SteamVR Tracked Device

Kevin K. Balke, *Student Member, IEEE*

***Abstract*—Modern virtual reality systems require high-precision position and rotation tracking of displays and controllers in order to achieve immersion. This tracking technology has many applications both within and outside of virtual reality, and one such system is simple to implement and highly capable, but has no open-source device implementation available. I propose a simple, low-cost, open-source embedded system and algorithm implementation that achieves tracking with components of the HTC Vive virtual reality system. The system implementation described here achieves positional accuracy of $\pm 1mm$ at a rate of $\sim 60Hz$, is low-cost, and is open-source.**

***Index Terms*—Axis-Aligned Bounding Box, Head-Mounted Display, Light field, Lighthouse, Play Space, Pose, Tracked Device, Virtual Reality**

I. Introduction

A. History

(c2) Virtual Reality (VR) systems are families of devices that work together to present an artificial world environment to a user in a believable, "immersive" way. Achieving a convincing level of realism requires powerful graphics capability, high-resolution displays, and comfortable interface devices. These requirements made high-quality VR an impractical goal right up until a series of recent innovations brought the capability to handle VR rendering workloads in real-time to the consumer segment. This improvement in foundational technologies, coupled with the development efforts of major players such as Oculus, Valve, and HTC, brought two high-end VR systems and their corresponding software platforms to market in 2016: the Oculus Rift, and the Valve/HTC Vive.

In order to make these systems possible, several engineering challenges had to be overcome. The first was graphics horsepower; NVIDIA's existing 900-series GPUs and AMD's R9-200 series were only barely capable of handling the required 90Hz refresh rates, 2K display resolutions, and 20ms latency required to keep users from feeling nauseated in high-fidelity gaming experiences. The latest-generation NVIDIA 1000-series and AMD RX-400 series are designed to handle VR workloads and achieve the required performance[4,5,6].

The next challenge was to develop natural interface devices that don't encumber the user's gameplay, and support a range of input strategies on the same hardware, giving developers creative license to build diverse experiences. The Oculus Touch controllers and Vive hand controllers incorporate high-speed motion sensors, capacitive touch, and haptics to increase the "hand presence" or sense of the hands being a natural part of the experience[1] The head-mounted displays (HMDs) of these and other VR systems were made light and comfortable thanks to modern display technologies to enable a wide range of motion for the user's head. OLED display panels provide a contrast ratio that does not overstimulate the eyes in dark scenes while achieving acceptable frame rates[3].

The final challenge was to develop a system for precision motion tracking within the "play space" (area of a room the user has designated for VR experiences) that would enable the user to look over, under, and around objects, walk around the environment, pick up and interact with objects, and the like. The accuracy, speed, and size of the tracking mechanism directly impact the quality of the VR experience, the goal being to enable tracking of the full range of human motion and speed in the hands, head, and eventually, the whole body.

The Oculus Rift system achieves motion tracking with the "Constellation" system[2], wherein USB webcams stream video of the tracked HMD and Touch controllers to the application processor on the gaming computer, and image processing algorithms are applied to estimate the pose of the tracked devices from the video. The HMD and controllers are covered in infrared LEDs that emit pulses of light to identify themselves to the cameras for correct pose estimation. This system suffers from high computational cost and USB bandwidth restrictions, and does not scale well to large play spaces where more cameras are required to track motion.

The HTC Vive achieves motion tracking with the "Lighthouse" system[7], the design and implementation of which is the subject of this paper. Reverse-engineering the tracking strategy for the Vive has enabled the creation of an open-source, completely free-to-use implementation of a tracked device, with all tracking algorithm computation performed on the tracked device itself.

B. Global Constraints

(c4) The hard constraints on the tracked device performance implemented here are driven by the performance requirements for comfortable VR, and by the environmental constraints of an indoor tracking system. These constraints are:

- Produce pose at $\geq 90Hz$ and $\leq 20ms$ latency
- Be accurate to $\pm 1mm$
- Rely upon only 1 lighthouse
- Be robust to changes in ambient lighting

The first constraint is not achievable with the system architecture presented here, for reasons that will be discussed

Kevin K. Balke is with the University of California at Los Angeles, Los Angeles, CA 90024 USA (e-mail: kbalke@ucla.edu).

subsequently. The rest of the constraints are met.

II. Motivation

The Lighthouse tracking system, as I will describe subsequently, is a very capable and accurate room-scale tracking system that has applications both in and out of Virtual Reality systems. Other applications include tracking robots in coordinated actions, tracking large numbers of physically interacting objects in experiments, and including many more objects into Virtual Reality environments. The Lighthouse tracking technology, however, is proprietary, and there exists no open-source device implementation that can be quickly adapted to build specialized tracked devices that leverage the performance, simplicity, and relative low-cost of the Lighthouse system.

In order to address this need, the system implementation described here is simple to construct and operate, and entirely open-source.

III. Approach

A. Team Organization

(c1) Kevin Balke is the only team member. All system planning, design, implementation, and testing were performed by him.

B. Plan

(k1 & k2) First, the functional characteristics of the lighthouse had to be determined. The lighthouse produces a spatiotemporally indexing light field, which can be measured and characterized with a photodiode and an oscilloscope. This characterization also leads to an understanding of the mechanics of the lighthouse itself. After this characterization, a circuit to condition the signal from the photodiode had to be designed and implemented, once for each sensor on the tracked device. These conditioned signals contain the timing information which determines the pose, and this timing information must be somehow extracted. A digital system had to be designed to extract this timing information, and convert it into a pose.

C. Standard

(c3) There exists no publicly-available written standard for the lighthouse tracking system. However, the physical implementation of the system in the HTC Vive serves as a *de facto* standard against which the implementation described here was designed.

1) The Lighthouse System

The Lighthouse tracking system consists of two main components:

- a lighthouse (or lighthouses)
- a tracked device (or tracked devices)

2) The Lighthouse

The *lighthouse* is a device which produces a spatiotemporally indexing light field that sensors on the *tracked device* can measure to determine position and orientation with low latency and high accuracy. The first generation lighthouses that ship with the HTC Vive virtual reality system consist of two rotary line laser guns and an LED panel, all of which emit infrared light. The two rotary guns are rotating at 60 revolutions per second in planes perpendicular to one another, and are phase-locked so as to take turns sweeping across the field of view of the device (180 degrees out-of-phase). The LED panel flashes at $120Hz$, and is phase-locked with the rotors so as to produce short flashes in between rotary gun sweeps. A diagram of the lighthouse construction is given in Fig. 1.

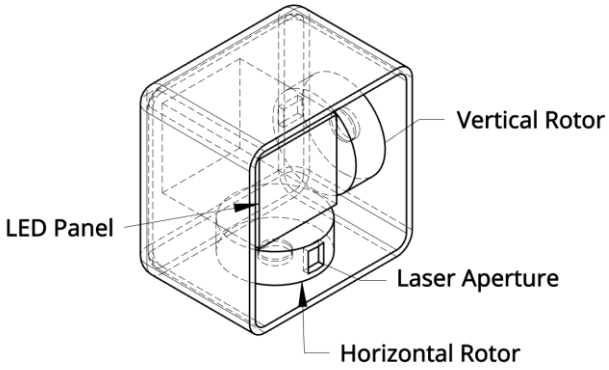

Figure 1: The lighthouse (original work)

Note that the vertical rotor is facing the back of the device when the horizontal rotor is facing forwards. This phase condition is enforced by the control loop running the two rotor motors; if the lighthouse is rotated while it is operating (thus imparting an angular moment that disturbs the phase lock), the laser guns and LED panel will stop emitting until the system has regained this condition.

As a result of this design, an infrared sensor placed somewhere in the field of view of the lighthouse will produce a signal with the shape given in Fig. 2 (blue signal).

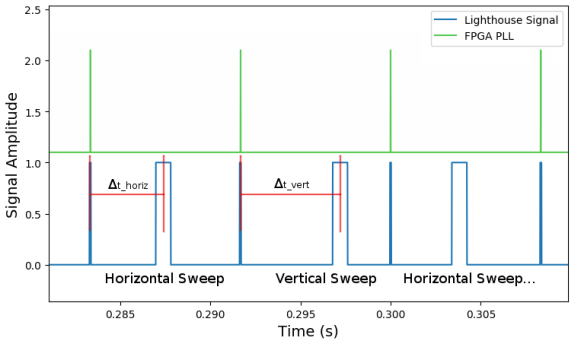

Figure 2: Lighthouse signal timing (original work)

The green signal in Fig. 2 is the output of the tracked device's internal 120Hz PLL. It is phase-locked against the thin synchronization pulses. Δt_horiz and Δt_vert are the elapsed times from the phase-locked internal 120Hz pulse to the sweep pulse (wider blue pulse) for each horizontal and vertical timing window, respectively. A successive pair of these measurements for each sensor describes a set of rays, projected from the lighthouse, upon which each corresponding sensor of the device must lie. The pitch and yaw angles of these rays are calculated, in radians, as:

$$\theta_{yaw} = \Delta t_{horiz} \times 120 \times \frac{\pi}{2}; \theta_{pitch} = \Delta t_{vert} \times 120 \times \frac{\pi}{2}$$

When a sensor is dead front and center, the angles for that sensor's ray are both π/2.

The Lighthouse system, and the HTC Vive for that matter, is capable of using more than one lighthouse simultaneously in the tracking space. In this scenario, one lighthouse is designated as the "master", and the additional lighthouses are "slaves". The master sets the timebase (the other lighthouses align their 120Hz sync pulses against the master optically), and then each lighthouse takes turns round-robin-style sweeping their rotors through their FOV.

3) The Tracked Device

The tracked device is a set of light sensors, as described above, arranged on the surface of some physical object. In the case of the HTC Vive system, the tracked devices are the HMD (Head-Mounted Display) and the two handheld controllers. The small indentations covering the surfaces of these devices are apertures behind which sensors are located (Fig. 3).

Figure 3: The HTC Vive handheld controller (original work)

The arrangement of the sensors on the device is known *a priori*, and as such, the problem of determining the position and orientation of the device is to find a position and orientation for which the linear error (Euclidean distance) between the estimated positions of the sensors and the nearest points on their corresponding rays is as small as possible. In the HTC Vive system, each device also has an IMU (Inertial Measurement Unit) which is used to determine the high frequency pose changes that are too fast or minute to reliably capture with the light-based tracking method. Because the implementation presented here does not incorporate an IMU, the maximum tracking rate is bounded at the lighthouse sweep frequency of $60Hz$.

D. Theory

(c4) Simple circuit theory was applied to develop the analog frontend for the photodiodes. This frontend uses an open-loop amplifier and a low-pass filter to threshold the photodiode output against 2/3 of its average value, in order to meet the global constraint of robustness to changes in ambient lighting. The frontend design is given in Fig. 4.

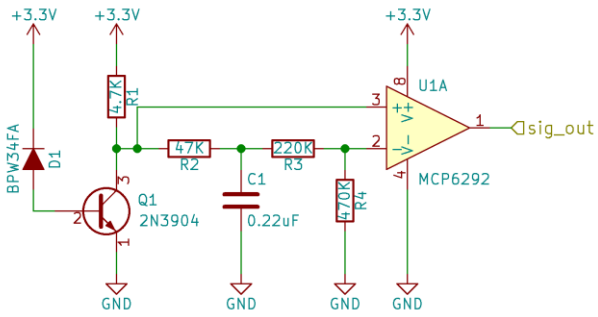

Figure 4: Analog frontend (original work)

Digital circuits design principles were applied in the development of the timing capture and PLL blocks of the FPGA implementation.

CAD and 3D printing were used to develop a mount for one of the photodiodes on the device. This is to meet the non-coplanar constraint described in (m1).

(m1) Linear algebra and geometry form the basis of the algorithm implemented on the microcontroller. All transformations are performed using matrices, quaternions, and vectors. Basic trigonometry is used to convert the captured signal timing into rays in 3-space. The optimization problem solved by the algorithm is given in Fig. 5.

Let $\vec{s}_i$ be the position, relative to the sensor array orthocenter, of the i^{th} sensor of the tracked device in the tracked device's coordinate system. Let:

$$\vec{r}_i = \langle cos(\theta_{vert,i}) \times cos(\theta_{horiz,i}), cos(\theta_{vert,i}) \times sin(\theta_{horiz,i}), sin(\theta_{vert,i}) \rangle$$

be the direction vectors of the rays projected from the lighthouse (chosen, for simplicity, to be located at $\vec{0}$), in terms of the measured angles $(\theta_{vert,i}, \theta_{horiz,i})$ from the sensor array.

These rays can be written, equivalently, as the vectors from the lighthouse to the sensor positions in the world coordinate system, $\vec{r}_i = \mathbf{A}\vec{s}_i + \vec{b}$, where $\mathbf{A}$ is the rotation matrix and $\vec{b}$ is the translation vector that together take a point in the device coordinate system to the world coordinate system.

Estimate the true device position and orientation, $(\mathbf{A}, \vec{b})$, as $(\mathbf{A}', \vec{b}')$, such that:

$$(\mathbf{A}', \vec{b}') = argmin_{(\mathbf{A}', \vec{b}')} \sum_i \| proj_{\vec{r}_i^{\perp}} (\mathbf{A}'\vec{s}_i + \vec{b}') \|$$

Figure 5: Optimization problem statement (original work)

The minimum number of sensors such that this problem has a definite solution is 4, with an additional constraint that one sensor be non-coplanar with the other 3. This constraint prevents the occurrence of an undecidable degenerate case wherein all sensors have either the same horizontal or the same vertical angle, and two of the sensors have the same of the other; in this case, there are two solutions, separated by a 180-degree rotation, which are equally correct. Moving one sensor out of the plane prevents this case from occurring.

There exists no closed-form solution to the problem given in Fig. 5, and so I propose an iterative solution. This solution takes the last known device state, given as $\boldsymbol{A_{t-1}}, \vec{b}_{t-1}$ (as in Fig. 5), and computes the current device state, $\boldsymbol{A_t}, \vec{b}_t$. This iterative solution is taken in two steps.

1) The Translation Step

First, the target rays (Fig. 6, in yellow), upon which each sensor must lie in order for the error bound given in Fig. 5 to be minimized, are computed ($\vec{r}_i$, as given in Fig. 5). The nearest points to the last known device state sensor positions

on each corresponding ray are computed. The deltas between the last device state sensor positions and these target positions are shown in red in Fig. 6. The average of these deltas is shown in white. This average is applied as a translation to the last device state (Fig. 6, in brown) to bring it close to the current device state (Fig. 6, in turquoise). The axis-aligned bounding boxes around the last device state (Fig. 6, in green), and the nearest points on each corresponding ray (Fig. 6, in blue) give an estimate of the last and estimated current device sizes. The ratio of the diagonals of these AABB's estimates the change in size of the estimated current device state from the last device state that results from moving closer or further away from the lighthouse (Fig. 6, in grey) in this solver step. A correction translation (Fig. 6, in white, at the end of the long white vector) is applied along the line from the lighthouse to compensate for this change in size (moving the estimated device state closer to the target region).

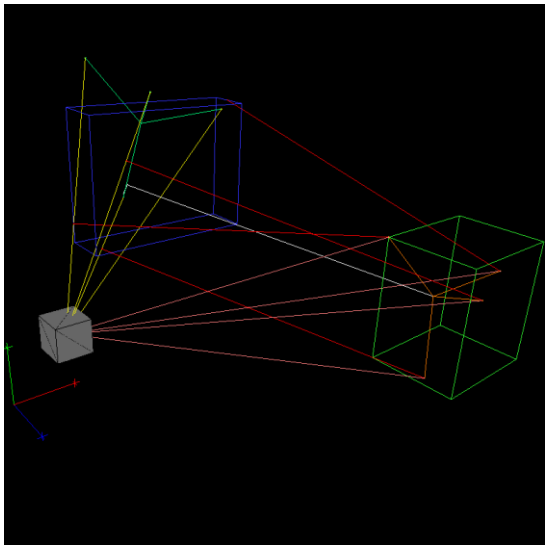

Figure 6: Translation step (original work)

2) The Rotation Step

Next, the deltas to the nearest points on the rays, after the translation from the previous step is applied, are taken (Fig. 7, in green). Then, the rotation that will close the angle A,B,C, where A is the estimated device state sensor position, B is the estimated device state origin, and C is the nearest point on the target ray, is computed. The axis around which this rotation is performed is given by $\overrightarrow{AB} \times \overrightarrow{BC}$. The average of these rotations is computed, and applied to the estimated device state. The axis of this average is shown in Fig. 7, in white.

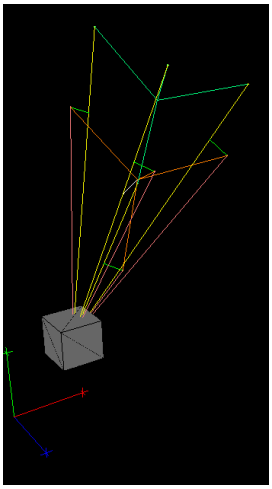

Figure 7: Rotation step (original work)

These two steps are applied one after the other repeatedly, until the error bound given in Fig. 5 is satisfactorily small.

(m2) Basic kinematic principles were used to enforce constraints on the output of the algorithm; the error must remain reasonably bounded, the device state cannot be behind the lighthouse (and thus out of sight of the lighthouse), and the device cannot move faster than a human operator could move it. As a result of bugs in the device implementation, the system will occasionally report incorrect pose information. These constraints are applied to filter out these incorrect poses (these dropped data frames reduce the tracking frequency to slightly below $60Hz$, on average).

E. Software/Hardware

(m3 & k3) Vim, GNU GCC, ninja-build, and the Xilinx ISE toolchain comprise the build system and development environment for the microcontroller firmware and FPGA bitstream. The above algorithm is implemented in C++ and runs on a Tiva-C microcontroller (TM4C123GH6PM). The FPGA design is synthesized and runs on a Spartan-6 FPGA (XC6SLX9). The FPGA is interfaced via GPIO to the analog frontend outputs, and presents a SPI slave interface with interrupt lines to the Tiva-C. The Tiva-C reads the timing information from the FPGA after each capture window, executes the solver, and outputs the device transform matrix over UART to the host computer for debug display.

The tracked device itself is composed of a hand-assembled perforation board which implements the analog frontend for 4 sensors, connected to the FPGA and Tiva-C boards. A 3D-printed mount for the 4th sensor brings it out-of-plane with the other 3 sensors (as described above).

The debug display script is written in Python.

F. Procedure

(k3) An oscilloscope and photodiode were used to initially characterize the lighthouse light field. A DMM was used to check the circuit layout.

(m4) Many revisions were made to the FPGA design, MCU firmware, and analog frontend layout over the course of this project. Engineering scientific strategies such as code revision bifurcation, hypothesis testing, and documentation were employed to improve development efficiency. The analog

fronted, for example, was reworked from a hard threshold set by a potentiometer to a diminished average threshold after I realized that the ambient light exposure on the photodiode would move the average reading below the fixed threshold if it had been calibrated for a different environment.

IV. Results

The outcome of this project is a functional, open-source tracked device implementation that relies solely on the structured light from one HTC Vive lighthouse, and tracks its position and orientation, accurate to $\pm 1mm$ at approximately $60Hz$.

A. Description of Results

The tracked device system is capable of tracking in the lighthouse light field under various ambient lighting conditions, and can track both orientation and position changes in real time. The accuracy of the tracking was measured by placing the tracked device at various distance intervals from the lighthouse and recording the outputted position. The results are given in Table 1.

Distance from lighthouse (cm)	Mean reported position, X (cm)	Reported position standard deviation, X (cm)	Delta-X (measured, cm)	Delta-X (reported, cm)
35.56	41.9407	0.0112	N/A	N/A
33.02	39.4675	0.0099	2.54	2.4732
30.48	36.9663	0.0118	2.54	2.5012
27.94	34.4319	0.0107	2.54	2.5344
25.40	31.9759	0.0078	2.54	2.4559

Table 1: Experimental data

B. Discussion of Results

As can be seen from the data above, the reported delta-X are within $\pm 1mm$ of the target $2.54mm$ (the device was moved $1in$ in between measurements). The uncertainty can be estimated from the standard deviation, and is well within $1mm$. The system presented here is entirely open-source, with all code and designs available on GitHub. The system, as-implemented, is also less than $100 USD in parts, and can be implemented for less if individual parts are used over development boards, and if the Spartan-6 FPGA is replaced by a Lattice Semiconductor iCE40 or other similar low-cost FPGA.

References

[1] Brown, Matt. "Exploring the Magic behind the HTC Vive Controller." *VRHeads*. Mobile Nations LLC, 28 July 2016. Web. 29 Mar. 2017.

[2] "Constellation." *Constellation - Virtual Reality and Augmented Reality Wiki - VR & AR Wiki*. XinReality, 17 Dec. 2016. Web. 29 Mar. 2017. <https://xinreality.com/wiki/Constellation>.

[3] Lang, Ben. "CES 2014: Oculus Rift Interview with Palmer Luckey and Nate Mitchell." *Road to VR*. Road to VR, 11 Jan. 2014. Web. 29 Mar. 2017.

[4] "System Requirements for VR." *NVIDIA GeForce*. NVIDIA Corporation, 2017. Web. 29 Mar. 2017. <http://www.geforce.com/hardware/technology/vr/system-requirements>.

[5] " Virtual Reality with AMD LiquidVR™ Technology." *AMD*. AMD, 2017. Web. 29 Mar. 2017. <http://www.amd.com/en-us/innovations/software-technologies/technologies-gaming/vr?utm_campaign=www.amd.com_liquidvr&utm_medium=redirect&utm_source=301>.

[6] Walton, Jarred. "We Test 15 Graphics Cards to Find the Best One for VR." *Pcgamer*. PC Gamer, 29 Mar. 2016. Web. 29 Mar. 2017.

[7] "Welcome to Steamworks." *Welcome to Steamworks*. Valve Corporation, 2017. Web. 29 Mar. 2017. <https://partner.steamgames.com/vrtracking/>.