degrees.



Hence using two or three IMUs and fusing them is better solution for having more accurate data.