```
/*
CODE REVIEW
- Sasanka Kuruppuarachchi

Project - "https://github.com/SasaKuruppuarachchi/miniDAQ"

TSR_2018_MINIDAQ
Designed for Teensy 3.2
By Sasanka Kuruppuarachchi
Cheif electrical Engineer
Team SHARK racing
UOM
*/

/*
Lets devide the code into 8 parts
1. Kalman implementation
2. Initialize Hardware
3. GPS related methods
4. IMU related methods
5. Data logger
6. Display
7. Setup
8. Loop
*/

// PART 1 - KALMAN IMPLEMENTATION
**************************************************************************

#include <MatrixMath.h>                                    //import library for matrix calculations

# define cov_X 0.025495
# define cov_y 0.04467                                     //experimental covariance values (ms-2)2

/*
Volatile varient is used for following variables as they are updated
at higher freq. and to reduce data loss across threads
*/
volatile double posCov = 1.1;
volatile double accCov = 0.08;                             //GPS and ACC covariances
volatile double estNpos,estEpos,estNvel,estEvel;           //estimation step variables

volatile double timeStamp;
volatile int timer2;                                        //Timestamp variables for predict step

class Kalman {                                              //Class definition of kalman filter

  private:                                                  //private decleration of state variables

  //predict
  double currentState[2][1];                                // state vector [ p
                                                            //                v ]
  double u;                                                 //accelerometer reading
  double P[2][2];                                          //estimate error (initialize identity)
  double Q[2][2];                                          //Accelerometer noice covarience (ms-2)2
  double A[2][2];                                          //state transition metrix. Predicts next
  step with no external inputs
  double B[2][1];                                          //Control metrix. adds the influence of
  external influence
  double currentTimeStampSec;                              // Current timestamp

  //update
  double z[2][1];                                          //GPS reading for pos, vel
  double H[2][2];                                          //transform pred. in form of GPS reading
  (identity as degrees are converted to m in API for simplicity)
  double R[2][2];                                          //GPS error covarience +/- m
  double K[2][2];                                          //Kalman gain

  double u1[2][1];                                         //Intermdiary matrises
  double zi[2][1];
  double AT[2][2];
```

```
double s[2][2];
int err;

public:

Kalman(double initialPos,double posCov, double accCov, double currentTimeStamp)
//constructor
:currentTimeStampSec(currentTimeStamp){

  currentState[0][0] = initialPos;
  currentState[1][1] = 0.0;                       // init current state matrix

  Q[0][0] = accCov;
  Q[1][1] = accCov;

  R[0][0] = posCov;
  R[1][1] = posCov;                               //init covariances

  Serial.println(currentTimeStampSec);
  Matrix.Print((double*)currentState, 2, 1, "C"); //init acnowladgement

}

void recreateA(double deltaSec){                  //update state transition metrix
  A[0][0] = 1.0; A[0][1] = deltaSec;
  A[1][0] = 0.0; A[1][1] = 1.0;

}

void recreateB(double deltaSec){                  //update control metrix
  B[0][0] = 0.5*deltaSec*deltaSec;
  B[1][0] = deltaSec;

}

//predict step (Measurement) using IMU outputs
void predict(double accThisAxis, double timeStampNow){
  double deltaT = timeStampNow - currentTimeStampSec;                    //interval
  between predict steps
  recreateA(deltaT);
  recreateB(deltaT);
  u = accThisAxis;

  //State Prediction
  //currentState_n = A*currentState_n-1 + B*u
  Matrix.Multiply((double*)A, (double*)currentState, 2, 2, 1, (double*)u1);
  Matrix.Scale((double*) B, 2, 1, u);
  Matrix.Add((double*) u1, (double*) B, 2, 1, (double*) currentState);

  //Covariance Prediction
  //P = A*P*AT + Q
  Matrix.Multiply((double*)A, (double*)P, 2, 2, 2, (double*)P);
  Matrix.Transpose((double*)A, 2, 2, (double*)AT);
  Matrix.Multiply((double*)P, (double*)AT, 2, 2, 2, (double*)P);
  Matrix.Add((double*) P, (double*) Q, 2, 2, (double*) P);

  currentTimeStampSec = timeStampNow;
}

// update step (Absolute) Using GPS outputs
void updates(double pos, double velocity){
  z[0][0] = pos;
  z[1][0] = velocity;

  //kalman gain
  //K = P*(P+R)-1
  Matrix.Add((double*) P, (double*) R, 2, 2, (double*) s);
  err = Matrix.Invert((double*) s, 2);
  if (err == 0)Serial.println("No inverse");
  Matrix.Multiply((double*)P, (double*)s, 2, 2, 2, (double*)K);
```

```
    //State update
    //estState = preState + K( z - preState)
    Matrix.Subtract((double*) z, (double*) currentState, 2, 1, (double*) z);
    Matrix.Multiply((double*)K, (double*)z, 2, 2, 1, (double*)zi);
    Matrix.Add((double*) currentState, (double*) zi, 2, 1, (double*) currentState);

    //Covariance Update
    // P = P + KP
    double Ps[2][2];
    Matrix.Multiply((double*)K, (double*)P, 2, 2, 2, (double*)Ps);
    Matrix.Add((double*) P, (double*) Ps, 2, 1, (double*) P);
  }

  double getPosition(){
    return currentState[0][0];                        //get real poition
  }

  double getVelocity(){
    return currentState[1][0];                        //Get real velocity
  }
};

/*
two separate kE and kN kalman filter objects are created
for East and North GPS axis
For this application 2D estimation is adequat as we assume race track is relatively flat
*/
Kalman kE(GPSlongitude,posCov,accCov,millis()/1000.0);
Kalman kN(GPSlatitude,posCov,accCov,millis()/1000.0);
//global timestamp
double TimeNowSec;

// PART 2 - INITIALIZE
*********************************************************************************

//.....................................................Initialize threads
#include <TeensyThreads.h>
Threads::Mutex Serial_lock;                           //Parallel threads are initiated to
eliminate data loss

/*
GPS works at 10 Hz and IMU at 100 Hz
two threads are created for reading IMU and GPS
to eliminate missing of each data output while trying to
read one in another thread
- each output is then buffered until Kalman filter uses them to oredict
*/
int GPSthreadID, IMUthreadID;

//.....................................................Initialize SD data logger
#include <SD.h>                                       //Load SD card library
#include<SPI.h>                                       //Load SPI Library
File mySensorData;                                    //Data object you will write your sesnor
data to

const int chipSelect = BUILTIN_SDCARD;                // Select chipset setting

char fileName[] = "00_00_00.kml";                     //main File name variable

char fileNameG[] = "00G00G00.kml";                    //Secondary file name

double degWhole,deg,degDec;
double degWholel,degl,degDecl;                        //Lat, Lon calculation variables for KML
wrapper

//....................................................Init the button to start logging
#include <Bounce2.h>

#define BUTTON_PIN 24
Bounce logButton = Bounce();
volatile int logButtonState = 0;
```

```
int LBSprev = 0;

//...................................................Init GPS module
//Adafruit Ultimate GPS featherwing
https://learn.adafruit.com/adafruit-ultimate-gps-featherwing
#include <Adafruit_GPS.h>                                    //Install the adafruit GPS library
#define mySerial Serial3

Adafruit_GPS GPS(&mySerial);                                 //Create the GPS Object

#define GPSECHO  false                                       //Set echo to false

String NMEA1;                                                //Variable for first NMEA sentence
String NMEA2;                                                //Variable for second NMEA sentence
volatile char c;                                             //to read characters coming from the GPS

volatile int lHour,lMinute,lSec,GPSday,GPSmonth,GPSyear,GPSquality,GPSsatellites;
volatile bool GPSfix = 0;
volatile double GPSlatitude = 721226.3;
volatile double GPSlongitude = 8853627.2;  //home
volatile double GPSspeed,GPSangle,GPSaltitude,GPSspeedN,GPSspeedE;
volatile char GPSlat,GPSlon;
volatile double GPSlatitude1 = 721226.3;
volatile double GPSlongitude1 = 8853627.0;

/*
Volatile varient is used for following variables as they are updated
at higher freq. and to reduce data loss across threads
*/

#define earthRadiusKm 6378.0                                  //Initialize variables

//....................................................Init IMU BNO055
//AbsOrient IMU
https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview
#include <Wire.h>                                             //import I2C library
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

/* Set the delay between fresh samples */
#define BNO055_SAMPLERATE_DELAY_MS (1)
Adafruit_BNO055 bno = Adafruit_BNO055(55);

/* Also send calibration data for each sensor. */
uint8_t sys, gyro, accel, mag = 0;
volatile double roll,pitch,yaw;
volatile double gDot[2];                                      //acceleration in g in Y & X
volatile double gACC[2];
volatile int Nx,Ny;
volatile double accN,accE;                                    //acceleration in N & E

/*
Volatile varient is used for following variables as they are updated
at higher freq. and to reduce data loss across threads
*/

#define BNO055_SAMPLERATE_DELAY_MS (1)                         // Set samplerate to 100 Hz

//....................................................Init Display
//Monochrome 128x64 OLED Display Module
https://learn.adafruit.com/1-5-and-2-4-monochrome-128x64-oled-display-module/overview

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Using hardware SPI
#define OLED_CLK 13
#define OLED_MOSI 11
```

```
// These are neede for hardware SPI
#define OLED_CS 5
#define OLED_RESET 6
#define OLED_DC 2


Adafruit_SSD1306 display(OLED_DC, OLED_RESET, OLED_CS); //init display object


// PART 3 - GPS RELATED METHODS
*************************************************************************
//Get GPS time refarance
void timeCorrection(){
  if ((GPS.minute + 30) > 60) {
    lMinute = GPS.minute + 30-60;
    lHour = GPS.hour+6;
    lSec = GPS.seconds;
  }
  else{
    lHour = GPS.hour+5;
    lMinute = GPS.minute + 30;
    lSec = GPS.seconds;
  }
  GPSday = GPS.day;
  GPSmonth = GPS.month;
  GPSyear = GPS.year;
  GPSfix = GPS.fix;
  GPSquality = GPS.fixquality;
}


boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy



#ifdef __AVR__
// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect) {
  c = GPS.read();
  // if you want to debug, this is a good time to do it!
#ifdef UDR0
  if (GPSECHO)
    if (c) UDR0 = c;
    // writing direct to UDR0 is much much faster than Serial.print
    // but only one character can be written at a time.
  #endif
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}
#endif //#ifdef__AVR__

uint32_t timer = millis();

//convert cordinates to m
double degreeToM(double LatorLan){
  return LatorLan*PI/180.0*earthRadiusKm*10.0;
}
```

```
Convert m to cordinates
double MtoDegree(double in){
  return in/PI*180.0/earthRadiusKm/10.0;
}


void readGPS(){

  // When not using the interrupt above, you'll
  // need to 'hand query' the GPS
  if (! usingInterrupt) {
    // read data from the GPS in the thread
    c = GPS.read();
    if (GPSECHO)
      if (c) Serial.print(c);
  }

  // if a sentence is received check the checksum
  if (GPS.newNMEAreceived()) {
    if (!GPS.parse(GPS.lastNMEA()))                  // this also sets the newNMEAreceived()
    flag to false
      return;                                        // if fail to parse a sentence, wait for
      another
  }

  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis())  timer = millis();

  // Set spamplerate to 10Hz
  if (millis() - timer > 100) {
    timeCorrection();                                // Get time

    if (GPSfix) {
      GPSlatitude = degreeToM(GPS.latitude); GPSlat = GPS.lat;
      GPSlongitude = degreeToM(GPS.longitude); GPSlon = GPS.lon;
      GPSspeed = GPS.speed*0.514444;
      GPSangle = GPS.angle;
      GPSaltitude = GPS.altitude;
      GPSsatellites = GPS.satellites;
      GPSspeedN = (GPSlatitude-GPSlatitude1)/(millis() - timer)*1000;
      GPSspeedE = (GPSlongitude-GPSlongitude1)/(millis() - timer)*1000;
      GPSlatitude1=GPSlatitude;
      GPSlongitude1=GPSlongitude;
    }
    timer = millis(); // reset the timer
  }
}

// Serial output for debugging
void GPSserial() {

  Serial.print("\n\nTime: ");
  Serial.print(lHour, DEC); Serial.print(':');
  Serial.print(lMinute, DEC); Serial.print(':');
  Serial.print(lSec, DEC); Serial.print('.');
  Serial.print("Date: ");
  Serial.print(GPSday, DEC); Serial.print('/');
  Serial.print(GPSmonth, DEC); Serial.print("/20");
  Serial.println(GPSyear, DEC);
  Serial.print("Fix: "); Serial.print((int)GPSfix);
  Serial.print(" quality: "); Serial.println((int)GPSquality);

  Serial.println(usingInterrupt);

  if (GPSfix) {
    Serial.print("Location: ");
    Serial.print(GPSlatitude, 8); Serial.print(GPSlat);
    Serial.print(", ");
    Serial.print(GPSlongitude, 8); Serial.println(GPSlon);

    Serial.print("Speed (ms-1): "); Serial.println(GPSspeed);
```

```
    Serial.print("Angle: "); Serial.println(GPSangle);
    Serial.print("Altitude: "); Serial.println(GPSaltitude);
    Serial.print("Satellites: "); Serial.println((int)GPSsatellites);Serial.println();
  }
}

// PART 4 - IMU RELATED METHODS
*************************************************************************
//Get accelaration x,y
void gMap(void){
  imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL); //Init eular
  vectors
  // Possible vector values can be:
  // - VECTOR_ACCELEROMETER - m/s^2
  // - VECTOR_MAGNETOMETER  - uT
  // - VECTOR_GYROSCOPE     - rad/s
  // - VECTOR_EULER         - degrees
  // - VECTOR_LINEARACCEL   - m/s^2
  // - VECTOR_GRAVITY       - m/s^2

  gACC[0]= euler.x()/9.81;
  gACC[1]= euler.y()/9.81;                                                //acceleration
  in g in Y & X

  accN = euler.x()*sin(roll* PI / 180.0) + euler.y()*cos(roll* PI / 180.0);
  accE = euler.x()*cos(roll* PI / 180.0) - euler.y()*sin(roll* PI / 180.0);  //acceleration
  in N & E
  //got out raw values to calculate covariance
}

// Get orientation roll pitch yaw and convert to Nx, Ny angular rates
void getOri(void){

  bno.getCalibration(&sys, &gyro, &accel, &mag);
  sensors_event_t event;
  bno.getEvent(&event);

  roll = (double)event.orientation.x;
  pitch = (double)event.orientation.y;
  yaw = (double)event.orientation.z;

  Nx = 99 + 29*sin(roll* PI / 180.0);
  Ny = 34 + 29*cos(roll* PI / 180.0);

}

// Serial output for Debugging
void IMUserial(){
  //****************************Acc*********************************
  Serial.print("X: ");
  Serial.println(abs(gDot[0]));
  Serial.print("Y: ");
  Serial.println(abs(gDot[1]));

  //***************************ori************************************


  Serial.print(F("Orientation: "));
  Serial.print(roll);
  Serial.print(F(" "));
  Serial.print(pitch);
  Serial.print(F(" "));
  Serial.print(yaw);
  Serial.println(F(""));
}


// PART 5 - SD DATA LOGGER
*************************************************************************
//Genarate KML File name
void getFileName(){
```

```
  sprintf(fileName, "%02d_%02d_%02d.kml", lHour,lMinute,lSec);
  sprintf(fileNameG, "%02dG%02dG%02d.csv", lHour,lMinute,lSec);
}

void logData(){
  if(GPS.fix==1) {                                        //Only save data if we have a fix
    mySensorData = SD.open(fileNameG, FILE_WRITE);
    mySensorData.print(gACC[0]);mySensorData.print(",");mySensorData.println(gACC[1]);
    mySensorData.close();

    mySensorData = SD.open(fileName, FILE_WRITE);

    //in .kml longitude comes first
    degWholel = double(int(MtoDegree(estEpos)/100.0));
    degDecl=(MtoDegree(estEpos)-degWholel*100.0)/60.0;
    degl = degWholel + degDecl;
    if (GPSlon=='W') {                                    //If you are in Western
    Hemisphere, longitude degrees should be negative
     degl= (-1)*degl;
    }
    mySensorData.print(degl,10);                          //writing decimal degree longitude
    value to SD card
    mySensorData.print(",");                              //write comma to SD card


    //then latitude
    degWhole = double(int(MtoDegree(estNpos)/100.0));
    degDec=(MtoDegree(estNpos)-degWhole*100.0)/60.0;
    deg=degWhole + degDec;
    if (GPS.lat=='S') {                                   //If you are in Southern
    hemisphere latitude should be negative
      deg= (-1)*deg;
    }
    mySensorData.print(deg,10);                           //writing decimal degree longitude
    value to SD card
    mySensorData.print(",");                              //write comma to SD card


   //then altitude
    mySensorData.print(GPS.altitude);                     //write altitude to file
    mySensorData.print(" ");                              //format with one white space to
    delimit data sets

    mySensorData.close();
  }
}

//Initialize KML wrapper
void initKML(){
  mySensorData = SD.open(fileName, FILE_WRITE);
  mySensorData.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<kml
  xmlns=\"http://www.opengis.net/kml/2.2\">\n<Document>\n<Style
  id=\"yellowPoly\">\n<LineStyle>\n<color>7f00ffff</color>\n<width>4</width>\n</LineStyle>\n<P
  olyStyle>\n<color>7f00ff00</color>\n</PolyStyle>\n</Style>\n<Placemark><styleUrl>#yellowPoly
  </styleUrl>\n<LineString>\n<extrude>1</extrude>\n<tesselate>1</tesselate>\n<altitudeMode>cla
  mpToGround</altitudeMode>\n<coordinates>");
  mySensorData.close();
}
//Close KML wrapper
void closeKML(){
  mySensorData = SD.open(fileName, FILE_WRITE);
  mySensorData.println("</coordinates>\n</LineString></Placemark>\n</Document></kml>");
  mySensorData.close();
}


// PART 6 - UI AND DISPLAY DESIGN
*************************************************************************
void introDisplay(){
  display.begin();
  display.fillScreen(BLACK);
```

```
  display.display();

  display.setCursor(0, 0);
  display.setTextColor(WHITE);
  display.setTextSize(2);
  display.println("TeamSHARK");
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.println("Racing");
  display.println();display.println("UOM");
  display.println("(C)SKcreations");
  display.display();

}

void GPSdisplay(){
  timeCorrection();
  if ((int)GPSfix >=1){
    display.drawFastVLine(1,2,2, WHITE);
    display.drawFastVLine(2,2,2, WHITE);
    display.drawFastVLine(3,1,3, WHITE);
    display.drawFastVLine(4,1,3, WHITE);}
  if ((int)GPSfix >= 2){
    display.drawFastVLine(5,0,4, WHITE);
    display.drawFastVLine(6,0,4, WHITE);}

  if (sys>0)display.fillCircle(12, 2,2, WHITE); //Calibration status
  else display.drawCircle(12, 2,2, WHITE);

  if ( logButtonState == HIGH ) display.fillCircle(20, 2,2, WHITE); //Logging status
  else display.drawCircle(20, 2,2, WHITE);

  display.setCursor(80, 0);

  display.print("");
  display.print(lHour, DEC); display.print(':');
  display.print(lMinute, DEC); display.print(':');
  display.println(lSec, DEC);
  display.setCursor(0, 6);

  if (GPSfix) {
    display.print(MtoDegree(estNpos), 4);display.println(GPSlat);
    display.print(MtoDegree(estEpos), 4);display.println(GPSlon);

    display.print("Vn : "); display.println(estNvel);
    display.print("Ve : "); display.println(estEvel);
  }
}

//Display accelerometer data
void AccDisplay(){
  display.drawRect(70, 5, 58, 58, WHITE);
  display.drawFastVLine(99, 5, 58, WHITE );
  display.drawFastHLine(70, 34, 58, WHITE );

  display.setCursor(0, 40);
  display.setTextColor(BLACK, WHITE);
  display.println("      G-Map");
  display.setTextColor(WHITE);
  display.print("X: ");
  display.println(abs(accN));
  display.print("Y: ");
  display.print(abs(accE));

  gDot[0]= map(gACC[0], -0.9,0.9 ,70,128);
  gDot[1]= map(gACC[1], -0.9,0.9 ,63,5);

  display.fillCircle(gDot[0], gDot[1], 3, WHITE);

  //North comapss
  display.drawLine(99, 34, Nx, Ny, WHITE);
```

```
}

//Display orientation data
void OriDisplay(){
  display.setCursor(0, 0);
  display.setTextColor(WHITE);
  display.setTextSize(1.5);
  display.setTextColor(BLACK, WHITE);
  display.println(F("Orientation:"));
  display.setTextColor(WHITE);

  display.setTextSize(1);
  display.print(F("X - "));
  display.println(roll);
  display.print(F("Y - "));
  display.println(pitch);
  display.print(F("Z - "));
  display.println(yaw);
  display.println();

  display.setCursor(120, 0);
  display.print(sys, DEC);
}


// Display sensor details
void displaySensorDetails(void)
{
  sensor_t sensor;
  bno.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" xxx");
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" xxx");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" xxx");
  Serial.println("------------------------------------");
  Serial.println("");

  display.fillScreen(BLACK);
  display.display();

  display.setCursor(0, 0);
  display.println("----------------------");
  display.print  ("Sensor:       "); display.println(sensor.name);
  display.print  ("Driver Ver:   "); display.println(sensor.version);
  display.print  ("Unique ID:    "); display.println(sensor.sensor_id);
  display.print  ("Max Value:    "); display.print(sensor.max_value); display.println(" xx");
  display.print  ("Min Value:    "); display.print(sensor.min_value); display.println(" xx");
  display.print  ("Resolution:   "); display.print(sensor.resolution); display.println(" xx");
  display.println("----------------------");
  display.println("");
  display.display();

  delay(2000);
}


// PART 7 - MCU SETUP
*********************************************************************************
void setup() {

  GPSthreadID = threads.addThread(GPSthread);                 // init GPS thread at 10Hz
  sample rate
  //IMUthreadID = threads.addThread(IMUthread);              // IMU thread is not used for
  now as 100 hz
  threads.setSliceMicros(10);                                 //sample rate is acceptable for
  DAQ
                                                              //If needed IMU can go upto
                                                              1000Hz Using thread
```

```
  Serial.begin(115200);                                          //Turn on serial monitor

  introDisplay();                                                //Display intro
  setupGPS();
  setupSD();
  setupIMU();                                                    // Setup hardware
  setupButton();

  getOri();                                                      // Get first IMU output
  readGPS();                                                     // Get first gps output
}

void setupButton(){
  // Setup the button with an internal pull-up :
  pinMode(BUTTON_PIN,INPUT_PULLUP);

  // After setting up the button, setup the Bounce instance :
  logButton.attach(BUTTON_PIN);
  logButton.interval(5); // interval in ms
}

void setupKalman(){

  Kalman kN(GPSlatitude,posCov,accCov,double(millis())/1000.0);
  Kalman kE(GPSlongitude,posCov,accCov,double(millis())/1000.0);

}

void setupGPS(){
  GPS.begin(9600);                                              //Turn on GPS at 9600 baud
  mySerial.begin(9600);
  GPS.sendCommand("$PGCMD,33,0*6D");                             //Turn off antenna update
  nuisance data
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);                  //Request RMC and GGA Sentences
  only
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_10HZ);                    //Set update rate to 10 hz
  GPS.sendCommand(PMTK_API_SET_FIX_CTL_5HZ);
  delay(1000);
}


void setupSD(){
  pinMode(10, OUTPUT);                                           //Must declare 10 an output and
  reserve it to keep SD card happy
  SD.begin(chipSelect);                                          //Initialize the SD card reader

  if (SD.exists("NMEA.txt")) {                                   //Delete old data files to start
  fresh
    SD.remove("NMEA.txt");
  }
  if (SD.exists(fileName)) {                                      //Delete old data files to
  start fresh
    SD.remove(fileName);
  }
}


void setupIMU(){
  //Initialise the sensor
  if(!bno.begin())
  {
    //There was a problem detecting the BNO055 ... check your connections
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
    while(1);
  }
  bno.setExtCrystalUse(true);                                    //Use external crystal for
  better accuracy
  displaySensorDetails();                                        //Display some basic information
  on this sensor
}
```

```
// PART 8 - MCU MAIN LOOP and multi-threads
******************************************************************************
void loop() {
    // IMU thread is not used for now as 100 hz adequate for main loop
    getOri();
    gMap();                                         //Get IMU output
    TimeNowSec = millis()/1000.0;
    //................................................Kalman filter
    if (GPSfix)
    {
      kE.predict(accE,TimeNowSec);
      kN.predict(accN,TimeNowSec);                  // Prediction step Using IMU output at
      100 Hz

      // if millis() or timer wraps around, we'll just reset it
      if (timer2 > millis())  timer2 = millis();

      if (millis() - timer2 > 100)
      {
        timer2 = millis();
        kE.updates(GPSlongitude,GPSspeedE);
        kN.updates(GPSlatitude,GPSspeedN);          // Update step Using GPS output at 10 Hz
      }
      estEpos = kE.getPosition();
      estEvel = kE.getVelocity();                   // Get kalman output for East axis

      estNpos = kN.getPosition();
      estNvel = kN.getVelocity();                   // Get kalman output for East axis


    }
    LBSprev = logButtonState;
    logButton.update();
    logButtonState  = logButton.read();

    if (logButtonState > LBSprev) {
        getFileName();
        Serial.println(fileName);
        initKML();                                  //Start Log to SD card if Button pressed
    } else if (logButtonState < LBSprev) {
        Serial.println("down");
        closeKML();                                 //Stop logging is pressed again
    }
    if ( logButtonState == HIGH ) {                 //Log to SD card if Button state is true
      logData();
    }

    display.clearDisplay();
    GPSdisplay();
    AccDisplay();                                    // Display output
    display.display();

    delay(10);
}

void IMUthread(){
  while(1){
    // IMU thread is not used for now as 100 hz adequate for main loop
    threads.delay(10);
  }
}


void GPSthread(){

  while(1){
    //GPS thread at 10 Hz
    readGPS();
  }
}
```