Report
IMU data processing

Federico Tiberti

I. Description of exercise

In today's lesson we described the different sensor ,in particular: encoders, accelerometers and gyroscopes. In the practical exercise we therefore used an IMU, a sensor that can have up to 10dof if it integrates an accelerometer, a gyroscope, a magnetometer and a pressure sensor. My IMU is the MPU6050 GY-521 so it has 6dof.
We followed three steps:

1) Reading the raw data
2) Normalization
3) Filtering (via *complementary filter*)

II. Setup

- Circuit diagram used:

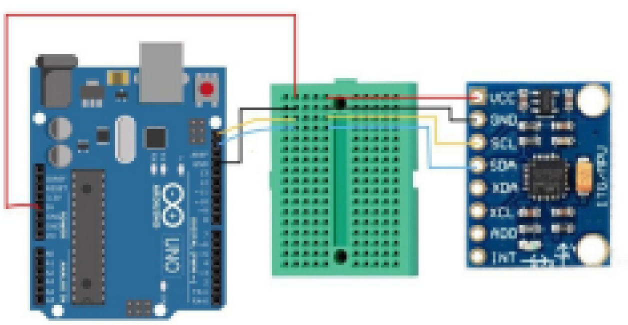

Fig. 1: Circuit diagram

- As regards step 1, I used the getMotion6(&ax,&ay,&az,&gx,&gy,&gz) method applied to the imu object instantiated by the MPU6050 class.
  This method assigns readings from the accelerometer and gyroscope to the respective variables passed as parameters, respectively. In this case the results obtained are expressed in the decimal conversion of the binary number read (the measurements have a 16bit scale).

- In step 2 I normalized the data according to the respective resolutions of the gyroscope and accelerometer which are equal to 250°/s and 2g of default respectively.

- In the last step I implemented the complementary filter by updating the angles at each loop and "integrating" the measurement of the gyroscope which is a speed by multiplying it by *dt* (time of each iteration).

III. Code

Part 1:

```
MPU6050_filtered_data
#include "math.h"
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

#define degToRad(angleInDegrees) ((angleInDegrees) * M_PI / 180.0)
#define radToDeg(angleInRadians) ((angleInRadians) * 180.0 / M_PI)
//#define readAngles() (imu.getMotion6(&ax,&ay,&az,&gx,&gy,&gz))

MPU6050 imu;

int16_t ax,ay,az,gx,gy,gz;
float accX,accY,accZ,gyroX,gyroY,gyroZ;
int ax_offset, ay_offset, az_offset, gx_offset, gy_offset, gz_offset;
float phiXa, phiYa, phiX=0, phiY=0, phiZ=0;

//time parameters
int16_t t0,t1;
float dt;

//normalization parameters
float accRes = 2.0; //default 2g per l'acc.
float divider = 32767.0;
float gyroRes = 250; //default 250°/s per il gyro

//complementary filter parameters
float alphaCF = 0;
float tau = 0.075;

void setup(){
  Wire.begin();
  Serial.begin(115200);
  imu.initialize();
  // da MPU6050_calibration.ino ottengo i seguenti offset:

  ax_offset = 820; ay_offset = -2621; az_offset = 718;
  gx_offset = 17; gy_offset = -14; gz_offset = 28;

  imu.setXAccelOffset(ax_offset);
  imu.setYAccelOffset(ay_offset);
  imu.setZAccelOffset(az_offset);
  imu.setXGyroOffset(gx_offset);
  imu.setYGyroOffset(gy_offset);
  imu.setZGyroOffset(gz_offset);

}
```

Fig. 2: Part 1 code

In this first part you can see: the libraries used, the functions for the conversion from degrees to radians and vice versa, the declarations of the imu object and all the variables and the setup. In the latter we "start" the communication I^2C from the Wire library and then we set the offsets of the instruments read from the "MPU6050calibration.ino" file.

Part 2:

```
void readAngles(){
  imu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  t1 = millis();
  dt = (t1-t0) / 1000.0;
  t0 = t1;

  //NORMALIZE ACC
  accX = ax*accRes/divider;
  accY = ay*accRes/divider;
  accZ = az*accRes/divider;

  //NORMALIZE GYRO (velocità angolari normalizzate)
  gyroX = degToRad(gx*gyroRes/divider);
  gyroY = degToRad(gy*gyroRes/divider);
  gyroZ = degToRad(gz*gyroRes/divider);

  gyroX = (gyroX + sin(phiX)*tan(phiY)*gyroY + cos(phiX)*tan(phiY)*gyroZ);
  gyroY = (cos(phiX)*gyroY - sin(phiX)*gyroZ);
  gyroZ = ((sin(phiX)/cos(phiY))*gyroY + (cos(phiX)/cos(phiY))*gyroZ);

  //calcolo del roll dall'acc. (phiXa)
  phiXa = atan2(cos(accY)*sin(accX), cos(accY)*cos(accX));

  //calcolo del pitch dall'acc (phiYa)
  phiYa = atan2(sin(accY), sqrt(pow(cos(accX)*sin(accX),2) + pow(cos(accY)*cos(accX),2)));
}

void compFilter(){
  alphaCF = tau/(tau+dt);

  phiX = alphaCF * (phiX + gyroX*dt) + (1-alphaCF)*phiXa;
  phiY = alphaCF * (phiY + gyroY*dt) + (1-alphaCF)*phiYa;
  phiZ = alphaCF * (phiZ + gyroZ*dt);  //no correction because no magnetometer
}

void loop(){
  readAngles();
  compFilter();
  Serial.print("Roll: "); Serial.print(radToDeg(phiX)); Serial.print(", ");
  Serial.print("Pitch: "); Serial.print(radToDeg(phiY)); Serial.print(", ");
  Serial.print("Yaw: "); Serial.print(radToDeg(phiZ)); Serial.print(", ");
  Serial.print("Loop time: "); Serial.print(dt*1000.0); Serial.println("ms.");
}
```

Fig. 3: Part code 2

In this part we note the two implemented functions "readAngles()" and "compFilter()". In the first step1 (in the first line) and step2 of the normalization are implemented. In the second function, however, there is the heart of the exercise (step 3), i.e. data filtering. Note that in the z component there is no accelerometer correction, this is because the imu used does not have a magnetometer. Finally, the functions are executed in the loop and the results are printed

IV. RESULTS

Monitor seriale step 1:

944	12	16192	-2	-3	16
908	56	16332	0	14	20
864	-60	16288	4	15	-16
772	52	16344	-18	-19	11
928	-8	16388	5	-7	15
888	-8	16384	-4	19	0
1004	64	16312	-24	-9	5
904	20	16356	16	2	1
884	28	16308	-13	-1	-2
912	92	16472	-10	-5	15
924	16	16384	-10	13	13
848	-44	16272	-20	7	1
888	-56	16288	4	-22	0
844	8	16360	-3	16	11
976	20	16404	-3	-3	-7
896	4	16268	3	3	-11
888	28	16388	4	-11	-5
960	-48	16436	19	-4	1
916	48	16328	-13	-17	-10

Fig. 4: Raw data

Note that on the z of the accelerometer there is a value approximately equal to `16384 which is the decimal conversion of the binary number that expresses the gravitational acceleration of 1g. In the x and y we observe the noise, while the data from the gyroscope tells us that it is stationary. `
Monitor seriale step 2:

0.06	-0.00	1.00	-0.01	-0.00	0.00
0.05	0.00	1.00	-0.00	0.00	0.00
0.05	0.00	1.00	-0.00	0.00	0.00
0.05	0.01	0.99	-0.00	0.00	0.00
0.05	-0.00	1.00	-0.00	0.01	-0.00
0.05	0.00	1.00	-0.00	0.00	-0.00
0.05	0.00	1.00	-0.01	0.00	-0.00
0.06	-0.00	1.00	-0.00	-0.00	0.00
0.05	0.00	0.99	-0.00	-0.00	-0.00
0.06	0.00	1.00	-0.00	0.00	0.00
0.05	-0.00	1.00	-0.00	-0.00	-0.00
0.05	0.00	1.00	-0.00	-0.00	0.00
0.05	-0.00	1.00	-0.00	-0.00	-0.00
0.05	0.00	1.00	-0.00	0.00	0.00
0.05	-0.00	1.00	-0.01	0.00	0.00
0.05	0.00	1.00	-0.00	-0.00	0.00
0.05	-0.00	1.01	-0.00	0.00	0.00

Fig. 5: Normalized data

Normalizing the data, the acceleration on z is equal to 1, while the other measurements are approximately 0 as the imu is stationary.

.

Serial plotter step 3 (imu stops):

Fig. 6: Filtered data

Serial plotter step 3 rotating the imu around y:

Fig. 7: Filtered data (pitch)

In this last graph you can see the hardware problem I had, as I physically rotated the imu by approximately ±130°, but the data obtained reaches a minimum of ±58° when in reality the imu is at ±90°. For this reason I could not see the effect of the Gimbal Lock.

V. DISCUSSION OF RESULTS

Of particular interest to me was measuring for the first time in reality the various noises that can affect the measurements. In fact From the various steps you can see how the accelerometer has greater noise but it does not accumulate it over time, vice versa the gyroscope is precise in short period but accumulates error (so called slowly *varying function*). To extrapolate the most accurate measurement possible from the two data we used the complementary filter that combines the two tools in the best way through the parameter *alphaCF*. The idea is to "amplify" the pros and reduce the cons of the two measures. Another effect you may notice is the Gimbal Lock, problem inherent in Euler angles. Unfortunately I couldn't see this effect because my imu goes into saturation at about 60° on the axis x and y, so I couldn't bring the pitch angle to $\pm\pi/2$. In any case, I saw this effect in class and was able to notice it as soon as this condition is reached, the other two signals also appear they start to get much more distorted. This happens for a singularity in the rotation matrices that carries the angle of roll and yaw to no longer be independent of each other, which is why you lose a degree of freedom.