Real-Time Pose Reconstruction from an Infrared Multicamera System

Bridger Herman, Nikki Kyllonen, and Liam Tyler

University of Minnesota

{herma582, kyllo089, tyler147}@umn.edu

May 4th, 2018

Abstract — Virtual Reality environments such as the CAVE necessitate real-time accurate tracking to be immersive. Optical tracking using passive markers and infrared cameras presents a non-intrusive way to accurately track the pose of the user. We present a basic real-time stereo tracking system using infrared cameras and passive retro-reflective markers that can be easily extended to full CAVE environments. Our results show that for an object with three markers attached, the system can reconstruct the object's position within 2.7cm and track a limited range of rotations within a few degrees.

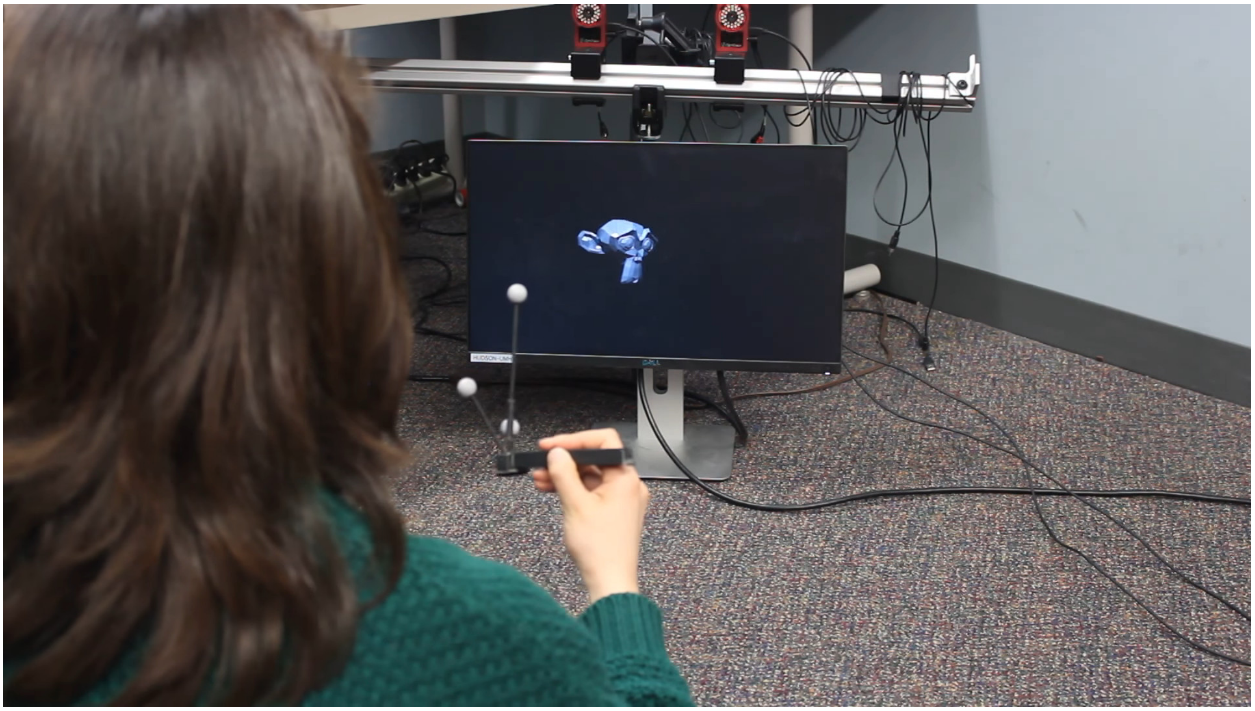


Figure 1: The real-time pose reconstruction system in action

1 Introduction

1.1 CAVE Overview

A CAVE, or a CAVE Automatic Virtual Environment, is described by Carolina Cruz-Neira as a "virtual reality/scientific visualization system" born out of the SIGGRAPH '92 Showcase effort [1]. A CAVE is a cubical room with the virtual world projected onto its sides, allowing the user to look and to walk in any direction. A generic four-sided CAVE is shown in Figure 2. The user's view into the virtual world updates to match the perspective of the user's pose (position and orientation) as they move. Commonly, CAVEs are set up with four or six screens. Although CAVEs are highly immersive, they are one of the least used VR environments due to their cost [2].

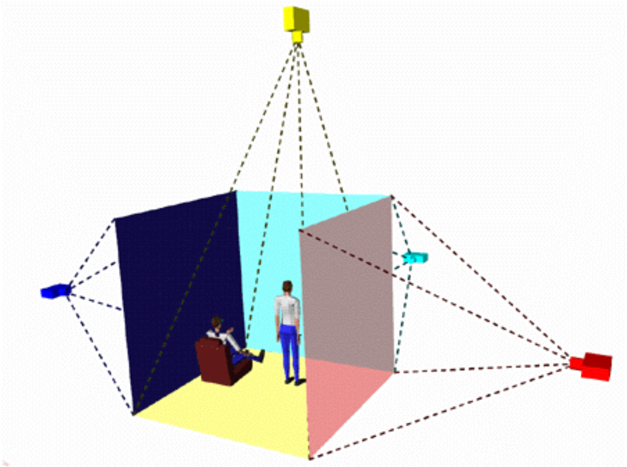


Figure 2: A generic four-sided CAVE setup [3]

A CAVE's immersion largely comes from its tracking system. Tracking is defined as the process of determining world coordinates of an object in real-time [4]. In order for the system to update the virtual world view to match the user's perspective, the user's world position and orientation have to be known. The glasses the user wears are tracked, which allows the correct viewpoint to be rendered. Users also usually have tracked tools to allow for more interaction with the virtual world, thus increasing the immersion. For this immersion to be successful, it is critical that the tracking is accurate and fast. Users become disoriented when the tracking is inaccurate or when the tracking introduces jitter. Similarly, latency has a large effect on human perception [5], and if the latency is too high then the view will not update quickly enough to match the user's current perspective. If disorientation or lag occurs, then the immersion is broken. As a result, accurate, lag-free, real-time (30+ frames per second [4]) tracking systems are critical in a CAVE.

1.2 System Setup

In order to track the glasses and tools, the CAVE is set up with several infrared cameras connected in sync. Camera sync is necessary to ensure that frames are taken at the same time, to ensure optimal point correspondence [6]. Each camera lens is surrounded by infrared LEDs that cast their light onto objects and users within the bounds of the screens. This setup allows for spherical retro-reflective markers to be attached to the tools and glasses. These markers reflect the infrared light cast by the cameras back to the cameras themselves.

The system used for this project has a similar setup to a CAVE, but with only two cameras, set up in a parallel stereo format. A diagram of the system setup is shown in Figure 3, and a photo of the system in action can be seen in Figure 1. A passive infrared tracking system was chosen for its compactness while still maintaining the necessary amount of accuracy. Other tracking methods include magnetic and mechanical systems which provide potentially even better accuracy, but at the expense of convenience [7]. Having an untethered, passive system allows for better immersion in a VR environment.

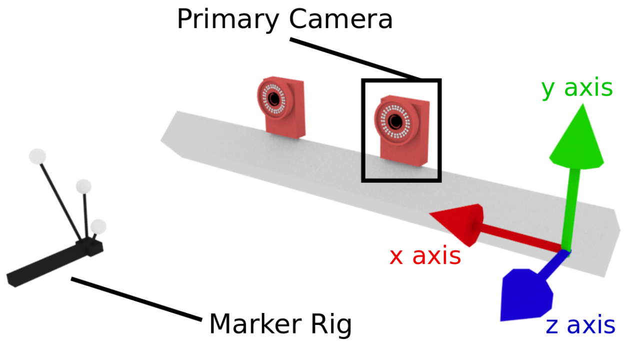


Figure 3: An image of our system setup

1.3 Method Description

The central focus of this paper is the method behind using an infrared system to accurately reconstruct the real-world poses of tracked items. The method we propose is as follows:

0. (System setup) Calibrate the cameras to get their intrinsic and extrinsic properties.
1. At each timestep, obtain the 2D image coordinates of all markers present, for all cameras in the system.
2. Triangulate the 3D positions of each marker from their 2D image coordinates.
3. Find the object's world rotation using the triangulated positions.
4. Update graphics to match object's world position

This method provides an accurate reconstruction of the tracked object's pose, which leads to an immersive experience in a virtual environment.

2 Related Work

Virtual reality and virtual environments have been around for several decades, and they continue to improve in both believability and cost-effectiveness every year. Alongside the development of VR have been improvements upon the tracking technologies used for VR systems. Real-time tracking is an inherently hard problem, because there are many things that can go wrong and many possible sources of noise. 3D tracking devices for immersive VR incur a higher noise-to-signal ratio than their non-immersive counterparts [5]. There are several tracking technologies available for use in VR environments. While many of the tracking systems provide advantages in certain aspects, there is not yet a "silver bullet" system that is the best of all worlds [8]. The types of tracking available to people wanting to set up a VR system are (listed in no particular order): magnetic, inertial, acoustic, optical, and mechanical [9].

The top requirements for a tracking system are maximizing accuracy, minimizing jitter, robustness to small changes, and mobility [4], all while maintaining real-time frame rates of 35Hz or better [10]. Additionally, it is very difficult to build a system that is able to convincingly fool human senses that have been trained since birth to perceive the real world [8]. Inertial tracking using accelerometers is often too imprecise for use in virtual environments; it relies only on the motion of the user and fails when the user is static [10]. Acoustic tracking has a limited update rate (only 10 to 50 samples per second) [11], which generally makes it unusable as a real-time tracking system for VR applications. Magnetic tracking is precise, but it is susceptible to many forms of distortion [12], and requires the user to be tethered, restricting their mobility. Wireless magnetic tracking exists, but it is not cost effective [7]. Mechanical tracking is extremely precise and low-latency, but makes the user nearly immobile or places them in an uncomfortable apparatus [9].

Optical tracking presents the least obtrusive option from a mobility perspective. Most mobile systems are so-called “outside-in,” which means that the cameras are fixed around the user [4]. The other option is “inside-out” tracking, where the user has cameras fixed to them that look outwards at the environment. These can be bulkier, and much more power-hungry [11]. Optical systems can sometimes introduce more jitter than magnetic systems [7], but in general, optical systems are good enough. Some optical tracking systems use direct feature tracking, but most use *marker-based* tracking because it requires much less processing, and gives much more precise triangulated positions [7]. Most optical tracking solutions are infrared-based, so they are less noticeable to people using the system. Other lighting schemes (visible, and near-infrared) have been used for tracking, but they can visibly interfere with the virtual environment [6].

Several systems ([13], [10]) have used outside-in *active marker* tracking, where the user wears infrared LEDs, and the cameras track the positions of those. However, these systems require the user to carry a battery pack, or to be tethered (both of which can reduce the user’s range of motion). The last option is a *passive marker*, outside-in infrared camera system, which uses retro-reflective spheres as tracking points. This configuration has been used successfully in multiple previous works ([14], [4]), because of its lightweight nature, freedom of motion, and high accuracy at a low price [10]. This is the method of tracking that was selected for the system that we present in this paper.

3 Camera Calibration

In order to reconstruct a point from its pixel coordinates, it is necessary to understand how points are projected onto the image plane. To capture this information, the cameras need to be fully calibrated [4]. The cameras are modeled as simple pinhole cameras, since lens and radial distortion are assumed to be negligible. The extrinsic parameters of the camera and their frames of reference relative to the camera are described in Figure 4 as the Camera Coordinate System (CCS). Intrinsic parameters describe the projective transformation of a point in camera space onto the image plane [6].

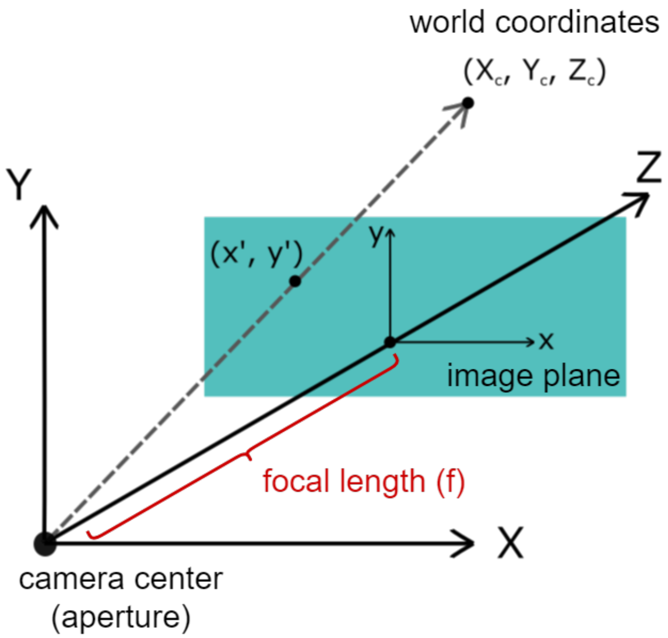


Figure 4: Relationship between the CCS, image coordinates, and a point in the world relative to the CCS

3.1 Camera Parameters

A camera’s intrinsic parameters are contained within a single matrix called the camera calibration matrix, denoted by **K** [6]. This matrix can be seen in Equation (1).

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \gamma & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

The parameters $\alpha_x = \frac{f}{dx}$ and $\alpha_y = \frac{f}{dy}$ describe the focal length in pixels [15]. dx and dy describe the physical dimensions of the camera's pixels. γ is called the skew coefficient and is often set to 0. The values in column three represent the principal point of the image plane in pixel coordinates. Ideally, the principal point (x_0, y_0) would be in the center of the image plane.

K is the projection matrix for transforming 3D points in camera space (displayed within Figure 4 as X_c, Y_c, and Z_c) to 2D image coordinates. This transform can be written as the following

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{2}$$

with $x = \frac{x'}{dx} + x_0$ and $y = \frac{y'}{dy} + y_0$ being the physical position of the pixel on the image plane [4].

A camera's extrinsic parameters describe the rigid rotations and translations necessary to transform the World Coordinate System (WCS) to align with the CCS. **R** is the rotation matrix to rotate the world coordinates into camera space, and $\vec{t}$ is the translation vector to shift the world origin to the camera's position [6]. This relationship is shown in Equation (3).

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \vec{t} \tag{3}$$

with the absolute world coordinates being represented as the column vector containing X, Y, and Z [4].

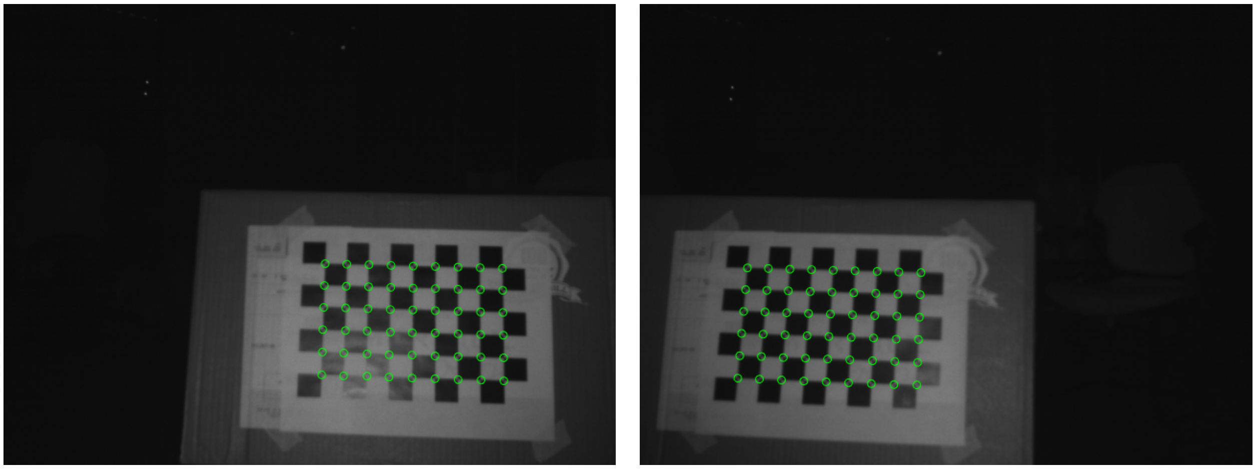
Combining Equations (2) and (3), the camera matrix **P** can be derived as the 3x4 matrix that contains **K**, **R**, and $\vec{t}$ such that $\mathbf{P} = \mathbf{K}\ [\mathbf{R}\ \ \vec{t}]$. The camera matrix **P** allows us to move between image coordinates $\vec{x}$ and world coordinates $\vec{X}$ through the simple proportion $\vec{x} \propto \mathbf{P}\vec{X}$, where each set of coordinates is represented as a column vector containing the homogeneous representation of their positions.

3.2 Single- and Stereo-Camera Calibration

Single-camera calibration is the series of steps necessary to determine a camera's **P** matrix. There are two main ways to do this: resectioning and plane-based calibration. Resectioning is the process of estimating **P** from known $\vec{x}$ and $\vec{X}$ values. Resectioning starts with an image on which feature points have been identified. These feature points and their corresponding 3D positions are then plugged into the proportion relating the two quantities: $\vec{x} \propto \mathbf{P}\vec{X}$. However, this approach is most often impractical since the 3D coordinates of objects are usually unknown.

As a result, plane-based calibration is the preferred approach to calibrating a single camera. Plane-based calibration determines a camera's intrinsic parameters by using multiple pictures of a planar surface [6], or with the camera stationary and the plane positioned differently. To provide strong corners as well as a uniform grid upon which to easily place feature points, a checkerboard pattern of uniform squares is generally used. See Figure 5 for a sample stereo pair of calibration images taken during the course of this project, with an overlay of the detected checkerboard pattern.

In a stereo setup, there is a fundamental matrix **F** defined by the relationship $\vec{x}^T\mathbf{F}\vec{y} = 0$ of the image



Left image

Right image

Figure 5: Stereo pair of camera calibration images

coordinates for one camera, $\vec{x}$, to the image coordinates of the second camera, $\vec{y}$. **F** contains the intrinsic parameters of each of the cameras through the definition

$$\mathbf{F} = \mathbf{K}^{-T}\ \mathbf{E}\ \mathbf{K'}^{-1} \tag{4}$$

where **K** and **K′** are the camera calibration matrices for each of the cameras and **E** is the essential matrix [4]. The essential matrix is determined by the transform between the cameras and contains information on the epipolar geometry of the camera setup. Using a Plane-Based Calibration technique, each camera's intrinsic parameters can be determined. Then, once the essential matrix **E** has been calculated, the fundamental matrix can be found.

3.3 Our Calibration

To determine **K**, **R**, and $\vec{t}$ for each of the cameras as well as the fundamental matrix **F**, we used the MATLAB Stereo Camera Calibration App. This app follows the calibration process describe by Zhang [16]. It does a non-linear optimization method on checkerboard images. Each pair of images must be of the same checkerboard, taken by each of the cameras, as shown in Figure 5. For our calibration, 26 pairs of images were taken and passed into the MATLAB app. The app then chooses which images are suitable before calculating parameters, and does the optimization to solve for all of the parameters.

4 Region Detection

Getting the coordinates of the retro-reflective markers in the infrared image pairs is necessary for obtaining the location in world space of the markers. The markers show up on the image as bright circles which can be easily thresholded, minimizing the amount of processing for each frame (since marker tracking is agnostic about edges and other complex features [6]).

Once the image is thresholded, it is passed to the `floodFill` function shown in Listing 1. After finding the region areas, all the important region properties (notably the region radius and its centroid) are returned for use in getting the world position. See Figure 6 for a flowchart of how this process works with a stereo-camera setup. In Listing 1, once a region is found, it is expanded using its eight-connected neighbors, which can be seen in Figure 7. The steps for a simple flood-fill approach to region finding are as follows:

1. Row-major scan until we hit a value greater than zero. Add to current region.
2. Add neighbors of pixel to queue.
3. If neighbors' values greater than zero, add them to current region.
4. Repeat until no more neighbors to expand.
5. Continue with row-major scan.

Listing 1: Pseudocode for finding regions in an image

```
function floodFill(image):
  visited = {}
  regions = {}
  queue = {}
  for pixel in image:
    if image[pixel] > 0 and pixel not in visited:
      visited.add(pixel)
      for neighbor in neighbors(pixel):
        if neighbor not in visited:
          queue.push(neighbor)
      currentRegion = {pixel}
    while queue not empty:
        top = queue.pop()
        currentRegion.add(top)
        visited.add(top)
        for neighbor in neighbors(top):
          if image[neighbor] > 0
              and neighbor not in visited
              and neighbor not in q:
            q.push(neighbor)
          regions.add(currentRegion)
  return regions
```

While a faster algorithm could have been used, this simple flood-fill works in real-time on the hardware used in this project. It should also be noted that unfortunately, due to the limitations of the infrared cameras used for the project, this algorithm was not used in the final product. The OptiTrack cameras have a 640x480 resolution, at 8 bits per pixel. The cameras use a USB 2.0 interface, which is not sufficient to transfer the ≈600 Mb/s necessary to accommodate real-time grayscale streaming for a stereo setup [6]. Instead, the camera SDK streams already segmented images and their centroids.

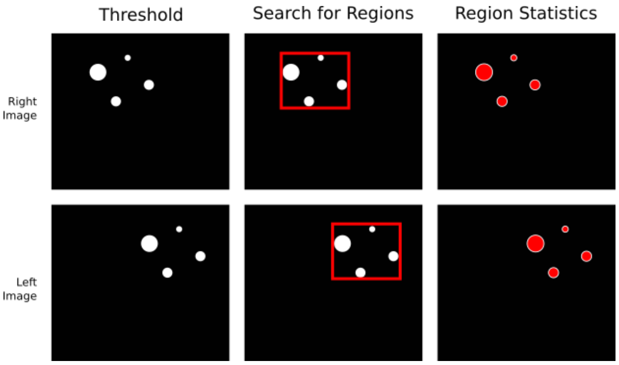


Figure 6: Flowchart for how regions are obtained

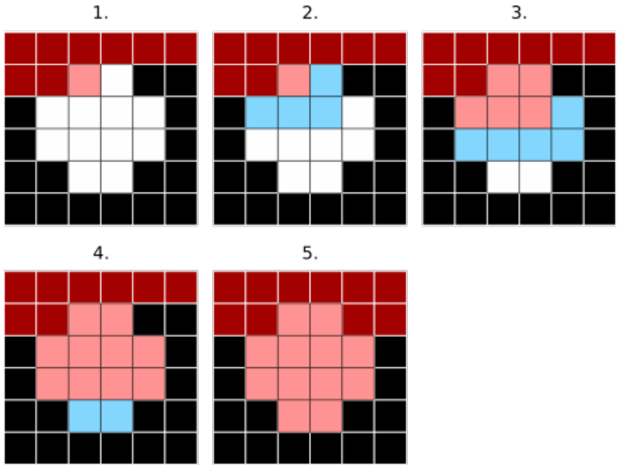


Figure 7: Region finding using eight-connected neighbors

5 3D Reconstruction

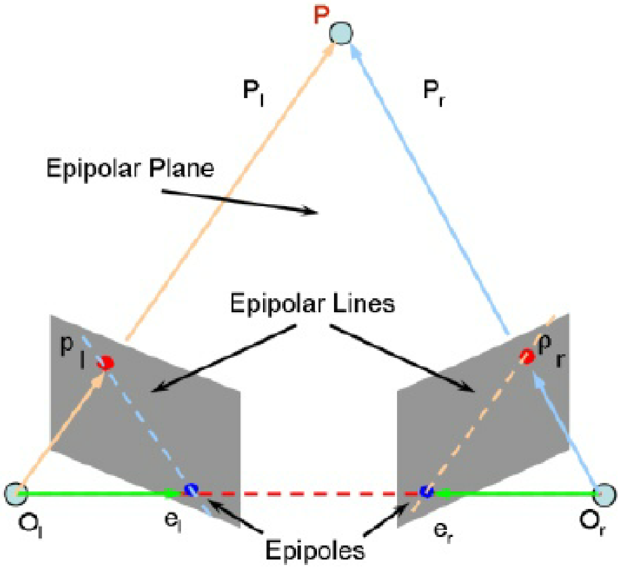


Figure 8: A drawing of two cameras, their image planes, and the epipolar geometry [17]

5.1 Point Correspondence

In order to reconstruct the 3D world point from two images, the point correspondences must be known. That is, which markers from image one correspond to the same markers in image two. To find what the corresponding points are, we use the epipolar constraint [4]. This constraint is seen in Equation (5) below.

$$\vec{p_1}^{\top} * \mathbf{F} * \vec{p_2} = 0 \tag{5}$$

$\vec{p_1}$ is a point in image one (the reference image), $\vec{p_2}$ is the corresponding point in image two, and **F** is the fundamental matrix described in Equation (4). If two points correspond, then this equation will be valid. This constraint also describes a line that, given a point in image one, its corresponding point fall on. This line is called an epipolar line, as seen in Figure 8. This simplifies the search for a corresponding point by narrowing it down to a point as opposed to the entire image. While epipolar geometry helps greatly with finding the corresponding points, there are two remaining issues.

The first issue is that the epipolar lines are not always horizontal, as in Figure 8. This makes searching along the lines more difficult. What is usually done to simplify this is called image rectification, which is the process of warping the image such that the epipolar lines become horizontal. Thus, the line scan becomes a one dimensional search, as opposed to of two dimensional. In the setup used for this project, the cameras are coplanar, so the epipolar lines are horizontal to begin with. As a result, image rectification was not necessary.

The second issue is more problematic: it is possible to rotate and position a set of markers so that multiple points fall onto the same epipolar line. For all of these points, Equation (5) will hold true and false matches will occur [10]. Matching points is further problematic by the fact that the order of the points can also differ from the left to right images. One common strategy used to mitigate this is to project all of the points onto a third camera plane, where the points will not fall onto the same epipolar line [10]. Since there are only two cameras, this is not an option. In our setup, we ignore this problem since it only occurs occasionally and would not be an issue when extending the system with more cameras.

5.2 Triangulation

Once the point correspondences are known, we can reconstruct the world coordinates of the centroid of the points. To do this, we need to use the projection matrices found in section 3.1 that describe how a world point is projected onto the image plane. This image plane will be the same for both cameras. We construct these matrices using the following equations [9]

$$\begin{aligned} \mathbf{P_1} &= \mathbf{K_1}[\mathbf{I_3} \mid 0] \\ \mathbf{P_2} &= \mathbf{K_2}[\mathbf{R_2} \mid \vec{t_2}] \end{aligned} \tag{6}$$

$\mathbf{P_1}$ and $\mathbf{P_2}$ are the projection matrices for the two cameras, with Camera 1 being the primary (reference) camera. $\mathbf{K_1}$ and $\mathbf{K_2}$ are the intrinsic matrices from the camera calibration, as described in Equations (1) and (2). Camera 1 has no rotation or translation since it is the reference camera and Camera 2 has rotation and translation described by $\mathbf{R_2}$ and $\vec{t_2}$, respectively. These transformations are in relation to Camera 1. Given these projection matrices, a world point projected onto the image plane and the actual image coordinates should be parallel, as seen in Equation (7).

$$\lambda \begin{bmatrix} \vec{u} \\ 1 \end{bmatrix} = \mathbf{P_1} \begin{bmatrix} \vec{X} \\ 1 \end{bmatrix} \tag{7}$$

$\vec{u}$ is the 2D image coordinate, $\mathbf{P}$ is the projection matrix, and $\vec{X}$ is the 3D world coordinate. Since these are parallel, their cross products are 0, and the following system can be used to solve for the world coordinate:

$$\begin{bmatrix} \begin{bmatrix} \vec{u} \\ 1 \end{bmatrix}_\times \mathbf{P_1} \\ \begin{bmatrix} \vec{v} \\ 1 \end{bmatrix}_\times \mathbf{P_2} \end{bmatrix} \begin{bmatrix} \vec{X} \\ 1 \end{bmatrix} = 0 \tag{8}$$

The subscript $_\times$ denotes the skew-symmetric matrix of the vector. We solve this system using a linear least squares triangulation method [6]. This gives the world coordinate $\vec{X}$.

5.3 Pose Reconstruction

Once the world coordinates of the tracking points are obtained from the previous step, we can find the pose of the object in world space. The first step is to find the centroid in world coordinates of the tracked points, which directly becomes the position of the virtual object. Slightly more complicated, however, is reconstructing the virtual object's orientation. Two vectors are computed between points $\vec{p_1}$ and $\vec{p_2}$, and points $\vec{p_1}$ and $\vec{p_3}$. The cross product between these two vectors is found (the normal of the plane between the three points), and is used to update the orientation of the virtual object. See Figure 9 for a depiction of pose reconstruction.

6 Implementation

The system setup is described in Figure 3. There are two parallel OptiTrack V100 infrared cameras, and we use a rig of three retro-reflective markers for the cameras to capture. The OptiTrack C++ SDK is used to obtain the thresholded images and centroids of the retro-reflective markers in real-time at about 60 frames per second. At each frame, the following steps occur. First, we find which pairs of stereo points correspond, using Equation 5. We then reconstruct the world positions of the markers. From there, we calculate the world pose of the virtual object, and display its 3D model on the screen.

6.1 Point Correspondence

Our system had three markers in which we had to find their corresponding points. According the equation (5), we should see that the corresponding points equal zero. In reality though, they don't perfectly equal zero due to calibration and inaccuracies. As a result, the actual way we found for corresponding points is described in the pseudocode in Listing :

Listing 2: Pseudocode for finding the corresponding points

```
for point1 in image 1:
  epipolar_constraints = []
  for point2 in image 2:
    epipolar_constraints.append(abs(transpose(point1) * F * point2))
  matched_point_value = min(epipolar_constraints)
  point1.matched_point = point_with_value(matched_point_value)
```

By taking the absolute value of the regular epipolar constraint equation (Equation (5)), we then take the point which most closely approximates the zero vector as the reference point's corresponding point. This still encounters the problems of point correspondence described in section 5.1, and is compounded with the errors that come along with working in the real world. For the purposes of our system however, the point correspondence method above was found to produce convincing enough results, without too many noticeable errors.

6.2 Triangulation and Pose Reconstruction

The Eigen C++ library was used to triangulate the corresponding points from the previous step, and was also helpful in reconstructing the object's world pose. The 6x4 matrix from Equation (8) was constructed from the skew-symmetric versions of pairs of corresponding points ($\vec{u}, \vec{v}$), and the camera projection matrices $\mathbf{P_1}$ and $\mathbf{P_2}$ from Equation (6). Eigen's `JacobiSVD` singular value decomposition was then used to find an orthonormal basis for the nullspace of the 6x4 matrix $\mathbf{V}^\intercal$, as shown in Equation (9). The last column of the 4x4 matrix $\mathbf{V}^\intercal$ represents the homogeneous world coordinates (4x1 vector) of the marker point. To obtain the final world coordinates, all that needs to be done is divide the entire vector by its last component.

$$\mathbf{A}_{6x4} = \mathbf{U}_{6x6}\mathbf{\Sigma}_{4x4}\mathbf{V}^{\intercal}_{4x4} \tag{9}$$

Once world positions for each point are gathered, the centroid is computed by taking the average of the point world positions. The orientation of the model is calculated from the normal of the plane spanned by the point world positions, as stated previously. Before the model's pose is updated, however, some additional processing needs to occur. First, it is determined if the world position vector is finite or not. Sometimes, during the first few frames (and sporadically throughout the rest of the program), the stereo

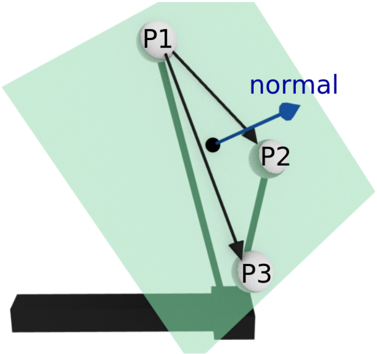


Figure 9: Diagram of marker rig, including the centroid of the world points, the plane between the points, and its normal

pixel coordinate pairs will cause the world coordinates to become infinite, which makes rendering the object impossible. Additionally, when we first got the tracking working, there was a noticeable jitter in the virtual model's position, even when the marker rig was held steady by a tripod. To mitigate this issue, a rolling average of 30 timesteps worth of positions and rotations are taken to be the current position and rotation, which leads the tracking to have a much smoother, continuous feel to it. This can be adjusted for the desired balance between latency and smoothness. Once the final world pose of the virtual object is calculated, it is sent to the graphics engine for rendering.

6.3 Graphics

While the goal of this system is to be generalizable for a CAVE, in order to show the results of our tracking, we rendered a 3D model with the position and pose that we calculate. For rendering the model in real time, we used a small C++ library built on top of OpenGL and SDL2 by Liam Tyler. It deals with the windowing, OBJ model loading, and all of the rendering. All we had to do is update the model's position and rotation matrix every frame with whatever values were calculated from the triangulation and pose reconstruction. This library was chosen for two reasons. The first was for speed, since our goal was to have the system be real-time, and rendering with C++ and OpenGL can be very fast. The second reason was simply ease of use, since we were already familiar with the system, and the camera SDK was forcing us to use C++ already. The library could easily be substituted for many other rendering libraries if desired.

7 Results

In order to test the accuracy of the system, the calculated position of the object point was compared against its real-world position. This comparison was done several times with real-world coordinate values which were known by measuring their X, Y, and Z distances from the primary camera. The rotational bounds of the system was determined by measuring at what angle about the X, Y, and Z axes the tracking began failing to accurately replicate the rotation of the model within a few degrees.

7.1 Translation

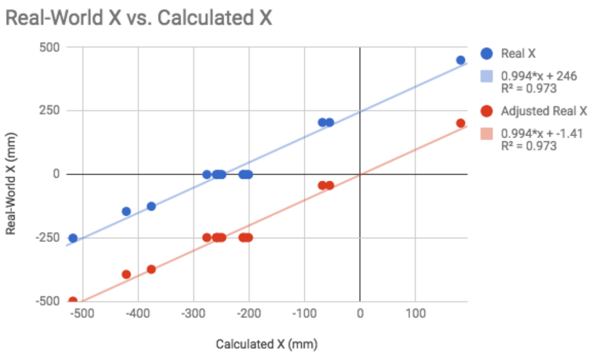


Figure 10: Plot comparing the Real-World X coordinates against the Calculated X coordinates

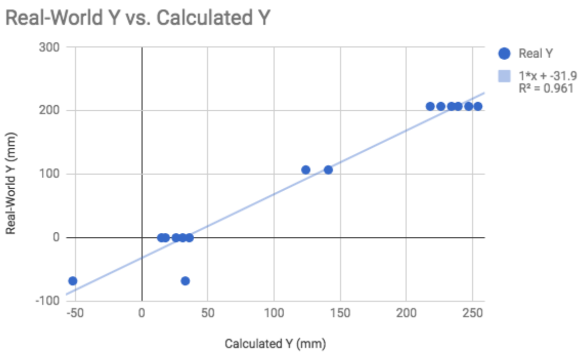


Figure 11: Plot comparing the Real-World Y coordinates against the Calculated Y coordinates

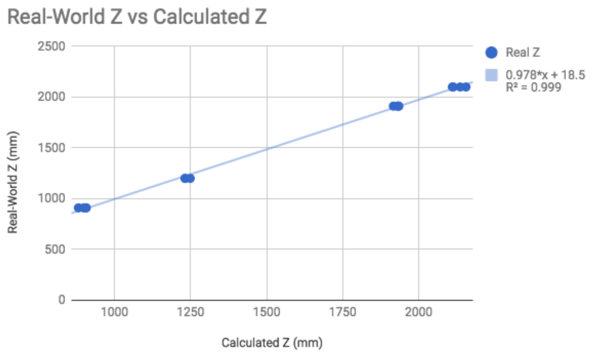


Figure 12: Plot comparing the Real-World Z coordinates against the Calculated Z coordinates

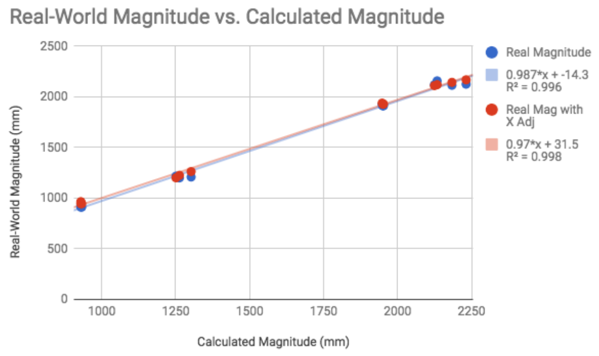


Figure 13: Plot comparing the magnitudes of the vector from the camera to the detected point calculated using Real-World coordinates vs. using Calculated coordinates

By sampling world positions within the effective bounding area, we were able to compare the actual positions against those that the system calculated. For Y and Z, as shown in Figures 11 and 12, the results were fairly accurate. The calculated line of best fit for Y was determined to have the ideal slope of 1, although outliers are clearly present, as shown by the worst $\mathbf{R}^2$ value of 0.961. The best fit for Z had a less accurate slope of 0.978, but showed a smaller variance from the fit line with an $\mathbf{R}^2$ value of 0.999. For X, however, the system seemed to have a consistent offset. By calculating the average offset from the measured values, 247.8mm, we then adjusted our Real-World values by this amount, producing the red data points and red fit line displayed in Figure 10. Both adjusted and unadjusted X data showed slopes of 0.994 with $\mathbf{R}^2$ values of 0.973, which is a near one-to-one correlation. If you take the adjusted X values, the average difference in the magnitude of the calculated position and the actual position was 2.7cm, with a standard deviation of 1.5cm.

Since the translational bounds of the effective area is purely hardware dependent, the bounds were not calculated. Field of view of the cameras used in the system determines these bounds, therefore, using cameras with wider or narrower fields of view would have different translational bounds.

7.2 Rotation

Axis	Max Angle	Min Angle
x	41	-81
y	90	65
z	0	0

Table 1: Table holding the measured rotational bounds of the system

The rotational bounds of the system are limited. These were measured by starting the marker rig pointing along the optical axis, with a depth of one meter, height equal to the cameras, and centered between both cameras. We then would start the program, and record the furthest rotation about an axis that could be made before the reconstruction became inaccurate or the system failed to show any change in rotation. The maximum angles were determined by rotating in the positive direction around a specific axis while the minimum angles were determined by rotating in the negative direction. The positive direction around an axis was defined using the right-hand rule with your thumb aligned with the positive direction of the axis. The angle measurements calculated by the system were within a few degrees of the actual measured values.

8 Discussion

8.1 Comparison to Previous Work

To understand the performance of our system, we examine other works. Kato and Billinghurst showed that they could reconstruct the marker positions in real time within about 10mm on average [18]. They also could detect the slant of their marker within usually a few degrees. Forsa was able to reconstruct the position within 5mm, and orientation of the head within three degrees [9]. The speed of that system was about 25 frames per second. Neumann et al. created an inertial vision hybrid system, which ran at 9 frames per second, and had positional drift after prolonged system use [19]. Compared to these three systems, our system ran at 60 frames per second, with an average position difference was 27mm, and the orientation was within a few degrees, given that the object's orientation was within the ranges listed in section 7.2.

With position reconstruction, our system performs the worst. All three systems had higher accuracy, although the hybrid inertial vision system would eventually have worse reconstruction with its position drifting limitation. Normally the accuracy of our system is not considered to be within an acceptable error [4]. For the scope of this project, we did not have enough time to increase the accuracy. It is also important to note that while the exact position isn't reconstructed very well, it is consistent. So, if a user walked across a CAVE, the tracking system would still have nearly the same motion, just maybe not the exact length of the path walked. This means that the system is still usable, as seen in the demo video. It just loses some of the immersiveness if it were to be used in a CAVE system. Our system also had the worst orientation reconstruction. This is simply due to the limited range of rotations allowed, and the lack of tracking rotations about the Z axis at all. For the allowed rotations though, our system performed nearly as well as the other ones. The rotational and positional reconstruction issues are further discussed in the limitations below.

Comparing latency, our system performs the best out of the papers surveyed. Kato and Billinghurst [18] had real time reconstruction, but neither 9, nor 25 frames per second are acceptable for CAVE virtual environments. It is worth mentioning however, that at the time these other systems were created, the hardware was much slower than it is today. Their systems could potentially be real-time now. While our system may not be the best at everything, the method has has some merit to it. It performs well in regards to latency, and allows the user to be mobile. The method used in this paper is also easily generalizable to more cameras than dual-camera stereo [6].

8.2 Limitations

One of the main limitations in our system is the lack of rotation about the Z-axis, and the limited range of rotations about the X and Y axes. The way we calculate the object's rotation uses the normal of the plane spanned by the markers as the desired direction the model. However, if this is the only analysis done, then it is impossible to tell how much the model is rotated about the Z-axis. This is because if the the plane's normal is facing along the Z axis (optical axis), then any rotation about the Z axis still has the same triangle normal, and is ambiguous. In our implementation, this means that the rotation matrix will not change at all.

The other issue with rotations is that the rotations about the X and Y axes are limited as seen in section 6.2. This is due to a lack of coverage with our stereo setup, which is a common problem in the literature [7]. There are many rotations where one marker occludes the other, the rig occludes a marker, or the hand holding the rig occludes a marker. Any time these one of these occurs, the our system fails to reconstruct the pose of the object, and does not update it. This would be largely solved if we added more cameras in positions with good coverage. This would decrease the chance of there being less than two occluded views, and reconstruction of pose could occur as normal.

The reconstruction the object's position is usually within 2.7cm of the true position on average. For our purposes, this was an acceptable accuracy, but if a user needs to know positions with a high degree of precision, then the current system would not be usable. A couple methods could be used to help fix this. More cameras could be used, so each point can be the average of multiple reconstructions from pairs of cameras. Another method would be a variant of bundle adjustment [20], but this would likely slow the system down.

9 Conclusion

This paper describes a basic real-time infrared tracking system that can be extended for a CAVE environment by adding more cameras to the setup. It describes the methods used to calibrate the system, obtain the marker image coordinates, and reconstruct the position and pose. Our results show that our two camera setup can reconstruct the position with acceptable accuracy, while the rotation is not reconstructed as well, and has a limited range of valid rotations.

In the future we would like to improve the system in three main areas. The first would be add more cameras to increase the volume of visibility and help with the correspondence aliasing. Second would be to do some sort of bundle adjustment so that our calculated world coordinates would be more accurate. Lastly, we would like to use a different method for calculating the pose, so that it could better handle rotations about the Z-axis.

10 Acknowledgements

The authors would like to thank Professor Daniel Keefe and the Interactive Visualization Lab for the use of their tracking equipment for this project.

11 Code and Demo

Code used in this project can be found in the GitHub Repository at `https://github.umn.edu/herma582/CVProject`.

The demonstration video can be found at `https://youtu.be/jnlOnVAUHtU`.

References

[1] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, “Surround-screen projection-based virtual reality: the design and implementation of the cave,” in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pp. 135–142, ACM, 1993.

[2] S. A. Miller, N. J. Misch, and A. J. Dalton, “Low-cost, portable, multi-wall virtual reality,” 2005.

[3] “Virtual reality laboratory website.” `http://umich.edu/~vrl/intro/`. University of Michigan.

[4] M. Ribo, A. Pinz, and A. L. Fuhrmann, “A new optical tracking system for virtual and augmented reality applications,” in *Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE*, vol. 3, pp. 1932–1936, IEEE, 2001.

[5] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and I. S. MacKenzie, “Effects of tracking technology, latency, and spatial jitter on object movement,” in *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pp. 43–50, IEEE, 2009.

[6] M. Mehling, “Implementation of a low cost marker based infrared light optical tracking system,” *Institute for Software Technology & Interactive Systems*, 2006.

[7] J. Chung, N. Kim, J. Kim, and C.-M. Park, “Postrack: A low cost real-time motion tracking system for vr application,” in *Virtual Systems and Multimedia, 2001. Proceedings. Seventh International Conference on*, pp. 383–392, IEEE, 2001.

[8] G. Welch and E. Foxlin, “Motion tracking: No silver bullet, but a respectable arsenal,” *IEEE Computer graphics and Applications*, vol. 22, no. 6, pp. 24–38, 2002.

[9] M. Foursa, "Real-time infrared tracking system for virtual environments," in *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pp. 427–430, ACM, 2004.

[10] W. Xu, B. Wang, and Y. Jiang, "Multi-target indoor tracking and recognition system with infrared markers for virtual reality," in *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 1549–1553, March 2017.

[11] D. Schmalstieg and T. Hollerer, *Augmented reality: principles and practice*. Addison-Wesley Professional, 2016.

[12] K. Dorfmüller and H. Wirth, "Real-time hand and head tracking for virtual environments using infrared beacons," in *Modelling and Motion Capture Techniques for Virtual Environments*, pp. 113–127, Springer, 1998.

[13] K. Dorfmüller-Ulhaas, *Optical tracking: from user motion to 3D interaction*. Citeseer, 2002.

[14] K. Dorfmüller, "An optical tracking system for vr/ar-applications," in *Virtual Environments' 99*, pp. 33–42, Springer, 1999.

[15] R. Radke, "Image formation and single-camera calibration," 2014. Rensselaer Polytechnic Institute.

[16] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, Nov 2000.

[17] I. Brilakis, H. Fathi, and A. Rashidi, "Progressive 3d reconstruction of infrastructure with videogrammetry," vol. 20, pp. 884–895, 11 2011.

[18] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on*, pp. 85–94, 1999.

[19] S. You, U. Neumann, and R. Azuma, "Hybrid inertial and vision tracking for augmented reality registration," in *Proceedings IEEE Virtual Reality (Cat. No. 99CB36316)*, pp. 260–267, Mar 1999.

[20] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment – a modern synthesis," in *International workshop on vision algorithms*, pp. 298–372, Springer, 1999.