physicslabs AT BRISTOL

Experiment I

Inertial Tracking

Aims
By the end of the experiment you should:

- Have used the MEMS accelerometer & gyroscope to perform basic tests of the devices capability.
- Have tested the usability of the device as a "inertial tracker" - following simple 2D and eventually 3D paths.
- *(Optional)* Have used the device in more complex scenarios.

Objectives
This experiment gives you the opportunity to learn more about:

- MEMS devices.
- Computer/device interfaces, and the coding that accompanies it.
- Experimental testing of hardware.

1 Introduction

Inertial tracking is a method of navigation which uses measurements of orientation and acceleration to estimate the path travelled. Inertial navigation systems (INS) are now ubiquitous and used in the navigation of ships, aircraft, submarines and spacecraft and are even found in mobile phones and games controllers. The value of INS is that they do not require an external reference, in contrast to the global positioning system (GPS) for example. During this experiment you should learn about the devices which make this technology possible as well as the data processing required to make location predictions. Two measurements are required for inertial tracking; orientation and acceleration. In this experiment, these two parameters will be measured with a set of microelectro-mechanical systems (MEMS). MEMS are a well established technology which allow cheap production and wide-spread use of all kinds of devices including microphones, transducers and a variety of other sensors. You should read about the types of MEMS designs which can measure linear acceleration and angular velocity.

2 Setup

You are provided with a InvenSense MPU-6050 6-axis accelerometer gyro chip, packaged as a GY-521 module. This module is then connected to an Arduino Nano microcontroller, which then interfaces to the computer via USB to process commands and record data. The wiring between the GY-521 and the Arduino is shown in Figure 1, and is assembled on a small circuit board within a casing.

The Arduino is programmed in a "C-like" language, using its own integrated development environment (Arduino IDE). Open this program and take a look through the various menus available. The code to utilise the GY-521 is provided - look through "MPU6050_raw.ino" [1] and check you understand the basics of what it is trying to do. The code needs to be uploaded (CTRL+U) to the Arduino Nano, ensuring that the correct board and COM ports are selected (under the tools menu). Once successfully uploaded, you can check the Arduino is doing as intended by monitoring the serial output (CTRL+SHIFT+M). You should observe a

[1]credit: J. Rowberg github.com/jrowberg

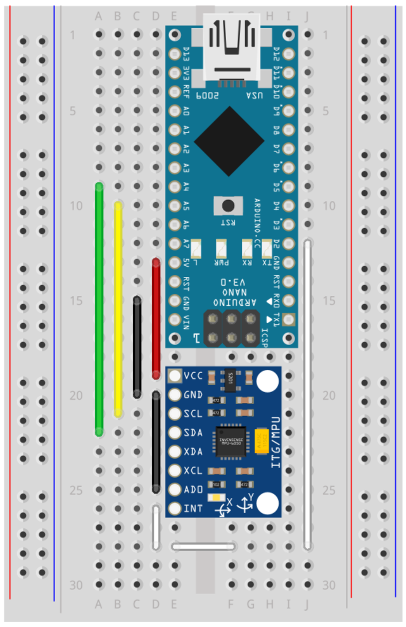

Figure 1: Wiring diagram for the connections between the Arduino and the GY-521.

stream of six channels of data (three for the accelerometer, three for the gyroscope) and a final column showing the elapsed time in ms. Move the device around to check if it responds to motion.

While interfacing with the GY521 to obtain the raw values is relatively straightforward, the values still need to be converted into more meaningful data. This can be done using a variety of programming languages, and for this exercise Python shall be used[2]. A basic script to "interpret" the GY-521 data is provided; locate "GY521_2025.py" in the same folder as you found the IDE script. This basic script will read the serial output from the GY-521 and convert it to linear acceleration and angular velocity, and subsequently plot & record the output. Make a copy of this script in your own OneDrive to allow you to edit it (files on the desktop are restricted). Run the script and check you get a sensible output for both $a_{x,y,z}$ and $\omega_{x,y,z}$. For example, can you reliably measure the acceleration due to gravity, g, in various device orientations? Do all three axes agree with each other?

[2]You are of course free to use whatever language you want.

3 Simple Navigation

The next step is to modify the basic data acquisition script to give angular and cartesian position from the recorded acceleration and angular velocity. This requires integration of the obtained data. To computationally perform this requires multiplying each acceleration point by the time separation to give the velocity:

$$v_{t+1} = v_t + (a_t \times dt) \tag{1}$$

and then again for the displacement:

$$x_{t+1} = x_t + (v_t \times dt) \tag{2}$$

where x, v and a are the displacement, velocity and acceleration, respectively in 1D. This kind of discrete integration can be easily implemented in a Python script (look up the "Euler Method" in a book on numerical computation if needed).

Now, when moving the device around in 2D, you should be able to recreate the path, and as such have made a simple inertial tracker.

However, you will quickly soon come to realise that it can be both quite inaccurate and imprecise. The effects of gravity (at least initially this can be ignored in 2D as it acts within the z direction, assuming the table is level) and also rotation of the device need to be taken into account. First of all though, we can improve the integration performed earlier by the use of some of Python's mathematical packages. Add these lines to the top of your current code:

```
import scipy.integrate
import numpy as np
```

this will allow you to use both the Scipy and Numpy packages. The stepwise integration performed earlier can now be replaced by:

```
vx = scipy.integrate.
cumulative_trapezoid(ax,None,0.01)
x = scipy.integrate.
cumulative_trapezoid(vx,None,0.01)
```

using scipy's cumulative trapezoid function (shown for the x-axis). More information about the various Python packages and functions associated with them can be found in the relevant online documentation.

In addition to gravity, there are drifts & other offsets that need to be compensated for - a non-zero error in acceleration can wildly affect your positional information when integrated. How can you account for this? Filtering of the data may be necessary. Take a look at Scipy's documentation with regards to signal processing[3]. Can you filter the signal and still obtain accurate data? What is the frequency of data collection? Which frequencies can you filter out? What are the device limitations (maximum and minimum measured acceleration)? Can you move the device in some well-defined and simple path and quantitatively reproduce the path it was moved in? For example straight lines and turns with an without a rotation about the vertical axis.

It may not be possible to write a "catch-all" script that accomplishes everything you would wish to do with the device. You should write your script, then test it in various scenarios and observe the results, and re-iterate the process. This is a crucial concept of experimental design and testing.

Note: at this point, in may be beneficial to write a new script for the data analysis, and simply use the original code to record the accelerometer data. This way, you can simply re-use any raw data with different processes applied and compare the outputs.

4 Extension: Rotation & 3D Motion

The above section assumes that the device only moves in a 2D environment, with no rotation or tilting of the device. If the device is rotated, the recorded accelerometer data will no longer be in the initial lab frame of reference, and as such needs to be rotated back to the correct frame (Figure 2). This can be achieved using the data obtained from the gyroscope and suitable rotation matrices for x, y & z:

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix},$$

[3]A few examples of the filtering functions are provided on Blackboard

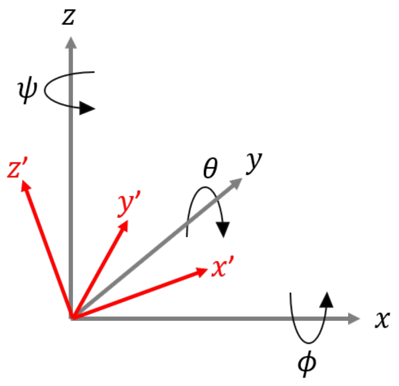

Figure 2: Rotation of the GY-521 out of the initial frame of reference (x, y, z), into a new one (x', y', z'). Rotation angles about the x, y and z axes are ϕ, θ and ψ, respectively.

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix},$$

$$R_z(\psi) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

such that a rotation back to the original frame is given by the application of these matrices to the acceleration vector. In Python these matrix manipulations can be using numpy arrays:

```
Rx = np.array([[1,0,0],
[0,np.cos(tx),-np.sin(tx)],
[0,np.sin(tx),np.cos(tx)]])
```

It may be beneficial to write a separate Python script to confirm these rotations perform as you expect (e.g. a simple rotation of π about the x axis). Once implemented, experiment with your device in 2D (and then 3D) and see if your device accurately tracks the paths you move through. Note that the order of the rotations may matter! Be sure to document improvements/changes to your code and tests you run within your lab-book: as you would for any other experiment.

5 Extension II: Further Experiments

Now you know how to use the accelerometer/gyroscope and how to collect and analyse data. Try another experiment with it. What about measuring fictitious forces in a rotating reference frame? Or using it as a pendulum?