VILLANOVA UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Interactions in the Virtual Realm

Jack Cieslik, Chris Marrs, Joseph Han, David Kaye

January 30, 2020

Abstract

This report contains insight into the Interactions in the Virtual Realm senior capstone project. The goal of this report is to describe each part of the group's final design project. This report will inform the reader of the requisite subsystems to catch a physical ball in virtual reality as well as each team member's role in designing each of the subsystems and code required to make this project work. Project management topics such as budget, schedule and necessary parts will be also included to provide a comprehensive outlook of this project.

Table of Contents

1. Introduction

1.1 Overview

Virtual reality is a type of simulated reality where a person can put on a headset and feel as if they are in a completely different environment. This idea of virtual reality has been around for decades but it was not until the late 20th/early 21st century that these ideas of a computer generated world became a reality. As early as the 1930's science fiction stories began to bring up the idea of simulated realities via sensory providing headsets. From the 90's to around 2014 there were many advancements in processing and computer technologies which would later lay the groundwork for VR technologies to be possible. In 2014, Facebook bought the oculus VR company, and after this, many other large tech companies such as Sony, Google, Samsung and more began working on their own headsets as they saw the emergence of the new and untapped market. It was at this point that VR technology really started to develop quickly as companies were able to make cheaper, more commercialized, and more immersive headsets. The future research and implementation of VR will be to integrate the physical world with the virtual. The idea of augmented reality is not new but has only recently started to gain traction in industry and consumer products. Companies such as google, apple, microsoft and many others are investing and developing augmented reality devices, but much more research needs to be done in order to perfect the designs.

1.2 Global, Societal, Cultural, and Economic Impacts

This project has many practical applications for the further development in entertainment, gaming, sports training and therapy. For gaming and entertainment, one of the main goals of future developments in the field is to make the experience more realistic. Our project relates to this because by adding the extra element of touch into our virtual reality experience, it can create a more immersive environment for the user which can later be applied for more commercial applications in the entertainment industry, as well as more realistic applications involving sports training and therapy. As an example, a training baseball player could put himself into the virtual realm where certain scenarios can be played, while also having a ball thrown to him to be caught. Moreover, our project could be used for physical rehabilitation for both athletes and patients with movement disabilities.

1.3 Ethical Issues related to Public Health, Safety and Welfare Impacts

The ethical issues related to projects involving virtual reality are primarily safety concerns. Some examples of these concerns are: chance of injury due to lack of awareness, developing disorders in the eye, motion sickness, hearing impairments, and distortion of reality (LaMotte 2017). Many of these concerns are similar to the risks often cited with video games and increased use of technology like mobile phones or television use. Such risks are amplified for children, who are more likely to experience the adverse effects of virtual reality. These safety concerns would need to be addressed with safety information packets that explain the dangers of virtual technology to users. Furthermore, the increased risk for children would need to be made very clear.

1.4 Regulatory Issues

Regulatory issues include standards enforced by the FCC. Similar to other electronics, safety information and health risks would be important to include with the product (HTC Corporation 2020). Because of the multitude of content available through virtual reality, regulations on content would also be included. For example, video games made available through virtual reality are regulated and rated for age-appropriateness by the ESRB.

1.5 Prior Work

The need for physical interaction in the virtual realm has been well documented and addressed by many hoping to improve the VR experience. Most notably, a team of Disney engineers created a system similar to this project utilizing virtual reality and some scheme of sensors to visualize and catch a thrown ball (Fingas 2019). Though unclear what the specific subsystems are, there does not appear to be any intellectual property regarding the objective of this project.

1.6 Organization of Report

Section 2 of this report will first introduce the specifications of the project, including functional and performance specifications of the design. Next, an overview of the design and accompanying block diagram will generally outline the subsystems used to accomplish our objective in Section 3. Each subsystem will then be explained in Section 4 to provide greater understanding of how the project achieved its goals and what technical limitations were encountered. For project management related matters, Section 5 will discuss each team member’s contribution, present an outline of the project timeline with a Gantt Chart, and list the necessary parts, skills, and budget used. Finally, a summary of achievements and conclusions will be presented in Section 6 and Section 7.

2. Specifications

2.1 Functional Specification

This system's primary functions include the following:

- Live tracking and representation of moving objects in a virtual environment
- Live tracking and representation of moving person in a virtual environment
- Enable user to reliably catch a physical object while in a virtual environment
- Display projected motion of the moving ball

2.2 Performance Specification

This system's performance specifications include the following:

- Functionality in a 3m by 4m rectangular area
- Projected ball path accurate to 6 inch radius
- Minimum of 1 second lag between actual and displayed motion
- Enable user to catch a ball moving over 10 mph
- Fully set up system in under 5 minutes

3. Design Overview

The design of this project was divided into three primary subsystems. The first is an Azure Kinect camera which is used the track the joint positional data of a user throwing a ball. This data was accessed using the Azure Kinect body tracking SDK and processed using custom code written in C++. The second subsystem is incorporating Colorado Engineering's iScan

Phantom radar unit to track and extrapolate the projected path of a moving ball. This system uses a 77 GHz antenna module and processes with code written in MATLAB to visualize an object's movement. The third subsystem consists of several HTC Vive components to create a virtual environment where everything is appropriately placed in relation to the users. This includes multiple Vive trackers, two base stations, the headset itself powered by a Dell tower. Integration of all the data from the subsystems (motion tracking of user and ball data) was done with Python is Vizard to display all components in virtual reality.

Integration of the subsystems introduced above is outlined in the block diagram shown in Fig. 1. After the user on the left throws the ball, the CEI radar unit will collect data from its initial motion and compute a projected parabola corresponding to the ball's motion. The Azure Kinect camera will record and constantly update positional data of the body's 26 joints. The ball and body positional data is then imported into Vizard every time it updates and displayed through the user's virtual reality headset. This data is visualized in the headset relative to the Vive tracker on the user's hand.

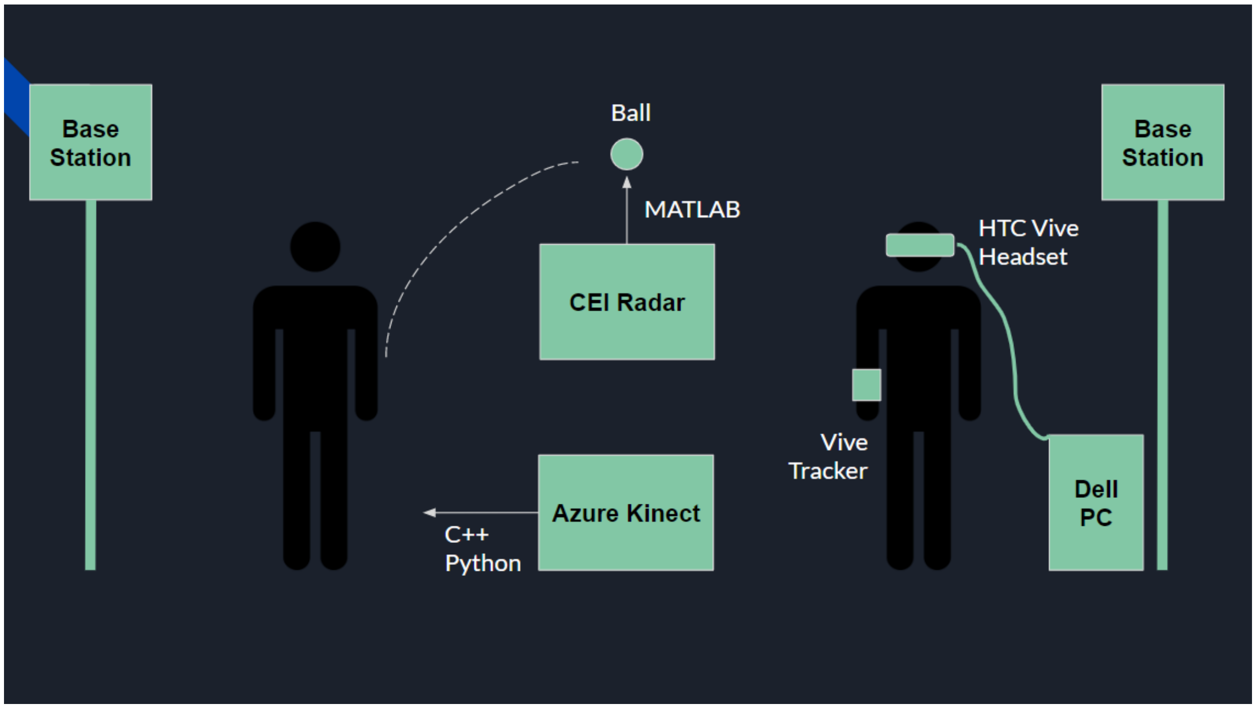


Figure 1: Full system block diagram with subsystems and corresponding code language

4. Subsystem Designs

4.1 Subsystem I: Motion capture with IR sensing camera

The Azure Kinect camera was chosen because of its IR camera and the structure and reliability of Microsofts' developer tools. IR depth sensing works by emitting an array of IR dots, then analyzes the distorted pattern as it is reflected off surfaces at various depths. Thus, the Azure Kinect will be placed on a tripod approximately 2 meters away with its field of view (57º horizontally and 43º vertically) directed down towards the user.

The Kinect body tracking SDK was used to collect x, y, z and orientation data (quaternions) for each of the 26 defined joints for each frame of video. Since this unit is quite new, the SDK had minimal functions for use and did not include data streaming. In C++, a custom function was written to constantly write and update the aforementioned data to a text file on the hard drive.

Initially, the previous Kinect camera version (Kinect 360) had been chosen as it had much more documentation and project examples, but the Azure Kinect was chosen because of its ability to discern orientation as well as position.

Figure 2: Azure Kinect camera

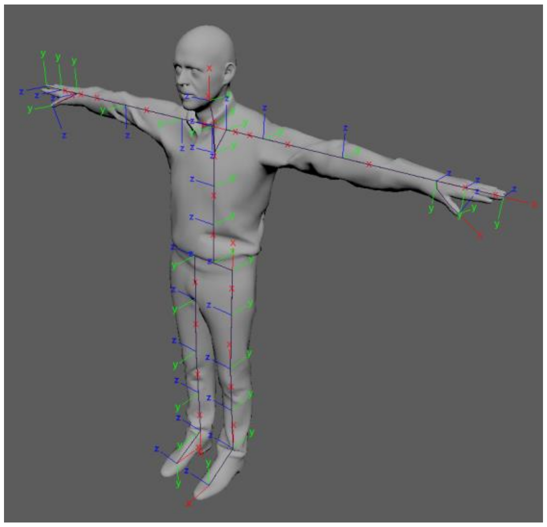


Figure 3: Visual example of sample joint data from the Azure Kinect SDK

4.2 Subsystem II: Object tracking with CEI radar

Colorado Engineering's radar unit was chosen because of its accuracy in measuring high speed objects with accuracy. With an azimuth angle of 33º and 5º angle of elevation, the module will be placed on a tripod two meters away and focused on the release of the ball from the user's hand. Fundamental laws of motion will be applied to the initial movement of the ball to extrapolate its expected path. This processing will be done using MATLAB.

MATLAB sample code from CEI was modified to more accurately track the ball in this project's application. The radar unit was placed facing upwards with the azimuth angle spanning from the 'thrower' to the 'catcher' in this system. To address the issue of streaming the ball's XY data live, MATLAB code was written to write and update a text file containing the positional data as it collected. MATLAB was also used to determine the projected path of the ball. Using initial data, the *polyfit* function was used to find the coefficients of a parabola which models the ball's predicted arc. These coefficients written to a text file to be opened in Vizard and display a curve.

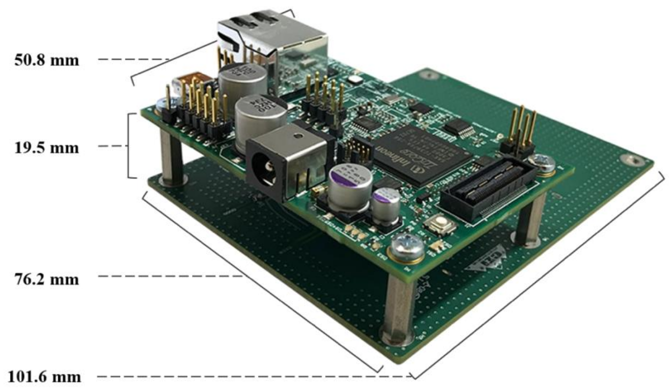


Figure 4: Colorado Engineering's iScan Phantom radar unit

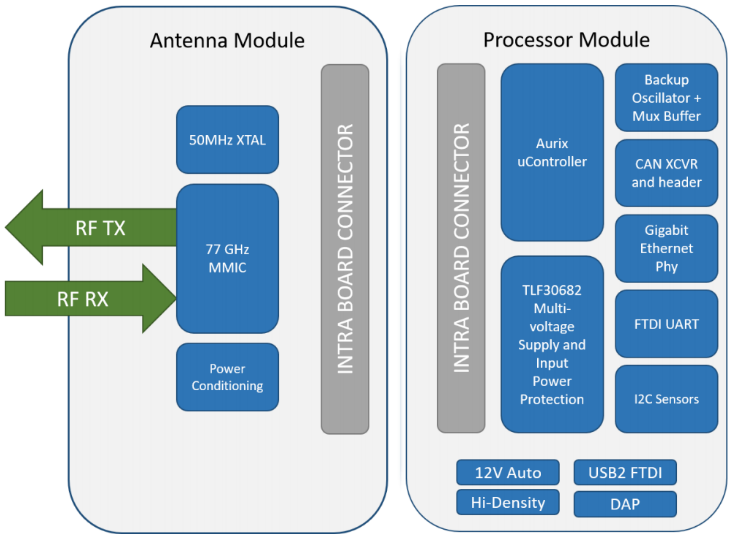


Figure 5: Visual of the iScan Phantom's antenna and processor module

4.3 Subsystem III: HTC Vive Pro Components

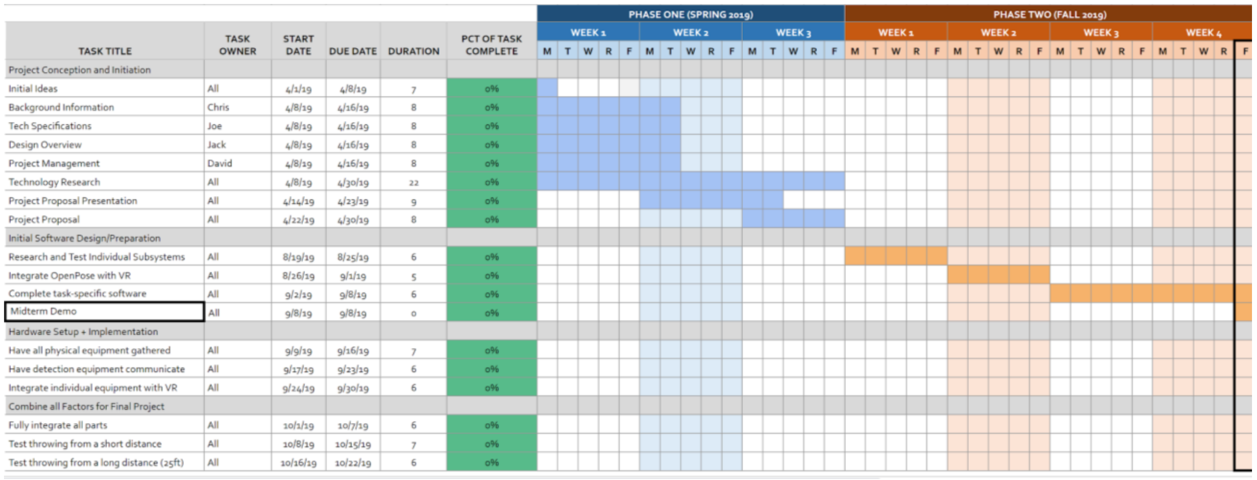
To create the limits of the virtual realm, two base stations are placed 5 meters from each other to create the ‘diagonal’ of the virtual realm. These base stations are angled downwards at about 45º with a field of view of 120º. Vive trackers will be placed on the radar module and on the foot of each user. This will provide the FAAST software with a reference point from which to build an avatar in the virtual realm. Furthermore, the use of Vive trackers will allow for a realistic sense of depth in the virtual space as the users can be perceived in their correct positions within the room. The Vive Pro headset will be worn by the user and run with a Dell XPS 8930.

Figure 6: HTC Vive Pro Headset

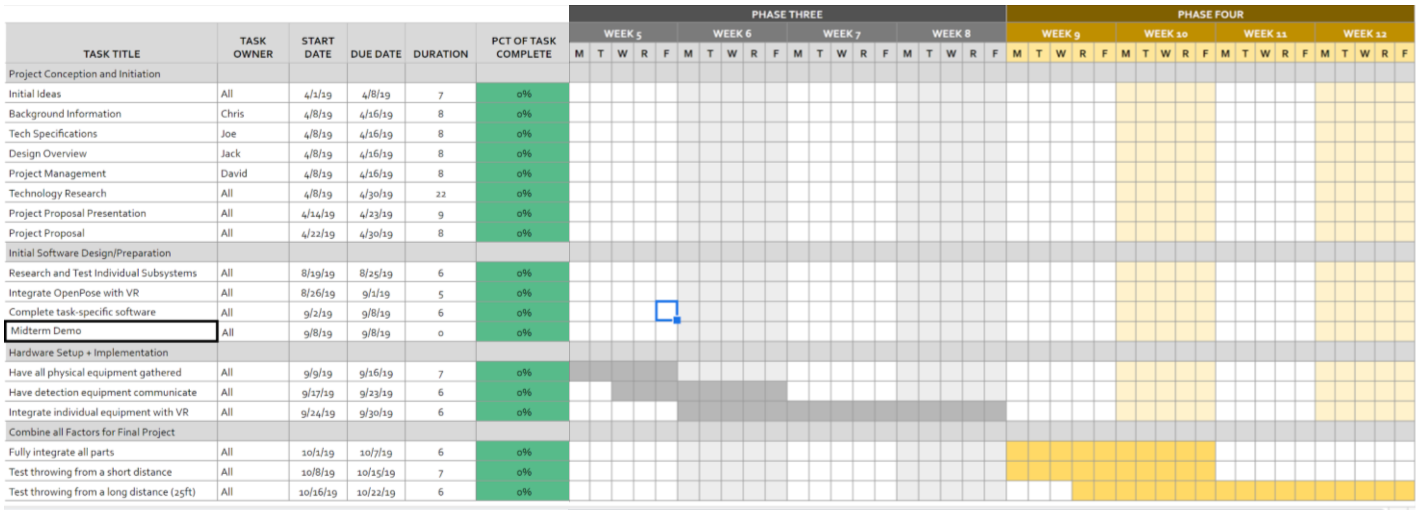
Figure 7: HTC Vive tracker

5. Project Management

5.1 Gantt Chart



TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
Project Conception and Initiation					
Initial Ideas	All	4/1/19	4/8/19	7	0%
Background Information	Chris	4/8/19	4/16/19	8	0%
Tech Specifications	Joe	4/8/19	4/16/19	8	0%
Design Overview	Jack	4/8/19	4/16/19	8	0%
Project Management	David	4/8/19	4/16/19	8	0%
Technology Research	All	4/8/19	4/30/19	22	0%
Project Proposal Presentation	All	4/14/19	4/23/19	9	0%
Project Proposal	All	4/22/19	4/30/19	8	0%
Initial Software Design/Preparation					
Research and Test Individual Subsystems	All	8/19/19	8/25/19	6	0%
Integrate OpenPose with VR	All	8/26/19	9/1/19	5	0%
Complete task-specific software	All	9/2/19	9/8/19	6	0%
Midterm Demo	All	9/8/19	9/8/19	0	0%
Hardware Setup + Implementation					
Have all physical equipment gathered	All	9/9/19	9/16/19	7	0%
Have detection equipment communicate	All	9/17/19	9/23/19	6	0%
Integrate individual equipment with VR	All	9/24/19	9/30/19	6	0%
Combine all Factors for Final Project					
Fully integrate all parts	All	10/1/19	10/7/19	6	0%
Test throwing from a short distance	All	10/8/19	10/15/19	7	0%
Test throwing from a long distance (25ft)	All	10/16/19	10/22/19	6	0%



TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
Project Conception and Initiation					
Initial Ideas	All	4/1/19	4/8/19	7	0%
Background Information	Chris	4/8/19	4/16/19	8	0%
Tech Specifications	Joe	4/8/19	4/16/19	8	0%
Design Overview	Jack	4/8/19	4/16/19	8	0%
Project Management	David	4/8/19	4/16/19	8	0%
Technology Research	All	4/8/19	4/30/19	22	0%
Project Proposal Presentation	All	4/14/19	4/23/19	9	0%
Project Proposal	All	4/22/19	4/30/19	8	0%
Initial Software Design/Preparation					
Research and Test Individual Subsystems	All	8/19/19	8/25/19	6	0%
Integrate OpenPose with VR	All	8/26/19	9/1/19	5	0%
Complete task-specific software	All	9/2/19	9/8/19	6	0%
Midterm Demo	All	9/8/19	9/8/19	0	0%
Hardware Setup + Implementation					
Have all physical equipment gathered	All	9/9/19	9/16/19	7	0%
Have detection equipment communicate	All	9/17/19	9/23/19	6	0%
Integrate individual equipment with VR	All	9/24/19	9/30/19	6	0%
Combine all Factors for Final Project					
Fully integrate all parts	All	10/1/19	10/7/19	6	0%
Test throwing from a short distance	All	10/8/19	10/15/19	7	0%
Test throwing from a long distance (25ft)	All	10/16/19	10/22/19	6	0%

5.2 Personnel

	Personnel			
	Jack Cieslik	Chris Marrs	Joseph Han	David Kaye
Tasks	Wrote custom C code to gather necessary data from Azure Kinect SDK and import to Vizard	Wrote custom code in MATLAB to collect ball data and predict curve, imported to Vizard	Imported external avatars and linked to live data from body trackers	Optimize room set-up for integration of all subsystems
	Wrote custom Python code in Vizard to import live data and view the moving 'throwing' user			

5.3 Parts & Components

Please refer to Section 9.1 Appendix A for a detailed list of hardware and software components used in the completion of this system.

5.4 Skills & Tools

The following skills were necessary to design and implement this system:

- MATLAB coding

- Python coding
- C++ coding

5.6 Budget

Item	Industry Cost	Team Cost
HTC Vive Pro Headset	$600	$0 (provided)
HTC Vive tracker (3)	$300	$0 (provided)
HTC Vive base station (2)	$270	$0 (provided)
CEI radar unit	Unknown	$0 (provided)
Tripod	$20	$0 (provided)
Dell XPS 8930	$1000	$0 (provided)
Azure Kinect camera	$400	$0 (provided)
Glove (2)	$15	$0 (provided)

6. Executive Summary of Achievements and Deliverables

What we have successfully done with this project is:

1. Write code to track the body using the Azure Kinect.
2. Use the CEI Radar to track the ball.
3. Able to tack the positional data and display it in VR.
4. Have the catcher catch the ball a few times.
5. Create a predictive plot of the trajectory of the ball to make catching easier.

The biggest obstacle we have faced in this project is the consistency of the ball tracking as well as the overall lag in the system. The consistency of the ball tracking was not optimal so it was very difficult for the catcher to anticipate where the ball was going to land. The lag did not help with catching either due to the ball already landing before it even displayed in VR. These issues could be solved with more time dedicated to enhancing the code and making more efficient. There was not much developer support for the Azure Kinect yet so there may be more efficient and sophisticated code in the future that could aid in improving the mody tracking of the system. This is talked about in further detail in the Conclusions section.

7. Conclusions

The project ultimately shows that interacting with the physical world in virtual reality is a lot more feasible than we may think. The system was able to track and plot the ball in VR.

Shortcomings: There is a very noticeable amount of lag between the ball tracking and plotting the ball in VR. With this amount of lag in the system it is fairly difficult for the catcher to be able to consistently and successfully catch the ball. A minor shortcoming of the system is the body tracking. The code may get the general position and orientation of the body being tracked, however, it has trouble accurately displaying the body in VR. Both of these issues could be solved by going back through the code and making it more efficient.

Recommendations:

1. To help with the consistency of the body tracking, we should implement moving point averages to allow the movements of the body to look smoother in the virtual realm.
2. To reduce overall lag of the system we could set up a location where all positional data, body tracking and ball tracking, can be stored. The data is then converted into binary so the data is easier to read from Vizard's end.
3. Ultimately we want a system with little lag to increase the consistency of catching the ball. The code that tracks the body is all custom code, however, if there is a future update to Vizard that officially supports the Azure Kinect or if FAAST officially implements the Azure Kinect into their software, the body tracking would be much smoother.

8. References

1. LaMotte, S. (2017, December 13). The very real health dangers of virtual reality. Retrieved from https://www.cnn.com/2017/12/13/health/virtual-reality-vr-dangers-safety/index.html

2. Fingas, J. (2019, July 19). Disney shows how you catch a real ball in VR. Retrieved from https://www.engadget.com/2017/03/20/catch-a-real-ball-in-vr/

3. Safety and Regulatory Guide. (2020). Retrieved from https://www.vive.com/us/legal/
4. "VIVE Pro: The Professional-Grade VR Headset." *VIVE*, www.vive.com/us/product/vive-pro/.
5. Tesych. "Azure Kinect DK Hardware Specification." *Azure Kinect DK Hardware Specification | Microsoft Docs*, docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification.

9. Appendices

9.1 Appendix A: Parts & Components

Hardware Components:

- HTC Vive Pro Headset
- HTC Vive tracker (3)
- HTC Vive base station (2)
- CEI radar unit
- Tripod to mount radar unit
- Dell XPS 8930
- Azure Kinect camera
- Glove to mount Vive tracker (2)

Software Components:

- WorldViz Vizard (Python)
- MATLAB
- Visual Studio (C++)

9.2 Appendix B: Visual Studios Code (Body Tracking)

```
        #include <stdio.h>
#include <stdlib.h>
#include <Windows.h>

#include <k4a/k4a.h>
#include <k4abt.h>
#include <k4abttypes.h>

#define VERIFY(result, error)                                                          \
    if(result != K4A_RESULT_SUCCEEDED)                                                 \
    {                                                                         \
        printf("%s \n - (File: %s, Function: %s, Line: %d)\n", error, __FILE__, __FUNCTION__,
__LINE__); \
        exit(1);                                                              \
    }                                                                         \

int main()
```

```
{

//File Creation
FILE * fp;
FILE * fpp;
fp = fopen("C:\\I_VR\\temp\\test.txt", "w");



k4a_device_t device = NULL;
VERIFY(k4a_device_open(0, &device), "Open K4A Device failed!");

// Start camera. Make sure depth camera is enabled.
k4a_device_configuration_t deviceConfig = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
deviceConfig.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;
deviceConfig.color_resolution = K4A_COLOR_RESOLUTION_OFF;
VERIFY(k4a_device_start_cameras(device, &deviceConfig), "Start K4A cameras failed!");

k4a_calibration_t sensor_calibration;
VERIFY(k4a_device_get_calibration(device, deviceConfig.depth_mode,
deviceConfig.color_resolution, &sensor_calibration),
"Get depth camera calibration failed!");

k4abt_tracker_t tracker = NULL;
k4abt_tracker_configuration_t tracker_config = K4ABT_TRACKER_CONFIG_DEFAULT;
VERIFY(k4abt_tracker_create(&sensor_calibration, tracker_config, &tracker), "Body tracker
initialization failed!");

int frame_count = 0;
do
{
fpp = fopen("C:\\I_VR\\temp\\test.csv", "w");
k4a_capture_t sensor_capture;
k4a_wait_result_t get_capture_result = k4a_device_get_capture(device, &sensor_capture,
K4A_WAIT_INFINITE);
if (get_capture_result == K4A_WAIT_RESULT_SUCCEEDED)
{
frame_count++;
k4a_wait_result_t queue_capture_result = k4abt_tracker_enqueue_capture(tracker,
sensor_capture, K4A_WAIT_INFINITE);
k4a_capture_release(sensor_capture); // Remember to release the sensor capture once you
finish using it
if (queue_capture_result == K4A_WAIT_RESULT_TIMEOUT)
{
// It should never hit timeout when K4A_WAIT_INFINITE is set.
printf("Error! Add capture to tracker process queue timeout!\n");
break;
}
else if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
```

```
{
printf("Error! Add capture to tracker process queue failed!\n");
break;
}

k4abt_frame_t body_frame = NULL;
k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker, &body_frame,
K4A_WAIT_INFINITE);
if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
{
// Successfully popped the body tracking result. Start your processing

size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);



k4abt_skeleton_t skeleton;
k4abt_frame_get_body_skeleton(body_frame, 0, &skeleton);
uint32_t id = k4abt_frame_get_body_id(body_frame, 0);

fprintf(fp, "Frame:%i\n", frame_count);

for (int i = 0; i<26; ++i )
{

k4abt_joint_t joint1 = skeleton.joints[i];
double x = (((joint1).position).xyz).x;
double y = (((joint1).position).xyz).y;
double z = (((joint1).position).xyz).z;

double qw = joint1.orientation.wxyz.w;
double qx = joint1.orientation.wxyz.x;
double qy = joint1.orientation.wxyz.y;
double qz = joint1.orientation.wxyz.z;



fprintf(fp, "j%i: (%f,%f,%f) Orientation: (%f,%f,%f,%f) \n", i, x / 1000, y / 1000, z / 1000, qw, qx,
qy, qz);
fprintf(fpp, "%f,%f,%f,%f,%f,%f,%f\n", x / 1000, y / 1000, z / 1000, qw, qx, qy, qz);

}
  fprintf(fp,"\n");

fclose(fpp);



k4abt_frame_release(body_frame); // Remember to release the body frame once you finish
using it
}
else if (pop_frame_result == K4A_WAIT_RESULT_TIMEOUT)
```

```
{
//  It should never hit timeout when K4A_WAIT_INFINITE is set.
printf("Error! Pop body frame result timeout!\n");
break;
}
else
{
printf("Pop body frame result failed!\n");
break;
}
}
else if (get_capture_result == K4A_WAIT_RESULT_TIMEOUT)
{
// It should never hit time out when K4A_WAIT_INFINITE is set.
printf("Error! Get depth frame time out!\n");
break;
}
else
{
printf("Get depth capture returned error: %d\n", get_capture_result);
break;
}

Sleep(50);
} while (frame_count < 5000);

printf("Finished body tracking processing!\n");

k4abt_tracker_shutdown(tracker);
k4abt_tracker_destroy(tracker);
k4a_device_stop_cameras(device);
k4a_device_close(device);


fclose(fp);

return 0;
}
```

9.3 Appendix B: MATLAB Code (CEI Radar)

```
%**************************************************************************
*
%                          IMPORTANT NOTICE
%**************************************************************************
*
% THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS,
% IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES
% OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
% SOFTWARE. INFINEON SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR
SPECIAL,
% INCIDENTAL, OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
%*********************************************************************
*

clc
clearvars
close all

addpath('C:\I_VR\CEI_Matlab_files_07_19_2019');
%% connect to board
board = strata.utils.connectToBoard();
boardRadar = board.getIBoardRadar();

%% get config structs
mmicConfig = strata.config.rangeDoppler();
sequence = strata.sequence.range3Tx_DoA();
stages = strata.stages.rangeData();

%% configure all settings
boardRadar.setConfiguration(mmicConfig);
boardRadar.setSequence(sequence);
boardRadar.setProcessingStages(stages);
boardRadar.setup();

%% Extra configuration
radarRxs = boardRadar.getIRadarRxs(0);
radarRxs.enableDividerOutput(true); % enable divider output for test

%% Get values for visualization
radarInputInfo = boardRadar.getInputInfo();
samples = radarInputInfo.samples;
ramps = double(radarInputInfo.ramps);
NoTX = double(radarInputInfo.txChannels);
NoRX = double(radarInputInfo.rxChannels);
%NoRange = 2^nextpow2(double(samples))/2;
NoRange = 512;

%% scaling for plot

c0=3e8;
%sweep duration
tsw=(sequence.tRamp-sequence.tRampStartDelay);
%effective RF bandwidth
bsw=(sequence.ramps(1).fDelta/sequence.tRamp*(tsw));
%pulse repetition time
PRT=sequence.tRamp+sequence.tJump+sequence.tWait;
if sequence.modulation == 0
    PRT=PRT/NoTX;
    ramps=ramps/NoTX;
end
%maximum velocity
f_sin = sequence.ramps(1).fStart+sequence.ramps(1).fDelta/2;
```

```
vmax=c0/2/(f_sin)/PRT/2;
rmax=mmicConfig.sampleRate*tsw/bsw*c0/2/2;

r_vect=linspace(0,0.5-1/NoRange,NoRange)*rmax*2;
v_vect=linspace(-0.5,0.5-1/ramps,ramps)*2*vmax;

FftBinSz = 8; % bytes;
nciBinSz = 4; % bytes;
Fft2Sz = NoRange*ramps*NoRX*NoTX*FftBinSz;
NciSz = NoRange*ramps*nciBinSz;

%% AoA parameters
NFFTAnt              =  256;
NrMIMO               =  8;       % Number of MIMO channels
min_range            =   1;  % minimum calculated range in meters
max_range            =   2.5;  % maximum calculated range in meters

[~, min_range_idx]  =   min(abs(r_vect - min_range));
[~, max_range_idx]  =   min(abs(r_vect - max_range));
range_view           =   r_vect(min_range_idx : max_range_idx);

% Window function for  NrMIMO virtual receive channels
winAnt               =   hanning(NrMIMO);
scale_winAnt         =   sum(winAnt);
winAnt2D             =   repmat(winAnt.',numel(range_view),1);
winAnt3D             =
permute(repmat(winAnt,1,numel(range_view),ramps),[2 3 1]);
angle_deg            =   (-NFFTAnt/2 : NFFTAnt/2-1)'./NFFTAnt.*180;

% Positions for polar plot of cost function
vU                   =   linspace(-1, 1, NFFTAnt);
[mRange , mU]        =   ndgrid(range_view, vU);
mX                   =   mRange.*mU;
mY                   =   mRange.*cos(asin(mU));

%% RD demo
figure(1);clf;
box on;

%% Setting count variables
k = 0;
%while (k<10)
x = [];
y = [];
xvelo = [];
yvelo = [];
%xvelo(1) = 0;
%yvelo(1) = 0;
xacc = [];,
yacc = [];
time = [];
drag = [];   %D = Cd*((r*v^2)/2)*A. using rad = 3.65cm, Cd = .3, r =
.92787
count = 1;
```

```
datasize = 4;

boardRadar.startMeasurements();
inittime = posixtime(datetime(datestr(now)));
timeidx= [];
fprintf('Starting time is %d \n', inittime);
for idx=1:1:2000000
    idx

    z = posixtime(datetime(datestr(now)));
    fprintf('%d',z)

    [measurement, timeStamp] = board.getFrameBuffer(20000);
    % Data to extract is Tx1 and Tx3 from the linear measurement memory
    % Reshape to get rid of unwanted half of FFT and remap the Rx channels
    RD_data = typecast(measurement, 'int32');
    %RD_data = double(complex(RD_data(1:2:end), RD_data(2:2:end)));
    %plot(abs(RD_data))
    RD_data = double(reshape(complex(RD_data(1:2:end), RD_data(2:2:end)),
NoRange*2, []));
    %RD_data = RD_data(1:NoRange,[3 4 1 2 11 12 9 10]);
    RD_data = RD_data(1:NoRange,[1 2 3 4 9 10 11 12]);

    % Look at range slices at zero doppler only to compare
    range_profile = RD_data;
    range_profile_FOV = range_profile(min_range_idx : max_range_idx, :);
    %plot(unwrap(angle(range_profile)));
    spec =
fftshift(fft(range_profile_FOV.*winAnt2D,NFFTAnt,2)/scale_winAnt,2);

    % normalize cost function
    JdB              =   20.*log10(abs(spec));
    %[JMax, idx_JMax] =   max(JdB(:));
    JMax = 130;
    JNorm            =   JdB - JMax;
    JNorm(JNorm < -20)  =   -20;

    if idx==1
        %polar plot
        figure(1);
        set(gcf,'visible','off')
        h = surf(mX,mY,JNorm);
        J1 = JNorm;
        shading flat;
        view(0,90);
        xlim([-5 5]);
        ylim([0 5]);
        caxis([-20 -8]);
        axis equal; xlabel('x (m)'); ylabel('y (m)'); colormap('jet');

        fileID = fopen('radartest.txt','w+');
    end

    if ~ishandle(h)
```

```
        break;
    end

    JNew = JNorm - J1 - 20;       %eliminate nonmoving objects
    JNew(JNew < -20)  =   -20;
    set(h,'ZData',JNew);

    drawnow;

%     maxJNew = max(JNew(:));      %find maximum intensity of this index
%     [row,col] = find(JNew == maxJNew);
%     xcoord = mX(row,col);
%     ycoord = mY(row,col);
    dlmwrite('radartest.txt', JNew,'-append','newline','pc')

    if (max(JNew(:))>-16)
        [i,j] = find(JNew>-16);
        row = round(mean(i));
        col = round(mean(j));
        %time(count) = posixtime(datetime(datestr(now)));
        timeidx(count) = idx/20;
        x(count) = mX(row,col);
        y(count) = mY(row,col);

%         xvelo(1) = 0;
%         yvelo(1) = 0;
%         xacc(1) = 0;
%         yacc(1) = 0;
%         drag(1) = 0;

%         if (count > 1)
%             xvelo(count) = (x(count) - x(count-1))/(timeidx(count)-
timeidx(count-1));
%             yvelo(count) = (y(count) - y(count-1))/(timeidx(count)-
timeidx(count-1));
%             xdrag(count) = .000582*xvelo(count);
%             ydrag(count) = .000582*yvelo(count);
%         end

        count = count + 1;
    %x(idx-1) = mY(row,col);
    %y(idx-1) = mX(row,col);
    end
    if length(x) > datasize
        break;
    end

end


% Plot flight path of
ball

% for k = 1:idx
%     [i,j] = find(JNew>-16);
%     row = mean(i);
%     col = mean(j);
%     x(k) = mX(row,col);
%     y(k) = mY(row,col);
% end

boardRadar.stopMeasurements();

%%%%%%%%% Destroy the board object %%%%%%%%%
board.delete();

%for debugging, unload the wrapper dll to be able to update it
clear wrapper_matlab

%% Scatterplot and trajectory


c = polyfit(-x,y,2);
Qx =[-max_range/4:.01:max_range];
Qy = c(1)*(Qx.^2) + c(2)*Qx + c(3);

i = 1;
tdiff = zeros(datasize-1,1);
vx = zeros(datasize-1,1);
vy = zeros(datasize-1,1);
v = zeros(datasize-1,1);

while (i < datasize)
    tdiff(i) = timeidx(i+1) - timeidx(i);
    vx(i) = (x(i+1) - x(i))/(tdiff(i));
    vy(i) = (y(i+1) - y(i))/(tdiff(i));
    i = i + 1;
end

xvelocity = mean(vx);
yvelocity = mean(vy);
v = sqrt(xvelocity^2 + yvelocity^2);

fileID = fopen('C:\I_VR\temp\Ball.csv','w');
fprintf(fileID, '%g,%g,%g,%g,%g,%g,%g',
c(1),c(2),c(3),x(1),y(1),xvelocity,yvelocity)
fclose(fileID);

plot(-x,y,'o');
title('Scatterplot of x and y Positional Values');
xlabel('x');
ylabel('y');
hold on;

plot(Qx,Qy);

pause(5);
k = k + 1;
```

```
%end
%IntitalVx
%InitialVy

% figure
% plot(timeidx,yvelo,'o');
% title('Scatterplot of Horizontal Velocity vs. Time');
% xlabel('Time');
% ylabel('Horizontal Velocity(m/s)');
```

9.4 Appendix B: Python Code (Main Program)

```
import csv
import viz
import vizact
import vizfx
import time
import Euler
import vizshape
import vizconnect
import steamvr
import math

#avatar = viz.addChild('mark.csf')
rhand = viz.addChild('glove.cfg')
catchHand = viz.addChild('glove.cfg')
lhand = viz.addChild('glove_left.cfg')


def Map_Joints():
      x = []
      y = []
      z = []
      w = []
      qx = []
      qy = []
      qz = []

      euler = []
      pitch = 0
      roll = 0
      yaw = 0

      with open('C:\\I_VR\\temp\\test.csv') as csvfile:
            readCSV = csv.reader(csvfile, delimiter=',')

            for row in readCSV:
                  x.append(row[0])
                  y.append(row[1])
                  z.append(row[2])
                  w.append(row[3])
```

```
                qx.append(row[4])
                qy.append(row[5])
                qz.append(row[6])

            x = map(float,x)
            y = map(float,y)
            z = map(float,z)
            w = map(float,w)
            qx = map(float,qx)
            qy = map(float,qy)
            qz = map(float,qz)



            for i in range(0,len(x)):
                if len(x) > 25:
                    #globals()['zxratio'] = (tracker_pos[2] -
zero_pos[2])/(tracker[0] - zero_pos[0])

                    #xOffset = tracker_pos[0]
                    #yOffset = tracker_pos[1] + max(y)
                    #zOffset = tracker_pos[2]

                    xOffset = 0;
                    yOffset = max(y)
                    zOffset = -2.3;

                    exec('joint' + str(i) +'.setPosition([x[' + str(i)
+ ']+ xOffset,-y[' + str(i) + '] + yOffset,z[' + str(i) + '] + zOffset])')
                    euler =
Euler.quaternion_to_euler(w[i],qx[i],qy[i],qz[i]);
                    yaw = euler[0]
                    roll = euler[1]
                    pitch = euler[2]
                    exec('joint' + str(i)
+'.setEuler([yaw,pitch,roll])')
                    #viz.link(joint11, rhand)
                    #viz.link(joint7,rhand)

def Map_Ball():
      c1 = []
      c2 = []
      c3 = []
      bx = []
      by = []
      v = []
      vx = 0
      vy = []
      vz = 0
      yball = 0
      xball = 0

      #print(tracker_pos)
      #print(head_pos)
```

```
with open('c:\\I_VR\\temp\\Ball.csv') as csvfile:
      readCSV = csv.reader(csvfile, delimiter=',')

      for row in readCSV:
            c1.append(row[0])
            c2.append(row[1])
            c3.append(row[2])
            bx.append(row[3])
            by.append(row[4])
            v.append(row[5])
            vy.append(row[6])

      c1 = map(float,c1)
      c2 = map(float,c2)
      c3 = map(float,c3)
      bx = map(float,bx)
      by = map(float,by)
      v = map(float,v)
      vy = map(float,vy)



            #print(Ty)
            #print(k)


      t = (time.clock()-tstamp)

      if (temp == v[0]):
            vangle = math.atan(zxratio)
            vx = v[0]*math.cos(vangle)
            vz = v[0]*math.sin(vangle)

            xball = vx*t
            zball = vz*t
            vball = v[0]*t



            yball = c1[0]*pow(vball,2) + c2[0]*(vball) + trackery
            ball.setPosition(xball+trackerx,yball,zball+trackerz)
            #print(xball,yball)
            #print(t)
            #print(temp,vx[0])
            globals()['temp'] = v[0]


      else:

            globals()['trackerx'] = tracker_pos[0]
            globals()['trackery'] = tracker_pos[1]
            globals()['trackerz'] = tracker_pos[2]
```

```
                  for i in range(0,200):
                        globals()['zxratio'] = (trackerz - 
zero_pos[2])/(trackerx - zero_pos[0])
                        k = float(i)/320
                        g = float(i)*zxratio/320
                        kg = math.sqrt(pow(k,2)+pow(g,2))

                        #angle = math.tan(zxratio)

                        #xoffset = (trackerx - zero_pos[0]) - 
bx[0]*math.asin(angle)
                        #zoffset = (trackerz - zero_pos[2]) - 
bx[0]*math.acos(angle)



                        Ty = c1[0]*pow(kg,2) + c2[0]*(kg) + trackery


                        exec('T' + str(i)
+'.setPosition(k+trackerx,Ty,trackerz + g)')

                  globals()['tstamp'] = time.clock()
                  #print(temp,vx[0])
                  globals()['temp'] = v[0]

def collision_detection():
      catchHand.enable(viz.COLLIDE_NOTIFY)
      #sphere_controller1.enable(viz.COLLIDE_NOTIFY)
      ball.enable(viz.COLLIDE_NOTIFY)

      catchHand.collideSphere(friction = .00000001, hardness = 1)
      #sphere_controller1.collideSphere(friction = .00000001, hardness = 
1)
      ball.collideSphere(friction = .00000001, hardness = 1)

def onCollide(e):
      if e.obj2:
            print('collision')
            viz.link(catchHand,ball)

#def Radar():

def HTC_Vive():
      hmd = steamvr.HMD();
      if not hmd.getSensor():
            sys.exit('HMD not detected')
      navigationNode = viz.addGroup()
      globals()['viewLink'] = viz.link(navigationNode,viz.MainView)
      viewLink.preMultLinkable(hmd.getSensor())

      # Create model for tracker1
      tracker1 = steamvr.getTrackerList()[0]
      tracker1.model = tracker1.addModel(parent=navigationNode)
      tracker1.model.disable(viz.INTERSECTION)
      globals()['controller_tracker1'] = viz.link(tracker1,
tracker1.model)

      globals()['sphere_controller1'] = vizshape.addSphere(radius=0.01252,
color = viz.RED)
      sphere_controller1.alpha(0.0)
      viz.link(controller_tracker1,sphere_controller1)

      # Create model for tracker2
      globals()['tracker2'] = steamvr.getTrackerList()[1]

      tracker2.model = tracker2.addModel(parent=navigationNode)
      tracker2.model.disable(viz.INTERSECTION)
      globals()['controller_tracker2'] = viz.link(tracker2,
tracker2.model)

      globals()['sphere_controller2'] = vizshape.addSphere(radius=0.01252,
color = viz.RED)
      sphere_controller2.alpha(0.0)
      viz.link(controller_tracker2,sphere_controller2)


      #create model for tracker3

      globals()['tracker3'] = steamvr.getTrackerList()[2]

      tracker3.model = tracker3.addModel(parent=navigationNode)
      tracker3.model.disable(viz.INTERSECTION)
      globals()['controller_tracker3'] = viz.link(tracker3,
tracker3.model)

      globals()['sphere_controller3'] = vizshape.addSphere(radius=0.01252,
color = viz.RED)
      sphere_controller3.alpha(0.0)
      viz.link(controller_tracker3,sphere_controller3)
      viz.link(tracker3,catchHand)

      #vizconnect.go('htcvive.py')
def Tracker_Pos():
      #Head Pos
      globals()['head_pos'] = viewLink.getPosition()
      globals()['head_orientation'] = viewLink.getEuler()

      #tracker Position and Euler
      globals()['tracker_pos'] = controller_tracker1.getPosition()
      globals()['tracker_orientation'] = controller_tracker1.getEuler()

      globals()['zero_pos'] = controller_tracker2.getPosition()

      #write for third tracker
```

```
def Intialize():

	viz.setMultiSample(4)
	viz.fov(60)
	viz.go()
	HTC_Vive()

	viz.phys.enable()
	viz.MainView.setPosition([.5,0,-2])
	room = vizfx.addChild('dojo.osgb')

	for i in range(0,26):
		globals()['joint%s' % i] = viz.addChild('ball.wrl')
		exec('joint' + str(i) + '.setScale(.05,.05,.05)')
	for i in range(0,200):
		globals()['T%s' % i] = viz.addChild('ball.wrl')
		exec('T' + str(i) + '.setScale(.03,.03,.03)')
	globals()['temp'] = 0
	globals()['trackerx'] = 0
	globals()['trackery'] = 0
	globals()['trackerz'] = 0
	globals()['zxratio'] = 1
	globals()['ball'] = viz.addChild('Ball.wrl')
	ball.setScale(.2,.2,.2)



Intialize()
#collision_detection()
vizact.onupdate(0,Tracker_Pos)
vizact.onupdate(0,Map_Joints)
globals()['tstamp'] = time.clock()
vizact.onupdate(0,Map_Ball)
#viz.callback(viz.COLLIDE_BEGIN_EVENT,onCollide)
```

9.5 Appendix B: Python Code (Quarternians)

```
import csv
import viz
import vizact
import vizfx
import time
import Euler
import vizshape
import vizconnect

rhand = viz.addChild('glove.cfg')
lhand = viz.addChild('glove_left.cfg')

def Map_Joints():
	x = []
	y = []
	z = []
```

```
      w = []
      qx = []
      qy = []
      qz = []

      euler = []
      pitch = 0
      roll = 0
      yaw = 0

      with open('test1.csv') as csvfile:
            readCSV = csv.reader(csvfile, delimiter=',')

            for row in readCSV:
                  x.append(row[0])
                  y.append(row[1])
                  z.append(row[2])
                  w.append(row[3])
                  qx.append(row[4])
                  qy.append(row[5])
                  qz.append(row[6])

            x = map(float,x)
            y = map(float,y)
            z = map(float,z)
            w = map(float,w)
            qx = map(float,qx)
            qy = map(float,qy)
            qz = map(float,qz)



            for i in range(0,len(x)):
                  if len(x) > 25:
                        yOffset = max(y)
                        exec('joint' + str(i) +'.setPosition([x[' + str(i)
+ '],-y[' + str(i) + '] + yOffset,z[' + str(i) + ']])')
                        euler =
Euler.quaternion_to_euler(w[i],qx[i],qy[i],qz[i]);
                        yaw = euler[0]
                        roll = euler[1]
                        pitch = euler[2]
                        exec('joint' + str(i)
+'.setEuler([yaw,pitch,roll])')
                        viz.link(joint11, rhand)
                        viz.link(joint7,lhand)


def Intialize():

      viz.setMultiSample(4)
      viz.fov(60)
      viz.go()
      vizconnect.go('htcvive.py')
```

```
    viz.MainView.setPosition([.5,-2,-2])
    room = vizfx.addChild('dojo.osgb')

    for i in range(0,26):
        globals()['joint%s' % i] = viz.addChild('arrow.wrl')
        exec('joint' + str(i) + '.setScale(.05,.05,.05)')

Intialize()
vizact.onupdate(0,Map_Joints)
```