Tutorial 5: Basic BerryIMU Usage

ECE 180D: Systems Design Laboratory

Contents

1 Introduction

This tutorial will allow you to use the BerryIMU to obtain IMU data so that you are able to make estimates as to what gestures are happening, approximate position tracking, etc. This will go over how installation and usage might look like in the beginning and then present a couple of guiding exercises and explanations. Everything will be on ozzmaker.com. These notes will mostly supplement their notes and provide your exercises.

1.1 Materials and Preparation

1. A computer.
2. Raspberry Pi Zero WH.
3. BerryIMU.
4. All necessary connections for Raspberry Pi.

1.2 What is an IMU?

An IMU is an inertial measurement unit. What that means is that it is able to measure a full-range of sensing. In particular, it can be segmented into three parts:

1. Gyroscope (angular velocity measurements)
2. Accelerometer (acceleration measurements)
3. Magnetometer (magnetic sensor)

While a magnetometer is typically not great indoors (especially in Engineering IV - may be better in a normal house / apartment, but this will have to be tested), the gyroscope and accelerometer have the possibility of doing some level of tracking of gestures and motions. Possible project ideas include:

- Head tracking for a VR project (gyroscope)

- Fall detection (accelerometer)
- athletic performance tracker (both accelerometer and gyroscope)
- indoor localization (both accelerometer and gyroscope)

While an IMU has many uses, there are several caveats to using an IMU. There is a huge amount of drift involved when using an IMU, so long-duration fine-tune tracking is a lot less plausible for use. Consider using the IMU for tracking quick gestures, such as a nod, a flick of the wrist, or the tapping of your foot.

1.3 Reviewing Coding

Since we will be using Python primarily for this tutorial, here is a quick tutorial on how to read and use Python. This is just a quick guide for those who are just looking for some syntax tips and tricks:
https://www.w3schools.com/python/default.asp

2 Installation

This installation assumes exactly the configuration of recommendations we have given before. That is, we assume that you have Berryconda installed, and we assume you have the newest BerryIMU (v3) with a QWIIC connector. If you do not, please find a previous tutorial to figure out what to do, or find out another way to interface your BerryIMU with your Raspberry Pi.

1. First, we need to get the setup right. Because you are all at home this year, we did not want to make soldering a requirement. Thus, hopefully you have a QWIIC cable. See Figure 1 to see how to connect the cables. Given the orientation you see in the figure, it should have, for the top 3x2 male headers,

$$\begin{bmatrix} red & none \\ blue & none \\ yellow & black \end{bmatrix}$$

Figure 1: Setup of the Raspberry Pi to use the BerryIMU

2. Set up I2C communications with the Raspberry Pi.

   (a) Install the necessary components:

```
sudo apt-get update
sudo apt-get upgrade
conda upgrade conda
conda update conda
pip install --upgrade pip
sudo apt-get install git i2c-tools libi2c-dev
conda install -c conda-forge smbus2
```

   Some of these commands **will** take a bit of time. Be patient! Many of these you may have already done, but you can just check again one more time anyway. You can also `sudo apt-get install vim / emacs` if you want to use these text editors.

   (b) Try opening the blacklist file

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

   If the file is empty or it does not exist, you can keep going. Otherwise, if there is the line `blacklist i2c-bcm2708`, put a '#' in front to comment this line out.

   (c) Add

```
i2c-dev
i2c-bcm2708
```

   into `/etc/modules` by using `nano` or your preferred text editor.

   (d) Add

```
dtparam=i2c_arm=on
dtparam=i2c1=on
```

   into `/boot/config.txt`.

   (e) Reboot your Raspberry Pi.

```
sudo reboot -h now
```

   (f) After reopening, your ssh connection, you can see if things worked by typing

```
sudo i2cdetect -y 1
```

   You will get something that looks like this Figure 2.

   This shows that there are 3 devices found, which may include the USB port and the BerryIMU. As shown in the guide, 0x6a is the gyroscope, 0x1c is the accelerometer and gyroscope, but you can double check this on the datasheet. If everything is a dash, then something did not work (your IMU isn't being read).

   When this works, this means that the code is able to read your IMU and you can move on to actually interfacing and writing code that uses the IMU.

3. Get the git repository.

```
git clone http://github.com/ozzmaker/BerryIMU.git
```

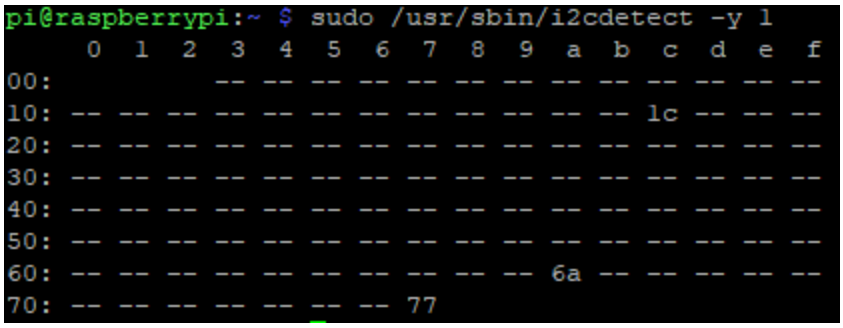

Figure 2: Output of I2C detection

4. While both C++ and Python work for this case, because of how compilation works, Python is typically easier. In the BerryIMU repository, change your directory into

   `BerryIMU/python-BerryIMU-gryo-accel-compass-filters/` and

   ```
   python berryIMU.py
   ```

   You should see an infinite output of the data presented by the IMU. Take a look at the file to see what is going on underneath.

   What you will need is to use these values (as a beginning thing) is to keep the 3 other python files and then build on top of berryIMU.py. Where you will do the most processing from a basic point of view is to add your code after the `######## END ###########` line.

   Feel free to also check out some of the other folders in the git repository as well. Those that do not start with 'python' are C++ based.

5. For now, play around with the data that you are getting and understand how each of the directions work. Show the TA how everything is working correctly.

3 Speech

There are many places for you to try speech processing. We'll be coming around **next week** to see which audio recognition system works well and which doesn't. Keep in mind that there are definitely some major points to keep in mind when considering these speech processing software.

- Purpose. What is the basic purpose of the library? Is it text-to-speech? Is it keyword-detection?
- Latency. That is, can this software operate on a real-time basis?
- Accuracy. In the limited vocabulary you want to use, what is the accuracy of the words that are picked up? In addition, how does the software operate under the presence of noise?

Since next week is focused on your design deliberations and audio is one of them, we hope that you will have a data-backed reasoning explaining why you choose one over the other.

1. CMU Sphynx, notably their Pocketsphynx, is a toolkit that is designed for mobile applications. Pocketsphynx is pretty tricky to set up.
2. Python library of Speech Processing. Here is a large-scale library of all sorts of speech processing tools. This includes CMU Sphynx, so if you were planning on using it after reading the description, this may be a great way to go. It is included in `pip`, so it is definitely a great way to easily get well-supported speech processing tools.

3. There are a couple of higher-complexity speech processing tools used by the teams last year. Some of the options used by previous years included Google Cloud Speech-to-Text and The Porcupine library.

4. If you plan on using Unity, there is also a speech processing toolkit. It is reportedly much better than CMU Sphynx, but may require knowledge of using C# as a programming language (not that it differs significantly with the standard programming languages). Potentially check out some of these audio tutorials.

4 Task: This section overrides all previous sections for tasks

1. Visually explore the IMU data from the constant stream of input upon running the IMU code. Let your weekly report have a screenshot of the constant stream.

2. Build a simple classifier to differentiate between two trivial actions (forward push, upward lift) - and compare your model with your teammates.

3. Challenge (not required but helps explain the non-triviality of IMUs): Build a classifier for 3 actions: the last 2 and a circular rotation motion.

4. Explore different speech recognition tools and implement any one.