EE 267 Lab 6 Notes

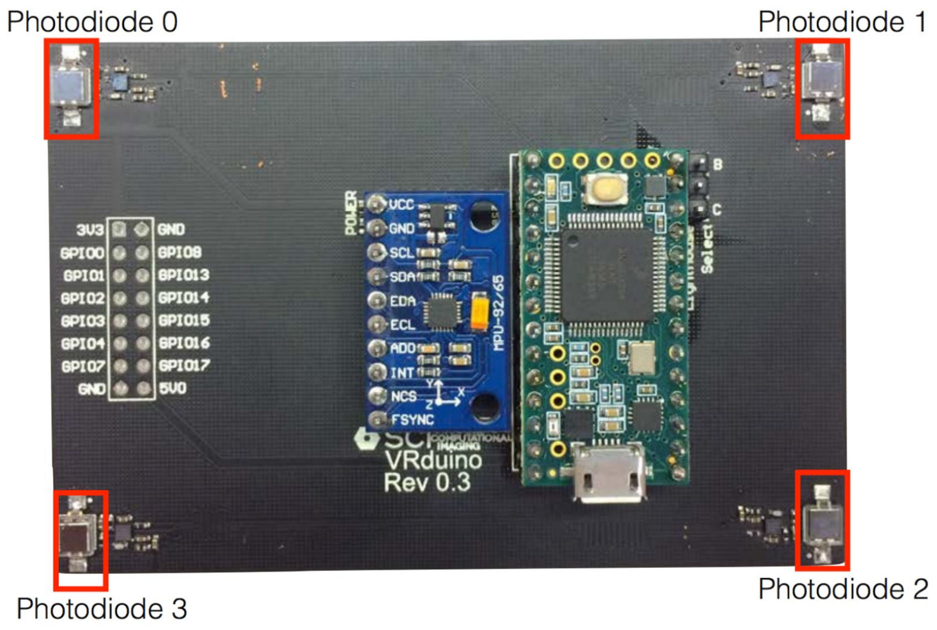
General:

A true virtual reality experience would allow you to fully explore the world beyond head rotation. In this lab, we'll be creating positional tracking which enables one to walk about their virtual scene. We'll explore two separate methods of finding your position in space and then combine our orientation calculation with our Quaternions from last week to create a fully immersive teapot extravaganza!

We'll be releasing all the Arduino starter code today, followed by an additional release of a rendering pipeline in the next few days. Homework will be due on Thursday, May 19th at 11:59pm.

Starter Hardware:

We'll use the same VRduino shield that is provided with the kit. Here's a breakdown of the position tracking features of the board:



The origin of the board is located at the very center. The photodiode locations are as follows:

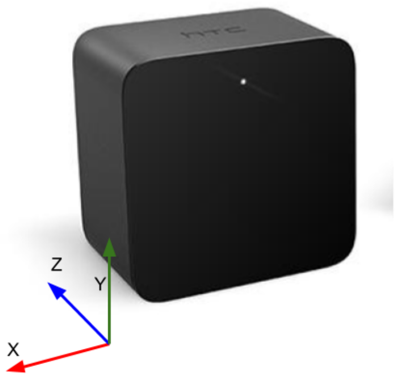
Photodiode 0: (-42.0mm, 25.0mm)
Photodiode 1: (42.0mm, 25.0mm)
Photodiode 2: (42.0mm, -25.0mm)
Photodiode 3: (-42.0mm, -25.0mm)

This geometry has already been provided in the starter code we provide.

The Lighthouse:

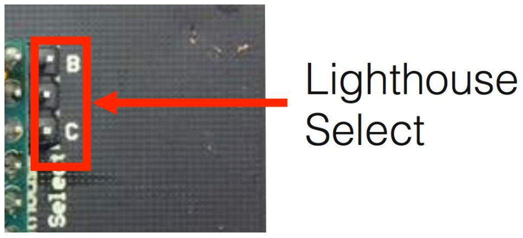
Our position tracking hardware responds to optical signals swept throughout the lab space by the HTC Vive Lighthouses. Please see the Lecture 11 slides for more information. A great video to understand the signals emitted from the Lighthouse can be found here: https://youtu.be/J54dotTt7k0

In this lab, we treat the lighthouse as if it was seated at the origin of the world, looking down the -z axis, in the same way a camera would. The translation vector that results from the position tracking will be with respect to this right-handed coordinate axis. The lighthouse coordinate axis is shown at right:



For our lab, we will use two Lighthouse transmitters in the same space at the same time. They are optically synced and send out slightly different signals, allowing us to distinguish between the Lighthouses on our VRduino device. As long as both devices can see each other's optical sync signals, the VRduino will be able to orient itself in the room, given proper position tracking code, of course.

Important note: In our lab, **Lighthouse B is the master, Lighthouse C is the slave.** These are the two channels our VRduino device has been programmed to listen to. Additionally, both lighthouses must be in the room together - **our VRduino will not work without both B and C lighthouses at this time.**



You can select which Lighthouse channel your device is tuned to with the Lighthouse Select pins, shown at left. With the 2-pin jumper, connect the middle pin to the B pin to listen to the B lighthouse, or to the C pin to listen to the C lighthouse. If no jumper is inserted, the VRduino will be listening to the B channel by default.

We will provide two Lighthouses in the lab at all times. We have two additional Lighthouses, available for use during office hours. Please share this hardware among all teams.

Starter Software:

We will use the same software toolchain that we set up last week in lab. Please refer to last week's writeup if you have any issues with the tools.

Starter Files:

The following files are available on Friday, May 5:

- Server: We use the same Node.js WebSocketsServer as we did last week. Please refer to last week's documentation and Piazza if you have any questions.
- vrduino: We'll be doing our orientation tracking with the IMU using the arduino as the interface between the IMU and the computer. We provide starter code for interfacing with the IMU and reading off quaternion and Euler angles.
    - photoDiodes.cpp/h watch for the timing and angle data we desire for our calculations. **We ask you to extend this file to calculate the rotational angle from the raw timing information**.
    - HomographyPose.cpp/h are used to calculate the position of the headset using traditional Homography methods. **You will modify this class to implement homography.**
    - LevenbergMarquardt.cpp/.h are used to calculate the position of the headset using LM optimization techniques.. **You will modify this class to implement LM position tracking.**
    - vrduino.ino is the code that will run on the Arduino. You will not need to modify this class! We have implemented all function calls for you and ask that you focus on homography and LM position calculations.
    - TestHomographyPose.cpp contains function calls to the methods we asked you to write, as well as known outputs to these functions. Your positional tracking using Homography methods will not work without passing these unit tests first!!
    - TestLMPose.cpp contains function calls to the methods we asked you to write, as well as known outputs to these functions. Like with homography, the Levenberg Marquardt unit tests will be necessary to properly implement LM position-based tracking.
    - Euler.h defines a class to hold rotation based on Euler angles.
    - Quaternion.h contains a complete implementation of the quaternion orientation tracking that was developed last week.
    - MatrixMath.cpp/.h provide useful mathematical Matrix operations. You'll find these useful when working with matrices!
    - Utils.h are helpful for debugging.
    - imu.cpp/.h contain the headers and class definition to interface with the IMU.
    - inputCapture.cpp/.h are libraries used to generate accurate timing signals from the photodiode interrupts.
    - lighthouseSensor.cpp/.h represent the individual photodiode circuits and store raw data values for each axis from both lighthouses.

We will be releasing an additional set of files later in the week which implement rendering on the headset with the values resulting from the positional tracker.

Resources:

- EE267 Lecture 11 Notes
- EE267 Lecture 12 Notes
- Arduino Reference Documentation
- Review of pointers - to skim

Hints:

- Be sure to make your hardware work in lab. While we will most likely be posting firmware updates to improve usability and robustness, we'd like for you to all leave the lab with properly working hardware.
- Be sure to pass all unit tests before moving on to serial streaming. Failed unit tests indicate that the full position calculation will fail. It's much harder to debug at this point, so be rigorous and diligent with your testing!
- Since we're moving around arrays of values within memory, it's important to keep track of pointers and memory allocation. If you need to make a duplicate of an array, we recommend the function `memcpy()`. Documentation is available here for reference. http://www.cplusplus.com/reference/cstring/memcpy/
- **The Arduino must be running at 48MHz.** Be sure that this is selected under Tools->CPU Speed. If this clock speed is incorrect, the conversion between clock ticks and angles will be treacherously off!
- You can switch between streaming modes by pressing the number keys in a serial stream. Here are the different streaming modes:
    - NONE = 0;
    - Quaternion = 1;
    - Homography = 2;
    - Levenberg-Marquardt = 3;
    - Quaternion + LM = 4;
    - Raw clock values = 5;
- Please follow the Homework 6 Updates thread on Piazza. We will update the writeup, starter code, and lectures as bugs are spotted. **We expect you to stay current with this thread and let us know if you have any issues or uncertainties.**

Submitting assignments:

There's a submit.js file that we include. Don't touch it! But when you're ready to submit, open the render.html file in your browser and drag the entire js folder onto the browser window. This will zip up all required files. Upload this zip file to gradescope with the .pdf writeup.