Journal of Young Scientist, Volume IV, 2016
ISSN 2344 - 1283; ISSN CD-ROM 2344 - 1291; ISSN Online 2344 - 1305; ISSN-L 2344 – 1283

ARDUINO BASED CALIBRATION OF AN INERTIAL SENSOR IN VIEW OF A GNSS/IMU INTEGRATION

Andra Mihaela OANCEA[1], Teodor-Alexandru SARARU[1]

Scientific Coordinator: PhD. Vlad Gabriel OLTEANU[2]

[1]University of Agronomic Sciences and Veterinary Medicine of Bucharest, 59 Mărăşti Blvd, District 1, 011464, Bucharest, Romania, Phone: +4072.725.96.60,
Email: andra.oancea@outlook.com; alex.sararu@outlook.com
[2]Romanian Space Agency, 21-25 Mendeleev Str., District 1, 010032, Bucharest, Romania,
Email: vlad.olteanu@rosa.ro

Corresponding author email: andra.oancea@outlook.com

Abstract

The aim of this paper is to present the testing and calibration of an Inertial Measurement Unit (IMU) by using an Arduino Uno microcontroller. To accomplish this, the Arduino microcontroller will be programmed through Matlab, considering the number of built-in math and engineering functions and the advantages of the numerous plotting methods. Furthermore, filtering algorithms will deal with the calibration of the sensor and analysis of its behaviour in order to reduce the errors caused by the bias and drift rate of the sensor. The last part of the article will focus on future improvements for the application, in terms of model used, general architecture and tuning techniques as well as the coupling of a GNSS sensor.

***Key words**: Arduino, IMU, Matlab, GNSS.*

INTRODUCTION

Arduino was created at Ivrea Interaction Design Institute with the purpose of being a tool for prototyping, but when it reached the mass marked it evolved into a complex tool with various parts adapted for certain needs and a wide library complementary to most of the projects.
Arduino is "an open source prototyping platform" according to the providers, which has both hardware and software parts.

There are various Arduino boards as it is the main component of a project, depending on facts such as power consumption, number of pins for input/output, working voltages as well as physical size, storage capacity, processing resources, processor frequency rate, evolution of the hardware part, price and more considerations. The board we used is the most basic one, namely an Arduino UNO v3.

The first step in any Arduino project is to make the initialisation of the board and write a set of commands in Arduino programming language using the Arduino Software (IDE) that will be sent to the microcontroller of the board, so it will know what to do. However, the uploading should not be performed until an errorless compilation is achieved.
The combination of the hardware and software can perform actions such as reading inputs (e.g. from gas, alcohol, dust, fingerprint, light, vibration, InfraRed (IR), magnetic, sonar, sound, weather sensors) and writing outputs (e.g. with LCD screens, Light Emitting Diodes (LEDs), speakers, motors or just Tweet a message and much more).
As examples of Arduino projects, it could be an electronic piano, a rocket stabilizer, a drone, electronic nose, sonic eye, mp3 player, a phone, RC car, thermostat, intruder alarm, 2D plotter, even a 3D printer and much more.
Besides the Arduino Software (IDE) and Arduino hardware components, we also used MAtrix LABoratory software (MATLAB), which is a strong enginnering tool, which uses a programming language developed by

Mathworks and it is well suited for such applications.
MATLAB is dealing with tools for mathematical calculations, statistics, optimization, communications, control systems, parallel computing, and application deployment and much more.

Because of the Arduino evolution and spreading, Mathworks is now supporting it, through integrating tools for connection between the two.

MATERIALS AND METHODS

As stated above, in order to build this project, we used an Arduino Uno v3 board as "the brain" of the project, an external IMU sensor – MPU6050 – from Sparkfun, a breadboard and connection wires.
The board used is the basic one as we considered it being enough for our project, when speaking about resources. It has an ATmega328P microcontroller, with 5V operating voltage, 16 MHz quartz crystal, 14 digital pins (6 of them can be Pulse Wide Modulation pins PWM), 6 analogue pins, USB connection and a power jack.
An Inertial Measuring Unit is an electronic device capable of measuring angular rate and specific force of a body using accelerometers and gyroscopes. As destination usage, some examples might be the manoeuvre of aircrafts (also Unmanned Aerial Vehicles UAVs), spacecraft, satellites, landers and much more.
The gyroscope is an angular velocity sensor. It is measuring the rate of change of an axis at the real moment, in time.
The IMU we used for this project is an MPU6050 from Sparkfun, which integrates a triple axis accelerometer and a triple axis gyroscope (angular rate sensor). It works with 2.3-3.4 input voltage and it has a Digital Motion Processing (DMP) capable of complex readings as well as gesture detecting and time synchronisation, a digital temperature sensor and an I2C connection used to reduce the noise. For this project we used only raw data reading from the gyroscope.

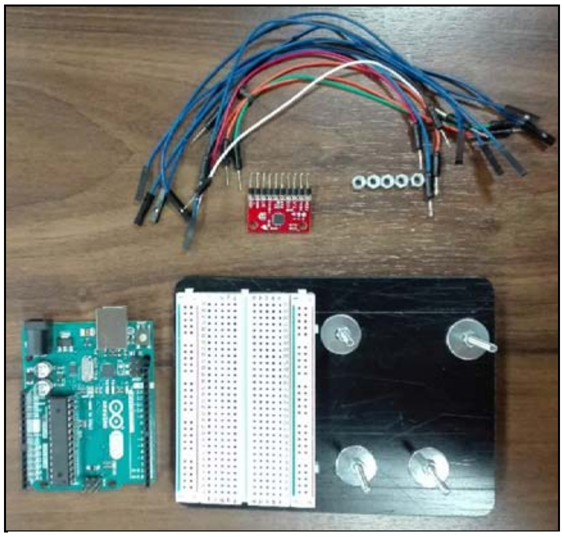


Figure 1. Materials used in the project

As for the methodology used, we divided this into three steps.
The first step refers to the general architecture used in connecting the board with the sensor and breadboard, using connection wires, according to the provider's schemes and breadboard capabilities. Also, we mounted the board, breadboard and sensor on a platform so the assembly resulted will be fixed, this being relevant for the sensor readings and plot.
The second part consisted in working under Arduino software to initialize the board and sensor, access the libraries linked to it and also consisted in writing command lines in order to have an export of the data collected through the sensors within the Arduino software, using Serial Monitor tool.
Once the script has been finished, a compilation is required to be performed, so there will be no errors, with respect to the programming language used.
If the compilation test passes, the script uploaded to the board will be performed as long as it is connected to a power source, regarding that the Arduino (IDE) software divides the script into two parts, one being performed when the board is plugged in or at any reset of it, while the second one is performed in loop after the first part is complete.
The third step regards the Matlab algorithm implementation, where we have processed the data collected from the sensors into a two iteration approach adjustment model, in order

to determine the offset and the drift of the sensor. The adjustment approach was represented by a least squares linear regression. The output of this step can be observed in Figure 7 where one can observe the difference between raw data and the corrected one.
The Matlab scripts were then fed into a very simple Graphical User Interface (GUI) which runs with the software itself and has only four buttons. The buttons can be pressed by the user to perform: connection to Matlab and calibration of the device (including here the measurement number desired) and create plots.

RESULTS AND DISCUSSIONS

The results of the first part, namely the connection between the components and the mounting on support can be seen in the figure below (Figure 2).

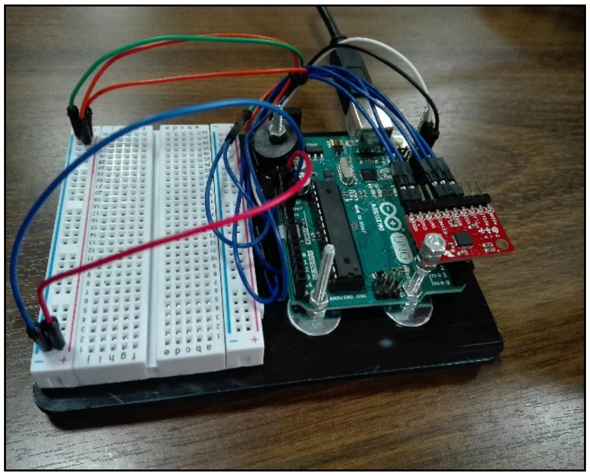
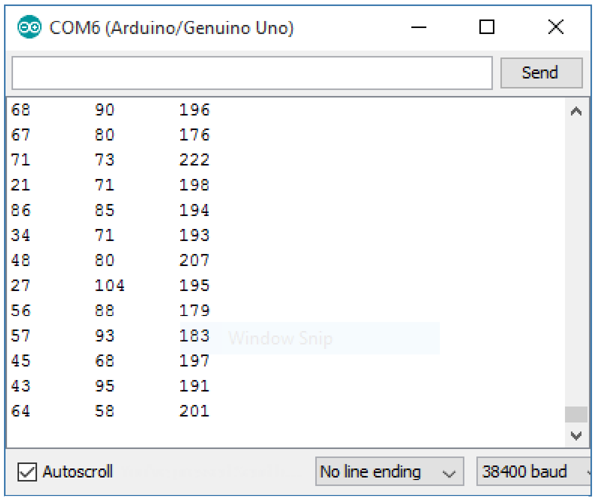


Figure 2. Connection between the components

As said, we first coded in Arduino (IDE) software and the output was achieved using a tool called "Serial Monitor" available in the software.
The Serial Monitor is used to print data collected by the sensor, after we have selected the so called baud rate (down corner on the right side from Figure 3), which represents the communication rate between the computer and the board.

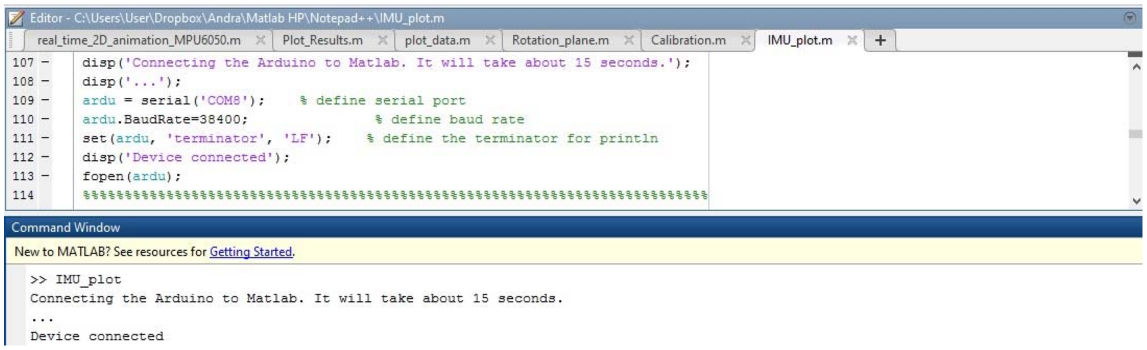
Before uploading the code, the software is making a compilation that verifies if the code was errorless, with salute to the programming language used.
In this case, we have selected to print only the gyroscope data, on columns, with a rate of 10 measurements per second in the following order: the first column is represented by degrees measured along X axis, second column with degrees measured along Y axis and the third column, with degrees measured along Z axis.



```
68      90      196
67      80      176
71      73      222
21      71      198
86      85      194
34      71      193
48      80      207
27      104     195
56      88      179
57      93      183
45      68      197
43      95      191
64      58      201
```

Figure 3. Output from the Serial Monitor tool

However, those are not the real values, because every measurement must be divided by the sensitivity of the sensor, which, with a reading rate set at 250 degrees every second is 131 LSB/dps (Least Significant Bit/degree per second).
After the code is uploaded to the board, the initialization and other commands are stored in the board's memory and it will perform the required tasks until it is unplugged from the power source or we want to upload a new code (or change the existing one).
After the initialization is done, the next step was to connect the Arduino board to the Matlab.
For this, we coded in Matlab software. The following picture is presenting the part of the main code responsible with the connection between Arduino and MATLAB.


Editor - C:\Users\User\Dropbox\Andra\Matlab HP\Notepad++\IMU_plot.m
real_time_2D_animation_MPU6050.m
Plot_Results.m
plot_data.m
Rotation_plane.m
Calibration.m
IMU_plot.m
Command Window
New to MATLAB? See resources for Getting Started.

```
disp('Connecting the Arduino to Matlab. It will take about 15 seconds.');
disp('...');
ardu = serial('COM8');     % define serial port
ardu.BaudRate=38400;                 % define baud rate
set(ardu, 'terminator', 'LF');    % define the terminator for println
disp('Device connected');
fopen(ardu);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
>> IMU_plot
Connecting the Arduino to Matlab. It will take about 15 seconds.
...
Device connected
```

Figure 4. Connection between Arduino board and Matlab

Because the data obtained from the sensor was affected by errors namely, drift and offset, we had to do a calibration. This calibration is based on an adjustment model with two iterations.
The aim of the calibration is to reduce the offset (fixed) and the eventual drift (time dependent) and by combining those two errors we obtain a form similar to linear equations.
As said, we have two iterations. The model used for the first iteration may be seen in the picture below:

$$V_1 = AX_1 - L_1$$

$$\begin{bmatrix} v_1^1 \\ \vdots \\ v_n^1 \end{bmatrix} = \begin{bmatrix} t_1 & 1 \\ \vdots & \vdots \\ t_n & 1 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} - \begin{bmatrix} m_1 \\ \vdots \\ m_n \end{bmatrix}$$

$$X_1 = N^{-1} A^T L_1$$

$$N = A^T A$$

Figure 5. Adjustment method

The V_1 is the vector of residuals, A is observations matrix and L_1 is a free term vector. Also, $t_{1\ldots n}$ is the time of the corresponding measurement $m_{1\ldots n}$, and the parameters a_0 and b_0 correspond to the bias and namely the drift of the sensor on a specific axis.
As there were no initial values for the parameters, we had to ensure that the estimation is convergent, and therefore we did a second iteration taking as initial values for the parameters the ones determined in the first step.
The second iteration is described in the below picture (Figure 6):

$$V_2 = AX_2 - L_2$$

$$\begin{bmatrix} v_1^2 \\ \vdots \\ v_n^2 \end{bmatrix} = \begin{bmatrix} t_1 & 1 \\ \vdots & \vdots \\ t_n & 1 \end{bmatrix} \cdot \begin{bmatrix} \delta_a \\ \delta_b \end{bmatrix} - \begin{bmatrix} m_1 - t_1 a_0 - b_0 \\ \vdots \\ m_n - t_n a_0 - b_0 \end{bmatrix}$$

$$X_2 = N^{-1} A^T L_2$$

Figure 6. Adjustment method (second iteration)

The final parameters were then obtained as the sum of the two:

$$X = X_1 + X_2 = \begin{bmatrix} a_0 + \delta_a \\ b_0 + \delta_b \end{bmatrix}$$

With the offset and drift determined, we have applied the correction to raw data in real time. As a result, one can see the improvements (Figure 7) which it is showing how the measurements were corrected comparing with the raw gyroscope data.
One can note that the drift is very small compared to the measurement error and thus it could be considered as null.

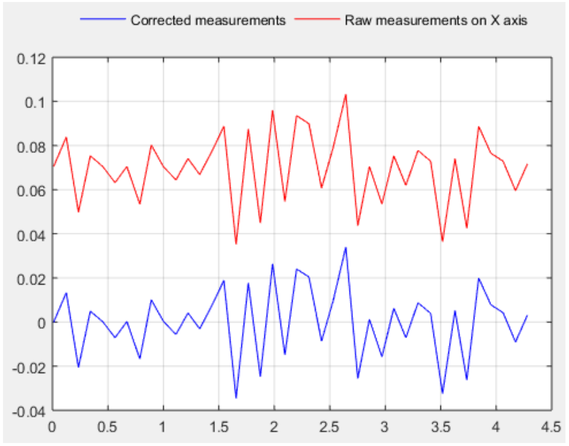


Figure 7. Raw measurements compared with corrected measurements

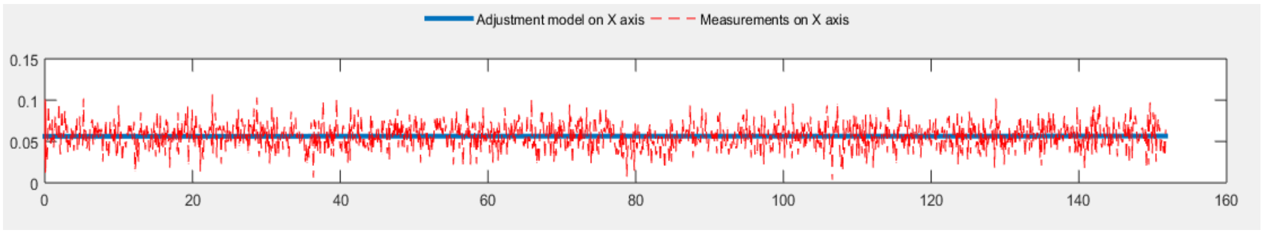


Figure 8. Raw measurements and adjustment model

The third plot is represented by an arrow that is associated with the orientation of the IMU sensor (Figure 9). When the sensor is fixed the arrow is fixed too and when is moving the arrow is moving too. The arrow is moving with the same rate, which the sensor is moving.

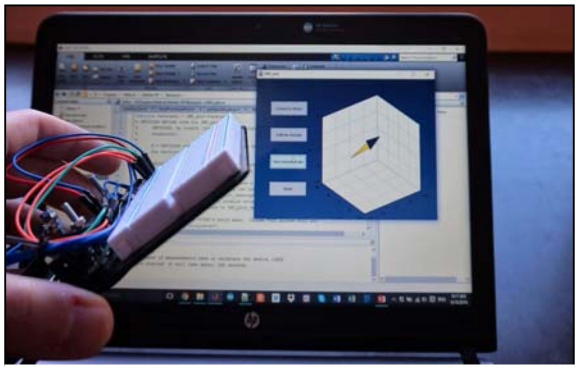


Figure 9. Inertial Measuring Unit orientation

In order to make a good visual representation, we wrapped all in a GUI which stands for Graphical User Interface. Using the buttons in the GUI, the connection between the Matlab and the board and the calibration can be realized together with the plots.

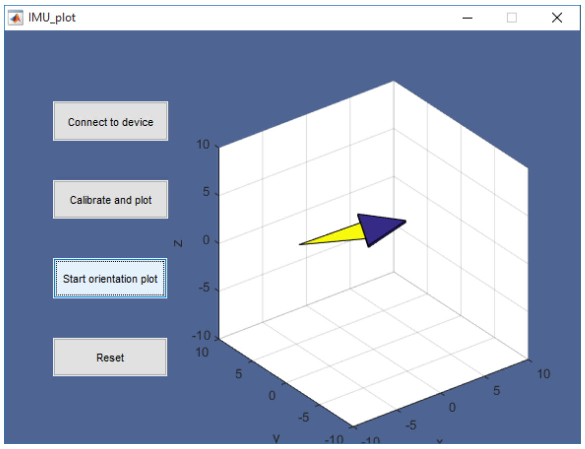


Figure 10. Graphical User Interface with buttons

FUTURE WORKS

Regarding future works, we will firstly want to obtain ever better results by optimizing the scripts and functions for the gyro calibration. Afterwards, we will develop a method for calibrating the accelerometer.
The second step will be represented by the loosely integration of a GNSS receiver and the IMU sensor for navigation purposes. So far we have connected the GNSS sensor and also coded the Arduino IDE part obtaining the individual positions. Those steps will be followed by creating a Kalman filter in order to obtain the position of a moving GNSS receiver considering the error model proposed for the IMU.

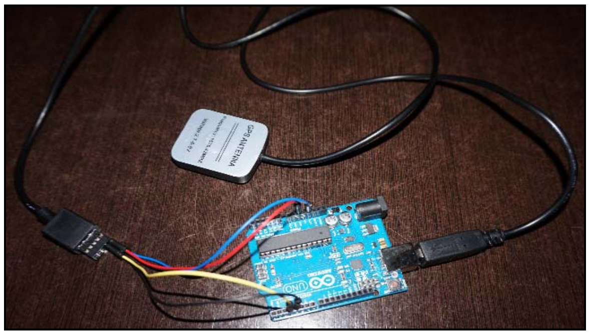
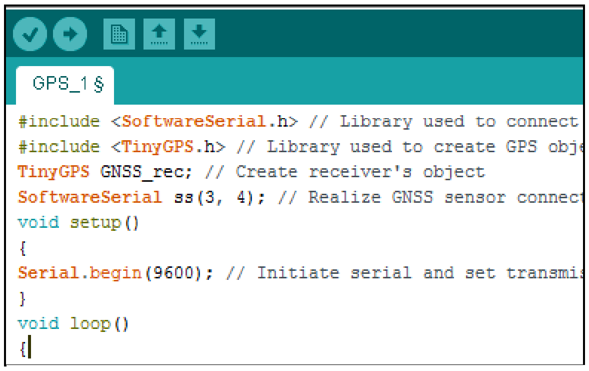


Figure 11. GPS sensor connected to the Arduino board.



```
#include <SoftwareSerial.h> // Library used to connect
#include <TinyGPS.h> // Library used to create GPS obj
TinyGPS GNSS_rec; // Create receiver's object
SoftwareSerial ss(3, 4); // Realize GNSS sensor connect
void setup()
{
Serial.begin(9600); // Initiate serial and set transmi
}
void loop()
{
```
Figure 12. Code section from IDE software that represent the initialization.

CONCLUSIONS

In conclusion, using Arduino hardware and software, together with the Matlab capabilities represents a good tool when we are speaking about small to medium scale projects, especially for student applications which do not require a great initial investment.

REFERENCES

John Baichtal, Arduino for beginners

Arduino, www.arduino.cc

Robofun, www.robofun.ro

Steven J. Miller, The Method of Least Squares

John Fox, Applied Regression Analysis and Generalized Linear Models

Mathworks, http://www.mathworks.com/support/books