MPU6050_light library documentation

Romain JL Fetick

November 2020

Abstract

The MPU6050_light library is made for your Arduino to communicate with the MPU6050 device. It retrieves the MPU6050 acceleration, gyroscopic and temperature measurements. The angles of the device are computed from the raw measurements. The tilt angles are computed by a complementary filter between acceleration and gyroscopic data, thus providing quite a good accuracy of estimation.

Contents

1 Introduction

The MPU6050 delivers three angular speeds and three linear accelerations. This raw data is directly accessible through the MPU6050_light library. Moreover the library performs basic angle computation from this data. Indeed an integration of the angular speed gives the angles. However any small bias in the angular speed measurement is integrated and leads to a severe drift of the estimated angle. This must be corrected using another information, that is provided by the accelerometer. The gravity is supposed to be the main component of the acceleration, so a projection of the gravity on the MPU6050 axis gives a rough (and noisy) estimate of the tilt angles of the device.

Merging the angles estimated from gyroscopic and accelerometric data provides a good estimation of the MPU6050 angles. A complementary filter performs the merging of the two sources. For this library, the gyroscopic data is weighted with a 0.98 factor, whereas the accelerometric data is weighted with the complementary 0.02 factor. This factor can be fine tuned by the user. The main hypothesises to ensure the consistency of the angle estimation are

- Small loop delay between to calls to the `update( )` function, so the approximation made for the numerical integration of the angular speed holds $\theta_g[i] \simeq \theta_g[i-1] + \dot{\theta}_g[i] \cdot dt$
- Small linear accelerations (the gravity is the main one). This constraint can be loosened since the complementary filter gives few confidence in angles computed from acceleration (only 2% of confidence as default value)
- Small tilt angles (due to approximations in the trigonometric equations)

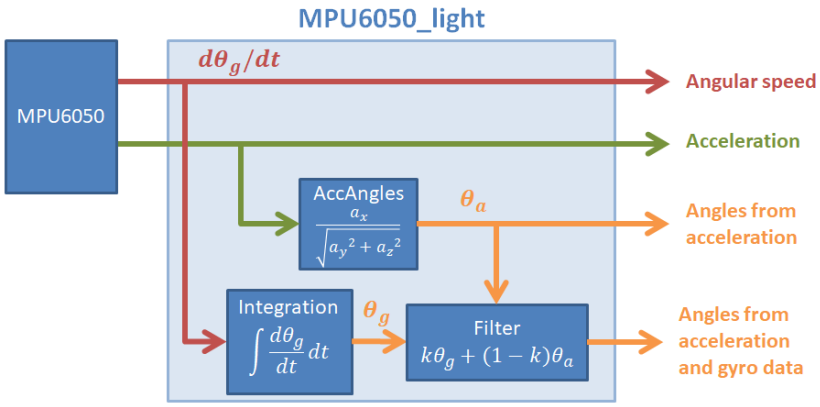

Figure 1: Overview of the data accessible through the MPU6050_light library. The raw data from the MPU6050 is the linear acceleration (green) and angular speed (red) for the three axis. Computations allow to derive the angles of the MPU6050. Angles computed from acceleration are noisy. The angles computed from data fusion between accelerometric and gyroscopic data are more accurate.

2 Methods of the MPU6050 class

2.1 Constructor and initialisation

MPU6050(TwoWire &w):
Constructor. Needs to be called with *Wire*.

begin(int gyroConfig=1, int accConfig=0): void
Start communication with the MPU6050 device. User may choose a gyroscope configuration (from 0 to 3) and accelerometer configuration (from 0 to 3) that set the sensitivity and working range of the device. The configuration list can be found on the Invensense website.
The function returns a byte to check the success of the connection (0=success).

calcOffsets(bool calcGyro=true, bool calcAcc=true): void
Compute gyroscope and accelerometer offsets to remove measurement bias. MPU6050 device must be on a flat surface during the calibration.

2.2 Configuration setters

setGyroOffsets(float x, float y, float z): void
Manually set gyroscope offsets if you already know them. Otherwise use the method `calcOffsets`( ).

setAccOffsets(float x, float y, float z): void
Manually set accelerometer offsets if you already know them. Otherwise use the method `calcOffsets`( ).

setFilterGyroCoef(float fg): void
Set the gyroscope coefficient for the complementary filter. The coefficient must be between 0 (accelero data only) and 1 (gyro data only). Recommended value $f_g = 0.98$. This function automatically sets the accelerometer coefficient to $f_a = 1 - f_g$.

setFilterAccCoef(float fa): void
Set the accelerometer coefficient for the complementary filter. The coefficient must be between 0 (gyro data only) and 1 (accelero data only). Recommended value $f_a = 0.02$. This function automatically sets the gyroscope coefficient to $f_g = 1 - f_a$.

2.3 Update measurement

update( ): void
Update data. This function must be called in the loop as often as possible to get consistent `angleX`, `angleY` and `angleZ`. Indeed these angles rely on the integration of the gyroscope data between two updates, a longer delay leads to a less accurate integration.

2.4 Data getters

getTemp( ): float
Last measurement of the device temperature.
Units: data is given in Celsius degrees.

getAccX[,**Y**,**Z**]( ): float
Last measurement of the acceleration on the X axis.
Units: data is given as multiple of the gravity norm $g = 9.81$ m.s^{-2}
Note: similar functions exist on Y and Z axis.

getGyroX[,**Y**,**Z**]( ): float
Last measurement of the angular speed around the X axis.
Units: data is given in degrees per second.
Note: similar functions exist on Y and Z axis.

getAccAngleX[,**Y**]( ): float
Estimated angle around the X axis, computed with the accelerometer data (tilt of the gravity vector). This data is noisy, and is valid only for small linear accelerations (the gravity is the major acceleration). It is better to use the angle given by the `getAngleX( )` function.
Units: data is given in degrees.
Note: a similar function exists on Y axis.

getAngleX[,**Y**,**Z**]( ): float
Estimated angle around the X axis. The angle is computed through a complementary filter merging data from accelerometer and gyroscope integration. This might be the best estimate of the angles provided by this library.
Units: data is given in degrees.
Note: similar functions exist on Y and Z axis.

3 Examples

The minimal code below shows how to initialise and retrieve some data from the MPU6050 device.

```
#include "Wire.h"
#include <MPU6050_light.h>
MPU6050 mpu(Wire);

void setup() {
  Wire.begin();
  mpu.begin();
  mpu.calcOffsets();
}

void loop() {
  mpu.update();
  float angle[3] = {mpu.getAngleX(),
                    mpu.getAngleY(),
                    mpu.getAngleZ()};
  float gyro[3] = {mpu.getGyroX(),
                   mpu.getGyroY(),
                   mpu.getGyroZ()};
  // process this data for your needs...
  // (do something...)
}
```

The setup can be fine tuned with the following commands

```
void setup() {
  Wire.begin();
  // define gyro and accelero config (from 0 to 3)
  byte status = mpu.begin(1,0);
  // if initialisation error -> stop
  while(status!=0){ }
  // define which offset (gyro,accelero) to compute
  mpu.calcOffsets(true,true);
  // define custom complementary filter (from 0.0 to 1.0)
  mpu.setFilterGyroCoef(0.98);
}
```

More examples can be found in the dedicated "examples" folder of the library. You may have a look at them in order to get started with the library.

4 Authors and license

- RFETICK (github.com/rfetick): modifications and documentation
- TOCKN (github.com/tockn): initial author (forked from v1.5.2)