Lighthouse Tracking for 3D Diode Constellation

Theo Diamandis
Stanford University
Stanford, CA, USA
tdiamand@stanford.edu

Georgia Murray
Stanford University
Stanford, CA, USA
gmurray@stanford.edu

Abstract

We extended the Homography and Levenberg Marquardt positional tracking techniques to be compatible with 3D photodiode constellations. We then constructed a 3D constellation using two VRduinos, positioned to maximize range of motion within lighthouse line of sight. We were thus able to increase the tracking systems range of rotation from 105° to 195°.

1. Introduction and Motivation

The VRduino in conjunction with HTC Vive Lighthouse tracking systems offers a straightforward implementation of VR positional tracking for amateurs and hobbyists. However, in its current iteration, the range of motion allowable by this tracking system is highly limited as the entire photodiode plane must remain in line of sight of the lighthouses in order for tracking to occur. In particular, this restricts the user's ability to look around a given scene, as head rotation is likely to cause occlusion of at least one of the photodiodes, thus disrupting the connection and terminating tracking.

In this work, we describe a positional tracking implementation using two VRduinos in order to extend the viable range of the tracking system. Additionally, we demonstrate additional robustness through the inclusion of multiple photodiode arrays arranged in a 3D constellation.

2. Related Work

Current industry approaches to the positional tracking problem span a space of solutions, from HoloLens [1] to Tengo [2]. However for hobbyists and amateur consumers, these solutions are generally out of scope.

The HTC Vive, released last year, pioneered the lighthouse positional tracking system for the consumer market. It's performance, both quantitative and qualitative, were characterized by [3], whose results we used to benchmark our own system.

In developing our Homography and Levenberg-Marquardt computations or a 3D photodiode constellation, we used the 2D computations of Professor Gordon Wetzstein as a guide[4] [5]. We also explored a variety of textbooks and journal articles on camera calibration and positional tracking, but were unable to find any that actually implemented the computations for a 3D constellation.

3. Methods

Our implementation of extended positional tracking consisted of three major components:

1. Design and testing of optimal physical arrangement.
2. Streaming and synchronization of data from independent VRdiuno boards.
3. Positional computation and mapping.

3.1. Physical System

3.1.1 Initial Design:

Before designing our physical system, we characterized the range of rotation of a single VRduino board while using the lighthouse positional tracking system. Z-axis rotation did not occlude any photodiodes and X-axis rotation seemed sufficient to look up or down; however, limited Y-axis rotation significantly hampered the user's interaction with the virtual environment. To maintain reliable performance, we found that Y-axis rotation was constrained to approximately $[-60^o, 45^o]$[1], providing only 30% of a full 360^o rotation. The off-center Teensy placement results in a difference of negative and positive rotational ranges. The Teensy sits closer to the rear two photodiodes on positive rotations, so they are more quickly occluded. Based on this test, we decided to position both VRduinos $\pm 45^o$ off of the Z-axis. This provides a 15^o overlap in the range of each VRduino, leaving room for hand off when rotating and providing more photodiodes for computation when facing directly towards

[1]The right handed OpenGL coordinate system is used

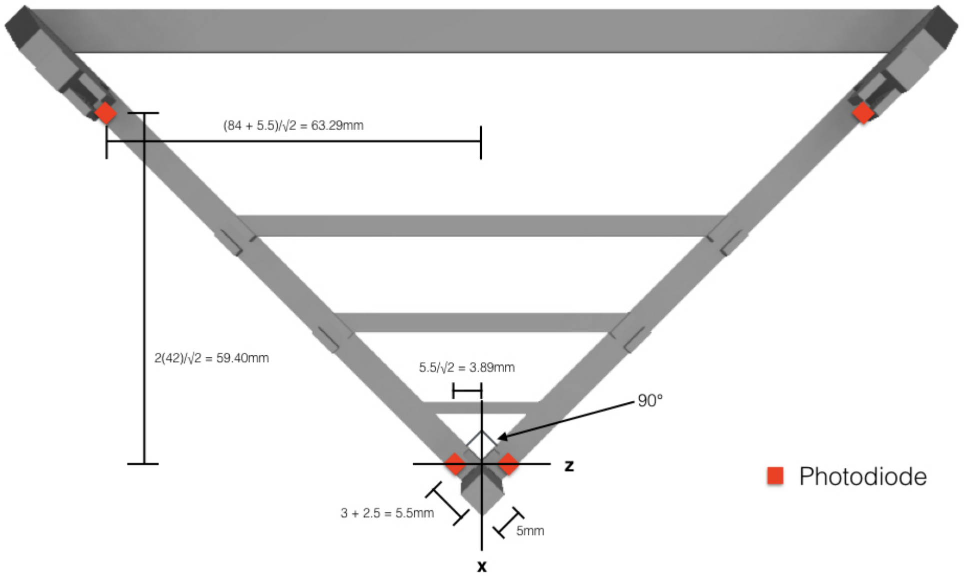

Figure 1. Location of each of the photodiodes in the xz plane.

the lighthouse.

The jig to hold these two VRduinos was modeled in Autodesk Fusion 360 (chosen due to a free educational license and a cloud-based environment allowing for easy collaboration and cross-platform compatibility) to ensure precise placement of each VRduino board. A 5mm by 5mm rectangular prism separates the two boards. Figure 1 illustrates the placement of each of the photodiodes in the xz plane. Y-axis coordinates of ±25mm were used.

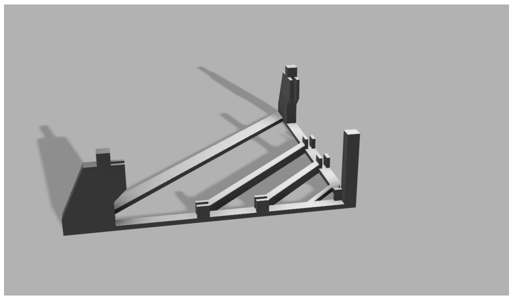

Figure 2. Half of the 3d jig (rendering)

3.1.2 Build Process and Design Iterations

An Ultimaker 2 printed all iterations of the jig, which each cost less than $1.75. While we initially considered laser cutting and assembling a multi-piece jig, we chose a 3D printed part due to assembly concerns and time constraints. Still, we iterated our 3D jig several times to address tolerance and accuracy concerns.

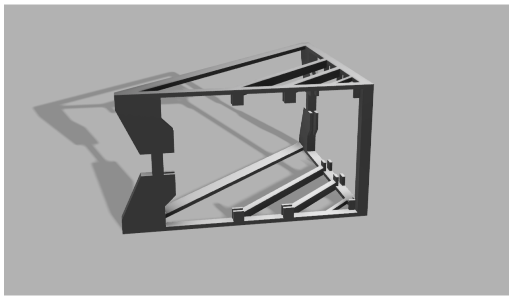

Figure 3. The full 3d jig (rendering)

The initial print was made with tolerances corresponding to the specified resolution of the printer. It consisted of two clips on the bottom and one on each side. Unfortunately, the clips on the side where the boards meet had to be removed. Structure wobble during printing due to the small support (5x5mm) produced large inaccuracies. One small tab was left at the bottom of each side to hold the back side of the VRduino. We added more support on the back clips to avoid wobble during printing. Additionally, we increased tolerances to 0.20mm on each side of the board's width and 0.25mm on each side of the board's length, giving us 2.0mm between clips and a 90.50mm space to hold the length of the board. These tolerances created a good fit. After the next print, we determined that the boards still wiggled too much on top. We printed a small cap with two clips in an attempt to solve this; however, the problem persisted. Eventually, we created the symmetric structure used for the final demo.

The final printed jig consisted of identical top and bottom pieces (Figure 2) with crossbeams. We fabricated both pieces simultaneously, requiring a six hour print. These two pieces were assembled together (Figure 3) and rubber bands running between the two sets of crossbeams ensured the VRduinos had a snug fit into the structure and remained immobile, depicted in Figure 5. No vertical tolerance was used to ensure the rubber bands applied constant pressure to the VRduinos, holding them in place. Additionally, the rubber band solution made for easy assembly and disassembly. Figure 4 shows the final 3D jig with the VRduinos. This final configuration assured the photodiode locations would be exact and immobile.

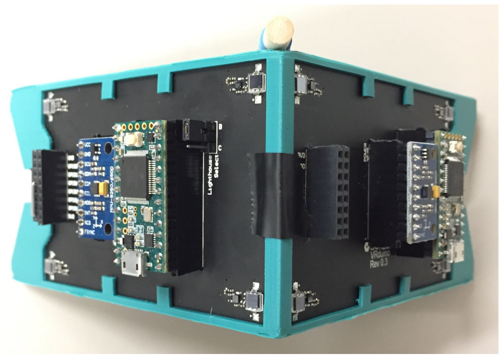

Figure 4. The final 3d printed jig with VRduinos attached

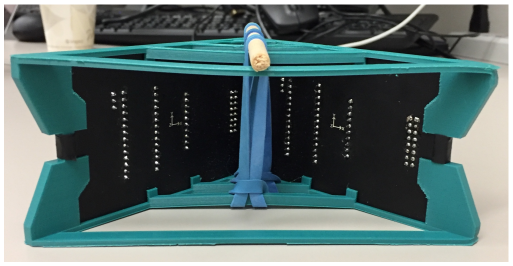

Figure 5. Rubber bands hold the top and bottom pieces of the jig together

3.2. Streaming and Synchronization

We considered several options to accumulate photodiode data from all eight photodiodes, spread across the two boards. Initially we wanted to used a master-slave configuration for the Teensys, as shown in Figure 6. A slave Teensy would send its photodiode data to a master Teensy, which would perform all computations and send the result to a computer.

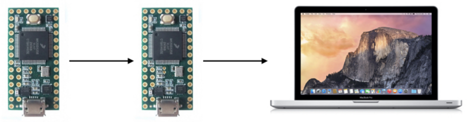

Figure 6. Data flow from a slave Teensy to a master Teensy to the computer

Unfortunately, we ran into some hardware problems using the Teensy. First, all eight general purpose interrupt vectors were already used to get photodiode readings from the horizontal and vertical laser sweeps. While we still could configure one Teensy as an SPI slave, a precisely timed solution seemed less obvious. However, we were never able to explore this approach. We quickly noticed that the stream of data from the Teensy to Theo's 2013 Macbook pro was significantly throttled. We found that removing all logic after the projection computation ameliorated this problem, though the Teensy still required several power cycles to stream properly. These observations caused us to attribute the streaming lag to the Teensy itself. As a result, we decided to stream all data individually from both Teensys to the computer over a 1Mbps serial connection, as shown in Figure 7. We felt that streaming data to the computer would have the highest probability of success and allow us to iterate quickly. Once we had a sufficiently fast stream of data, we never needed to power cycle the Teensy or re-upload code.

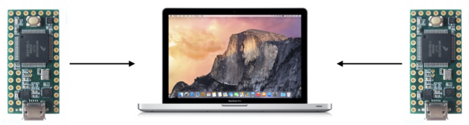

Figure 7. Data flow from each Teensy individually to the computer

Much later, we realized our data rate problem was confined to Theo's computer – Georgia's computer streamed in the data at a much faster, more consistent rate. Unfortunately Georgia's computer only had one USB port, and we realized this too late into our implementation to start over. Interestingly, Theo's computer streamed in data at an acceptable rate using the ”screen” terminal command; however, these results were reproducible by neither Arduino's serial monitor nor javascipt's serial library. After a brief foray into complex workarounds, we determined that Virtual Windows 7 machine running on Theo's computer could produce comparable data rates to Georgia's computer. We never determined the root cause of the issue. In the Future Work section, we discuss what our approach would have been given this knowledge (or if we had another week).

In our final implementation, each Teensy still polled the photodiodes as in the HW6 code. However, instead of waiting for all diodes to be ready, it computed X and Y projections and sent this data as soon as both horizontal and vertical sweep data from an individual photodiode was ready. In addition, these data were tagged with the diode number and the board number. A Teensy determined its board number from the value of the GPIO 2 pin input, which was checked in the setup. Board zero has this pin pulled to ground.

All of this data was sent as a continuous stream over two 1Mbps serial connections to our node.js server. A function called on each data received event accumulated the diode data. As soon as data was received from four unique diodes, these four were used for positional tracking. We explain the advantages and disadvantages of this implementation in the following sections.

3.3. Positional Computation

We used the HTC-Vive Lighthouse system to track the position of the VRduinos in the 3D scene with the Homography and Levenberg-Marquardt estimation methods. Since the problem of optics-based positional tracking is similar to the reverse camera celebration method, in which we are only concerned with the 2D plane of the image, most literature on the implementation of these methods assumed a 2D photodiode array. Consequently, we derived the full 3D equations ourselves before implementing them in code. Here we include a brief derivation, using the EE 267 Lecture 11 and Lecture 12 slides of Professor Gordon Wetzstein as a starting point and following the notation therein [4] [5].

3.3.1 Homography Pose Derivation

The Homography calculation assumes as an input the relative physical coordinates of the diodes in question as well as the 2D projection of the photodiodes onto the "camera plane." The calculation of this 2D projection is not described here, but can be found in [4].

Given these inputs, the 3D view position, denoted p^{view}, is the solution to the following matrix equation:

$$p^{view}_{x,y,z} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} p^{obj}_x \\ p^{obj}_y \\ p^{obj}_z \\ 1 \end{bmatrix} \tag{1}$$

For the remainder of this derivation we assumes that the camera and lens are properly aligned ($c_{x/y} = 0$) and that the focal-length is unit distance ($f_{x/y} = 1$). However, due to our 3D photodiode constellation, we cannot assume $p^{3D}_z = 0$, as done in most previous literature (see discussion in Section 2). With these assumptions, we can simplify Eq 1 to Eq 2 in order to solve for the homography matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} = s \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & 1 \end{bmatrix}$$

$$p^{view}_{x,y,z} = s \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & 1 \end{bmatrix} \begin{bmatrix} p^{obj}_x \\ p^{obj}_y \\ p^{obj}_z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} p^{proj}_x \\ p^{proj}_y \end{bmatrix} = \begin{bmatrix} \frac{p^{view}_x}{p^{view}_z} \\ \frac{p^{view}_y}{p^{view}_z} \end{bmatrix} = \begin{bmatrix} \frac{s(h_{11}p^{obj}_x + h_{12}p^{obj}_y + h_{13}p^{obj}_z + h_{14}}{s(h_{31}p^{obj}_x + h_{32}p^{obj}_y + h_{33}p^{obj}_z + h_{14}} \\ \frac{s(h_{21}p^{obj}_x + h_{22}p^{obj}_y + h_{23}p^{obj}_z + h_{24}}{s(h_{31}p^{obj}_x + h_{32}p^{obj}_y + h_{33}p^{obj}_z + h_{14}} \end{bmatrix} \tag{2}$$

Since the homography matrix for which we are solving has eleven unknowns, and since each photodiode provides three data points ($p^{obj}_{x,y,z}$), four photodiodes are required for solution.

Although more than four diodes can be used for the calculation, this over-constrains the problem and may result in a non-soluble system if there are disagreements in the data. To avoid this concern, we restricted our implementation to use the data from only four photodiodes in each calculation, as described in Section 3.3.3.

3.3.2 Levenberg-Marquardt Derivation

After obtaining an initial position estimate from Homography, our implementation used ten iterations of Levenberg-Marquardt nonlinear optimization to achieve our final positional estimate. As in Homography, we freshly derived the equations for a 3D photodiode constellation. We will not include the full derivation of the Jacobians here, but will simply present the extended homography matrix (the Jacobians are explicitly computed in server\flevenbergMarquardt.js in our code). Using this, the functions f and g and the Jacobians J_f and f_g can be found simply using the same procedure as in the 2D constellation case [5]. Note, however, that the dimensions of the Jacobians will change:

$$J_f : 8 \times 9 \rightarrow 8 \times 12$$
$$J_g : 9 \times 6 \rightarrow 12 \times 6$$

$$\begin{aligned}
h_1 &= \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(theta_y)\sin(\theta_z)\\
h_2 &= -\cos(\theta_x)\sin(\theta_z)\\
h_3 &= \sin(\theta_y)\cos(\theta_z) + \sin(\theta_x)\cos(\theta_y)\sin(\theta_z)\\
h_4 &= t_x\\
h_5 &= \cos(\theta_y)\sin(\theta_z) + \sin(\theta_x)\sin(\theta_y)\cos(\theta_z)\\
h_6 &= \cos(\theta_x)\cos(\theta_z)\\
h_7 &= \sin(\theta_y)\sin(\theta_z) - \sin(\theta_x)\cos(\theta_y)\cos(\theta_z)\\
h_8 &= t_y\\
h_9 &= \cos(\theta_x)\sin(\theta_y)\\
h_{10} &= -\sin(\theta_x)\\
h_{11} &= -\cos(\theta_x)\cos(\theta_y)\\
h_{12} &= -t_z
\end{aligned} \tag{3}$$

3.3.3 Development and Testing

For both Homography and Levenberg-Marquardt, our implementations assumed that each position estimate was calculated from the data of exactly four photodiodes. As discussed in Section 3.3.1, this is the minimum number of photodiodes necessary to fully constrain the system, but the question does arise whether, since our 3D constellation contains eight photodiodes, the accuracy of the estimate might not be improved by included the data from all available photodiodes.

This was, in fact, our original method, which we later abandoned or two primary reasons:

1. **Computational Efficiency:** As an example, let us consider homography, in which the solution requires the inversion $n \times 12$ matrix in the 3D constellation case ($n \times 8$ in the 2D case), where n is the number of photodiodes included in the computation. Assuming that we use the pseudo-inverse ($A^+ = A^T(AA^T)^{-1}$) rather than the true inverse (since requiring a square matrix predefines the number of photodiodes used), the complexity of this single pseudo-inversion grows as $O(12n^2) + O(n^{2.373})$. It is possible that this complexity growth might be manageable; however, given the time constraints of the project, we elected to avoid the risk.

2. **Implementation Complexity:** In our system, we do not wish to mandate that all eight photodiodes be in line of sight of the lighthouse for positional updating to occur – rather, we wish to use the data available to compute a positional estimate over the greatest possible range of user movement. Consequently, if we wished to compute each positional estimate using all diodes currently in line of sight, we would have to develop a system with flexible input, cutting off if fewer than four diodes were available, but accepting any number above four and consequently performing computations of variable dimensionality. This is certainly achievable and would be an interesting extension to our study. However, as it introduced additional opportunities for error without guaranteed improvement, we opted for the lower complexity of a fixed-input system.

However, even though only four photodiodes are being used in each calculation, the presence of all eight diodes does add robustness to the system due to the data-diversity introduced by pulling data from the diodes in a pseudo-random fashion (see Section 3.2 for a detailed discussion of the data streaming process). Since the computation is done on an independently selected four-diode array in each time-step, the presence of a exponentially-decaying moving-average low pass filter results in a *de facto* averaging across time of multiple four-diode array combinations, adding robustness to the system by decreasing the effect of bias or error in the estimation from any given four-diode array.

In theory, this robustness through multiple arrays could be further increased by computing multiple permutations for each time-step and filtering the results. We briefly attempted this implementation by requiring the data of five diodes rather than four, running the Homography and Levenberg-Marquardt computations for each of the five possible four-diode permutations, and averaging the results. In our implementation, however, these additional computations appeared to introduce too much lag into the system, causing unpredictable jumps in the final rendering. Thus, while we believe using multiple permutations per time-step could improve performance if implemented properly, we chose to restrain our system to a single permutation per time-step, trusting on the inherent randomness of the data-selection process and the sharpness of the low pass filter to improve the system's robustness.

In order to thoroughly test our estimation methods throughout development, we developed two fully-testable phases before completing the final implementation. In the first phase, we implemented standard 2D-constellation Homography and Levenberg-Marquardt in JavaScript. We then tested our 3D derivation by implementing the 3D estimations in JavaScript, but setting $p_z^{obj} = 0$ for all photodiodes, allowing us to run the same tests cases as the 2D case, where the $p_z^{obj} = 0$ assumption is implicit, and check our results numerically. Finally, we substituted in 3D p^{obj} inputs and tested qualitatively for smoothness and accuracy.

4. Evaluation

4.1. Latency in the System

Our greatest design concern throughout the development process was latency in the photodiode data. In order to ensure accurate estimates, all data used in a single computation should be pulled from a single laser sweep of the lighthouse. However, since our system sends diode information to the server immediately upon arrival, this was not guaranteed in our system. In practice, since the laser sweeps at 60 Hz, it is unlikely that using data from immediately adjacent sweeps would cause noticeable disturbances in the rendering.

We measured the time it took to get the four diode measurements over several seconds of system runtime. These measurements indicated an approximate mean update time of 40ms. As a result, we were likely pulling data from two to three adjacent laser sweeps. While this update time is suboptimal, low pass filtering allowed us to maintain a good visual experience, discussed in the next section.

The wide range of the mean update time, which varied from less than 1ms to a little over 100ms, was concentrated heavily around the mean. These statistics indicate the possibility of serial's buffer adding delay into the system. Single digit millisecond update times should be regularly achievable with more attention paid to timing of the entire system. The use of a virtual machine likely added additional lag into the system as well, though this was never characterized.

After getting all diode ticks, the Homography algorithm and twenty iterations of the Levenberg Marquardt for pose estimation consistently ran in an negligible time when using only four photodiode positions, suggesting that we have room to do more complex post-processing of the data, discussed in the Future Work section.

4.2. Qualitative Discussion of Results

We achieved our initial goal of extending the range of motion over which the lighthouse is able to realiably track, attaining a rotational range of $195°$, a 90.5% increase over the original, single-board range of $105°$.

In order to reduce jitter such that it did not noticeably detract from the VR experience, we used exponentially decaying moving averages to implement sharp low-pass filters in all translation components (X, Y, and Z). For both X and Y, we were able to weight the filters to remove almost all noticeable jitter while maintaining reasonable update rates – even sudden movements were tracked and displayed with reasonably speed in the X and Y direction.

The Z component, however, contained much more jitter than X and Y, and in order to a reasonably smooth experience, we were forced to use a filter which created lag in the update. Thus while our system does track movement in the Z direction and renders that movement fairly smoothly, the render displays a small but noticeable delay.

In contrast to the translation data, the rotation data showed far too much variance for reliable rendering, even with extreme low pass filtering. Due to time constraints, we eventually abandoned rotational tracking and confined our system to a purely translational one. We discuss possible alternatives for rotational tracking in Section 5.2.

5. Discussion

5.1. Major Challenges

1. Serial stream issue – Discussed in Section 3.2 and 5.2.1

2. Synchronizing data – Discussed in Sections 3.2 and 5.2.1

3. Lack of literature on extending Homography and Levenberg-Marquardt to 3D constellations – Discssed in Section 3.3

4. Precision of measurements – slight errors in photodiode coordinates caused severe drop in performance.

5.2. Future Work

Our project presents opportunities for improvement at many levels of the system, from the hardware through the integration into a virtual reality experience. We briefly outline some of the most enticing extensions below.

5.2.1 Hardware

We do not believe our solution of streaming all data to the computer is optimal. As mentioned previously, this solution was born from a combination of time constraints and hardware troubles. If we were to start over, we would link the Teensys together and optimize the connection to the computer. To communicate Teensy to Teensy, we would likely try SPI due to it's ability to support higher clock speeds. However, since no general purpose interrupt vectors are available, we would still have to come up with a means to ensure timing. The default SPI library does not include a slave mode, so this approach might require significant firmware programming. Still, we believe accumulating all photodiode ticks on one Teensy is optimal to ensure all photodiode data are part of one sweep. However, we believe computation of position on the computer affords many benefits due to increased processing power. The increased computing resources allow for more iterations of the Levenberg Marquardt algorithm and more sophisticated filtering.

In our project, we streamed serial data as a string rather than as a bit stream due to time constraints. Converting this data to a bit stream, sending this, and building a more complex parser on the computer side would offer significant performance benefits in communications speed and latency of the entire system.

In addition, occlusion of the photodiodes by the Teensy currently limits Y-axis rotational range of the VRduino.

Removing the headers and soldering the Teensy directly onto the PCB provides a quick remedy. Our jig leaves space for the Teensys to be on the back of the board instead, though this would require soldering headers to the top (rather than the bottom) of the Teensy or wiring to maintain the correct footprint.

5.2.2 Diode Selection

Our current method of selecting the diode measurements to use – selecting the first four measurements – optimizes for speed only. We implemented a permutations approach with five photodiodes, discussed in Section 3.3.3, but we did not explore many other potential algorithms to choose which photodiode measurements we use in the computation.

With more precise timing, we could have if each photodiode had a measurement within a certain laser time frame. Given this information, we could permute over more data or select particular photodiodes to give a more robust estimation. For example, if we faced the lighthouse directly, we would want to use the photodiodes on the sides if we were to pick four.

In addition, we could have used collected time stamp data to avoid making updates based on bad data. We already do data rejection and filtering in the LM implementation, though not beforehand. For example, if we knew two readings were updated several laser sweeps frames apart, we would never run the computation in the first place. Complementary filtering with IMU data or predictive filtering when using this method could help prevent noticeable jumps or lag in the scene.

5.2.3 Filtering

Our current implementation performs highly naive filtering on the translation data and would likely be substantially improved by more robust filters. In particular, a custom designed filter might be able to appropriately reduce jitter without introducing the lag currently noticeable along the Z-axis.

While we did not do any tests in this area, it is also possible that predictive filtering, such as extended Kalman filtering, might stabilize the Levenberg-Marquardt rotational data enough to allow its incorporation. However, as we did not experiment with this, we do not have specific recommendations to explore.

5.2.4 Demo Experience

Had we had more time, we would greatly have liked to include rotation in our demo. Despite the instability seen in the Levenberg-Marquardt rotational data, it should be possible to achieve a reasonably immersive experience by implementing gyro rotation. The only anticipated complication to this approach is the presence of two VRduinos, and consequently of two gyros. However, since the gyro measurements are taken on each VRduino independently, the quaternions can simply be sent to the server, converted into Euler angles, averaged, and the then used for rotational tracking.

A particularly exciting extension for this project would be use two lighthouses positioned at opposite sides of a room to allow full $360°$ rotation. We anticipate the hand-off (switching from one lighthouse to another) to be by far the most challenging element of this extension. However, as the VRduino can already distinguish between the two lighthouses, we believe this should be surmountable by using the region of overlap between the two lighthouses to calculate the transformation between the coordinate axes of the two lighthouses.

6. Acknowledgements

We would like to thank Professor Gordon Wetzstein of Stanford University for all his instruction and guidance throughout this project. We would also like to thank Robert Konrad, Hayato Ikoma, and Keenan Molner for their general support and guidence, and especially for their assistance with hardware difficulties and replacements.

References

[1] J. Ashley. How hololens sensors work. 2015.

[2] J. Durbin. Google: Wireless positional tracking solved, but heat still a problem for vr. 2016.

[3] D. C. Niehorster, L. Li, and M. Lappe. The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research. *i-Perception*, 8(3):2041669517708205, 2017.

[4] G. Wetzstein. Ee 267: Positional tracking i. 2017.

[5] G. Wetzstein. Ee 267: Positional tracking ii. 2017.