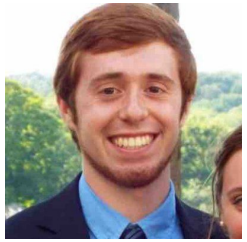
Step

A Step Forward in Virtual Reality

Team Step



Ryan Daly
Electrical Engineer

Joseph Roberts
Electrical Engineer

Jared Ricci
Electrical Engineer

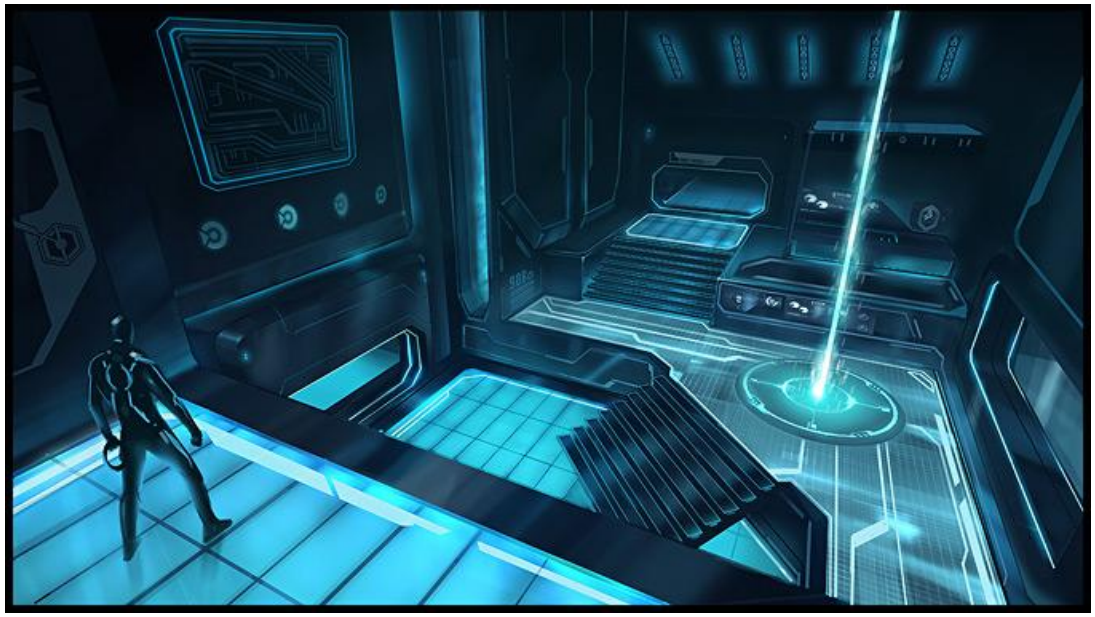
Steven So
Electrical Engineer

Motivation

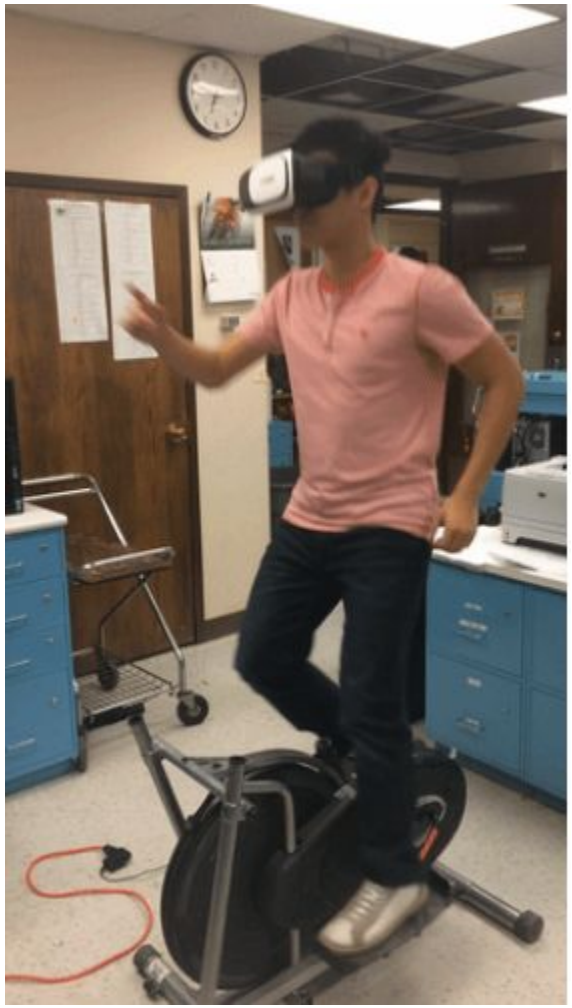
- Current Virtual Reality has lacked the REALITY aspect
- The market is pushing smartphone VR; peripherals can make the experience better
- Users do not feel immersed in the environment
- VR hasn’t reached its full potential

Introduction

- Step is a new virtual reality environment that will immerse the user with no added hardware
- The user will be able to interact, move, and feel the environment

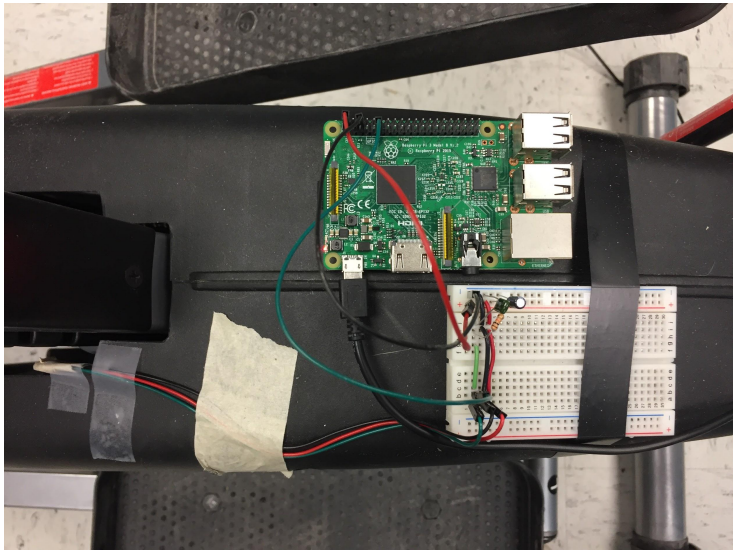


DEMO



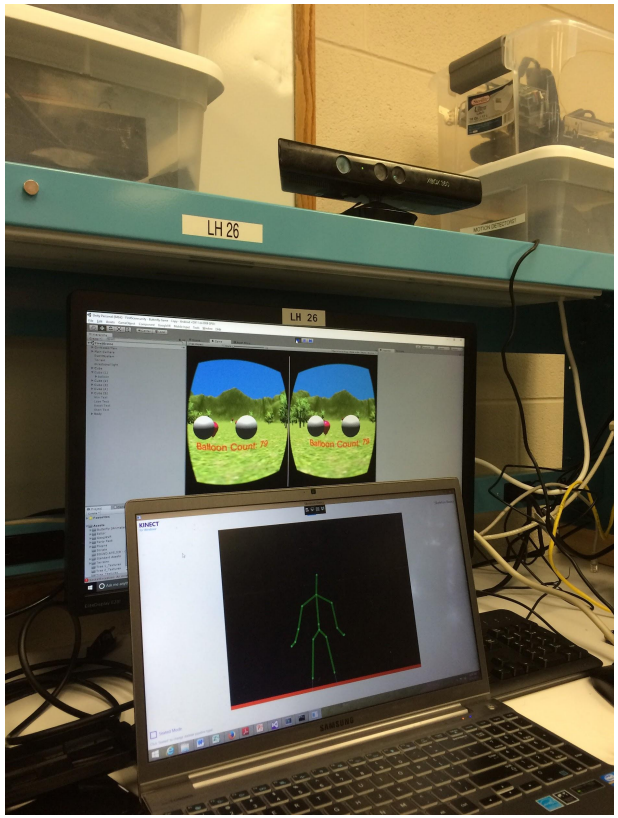
MDR Deliverables Met?

- ✓ Precise speed control with elliptical
- Adequate Kinect motion sensing
- Data processing and transmitting to android
- Programmed VR environment using inputs



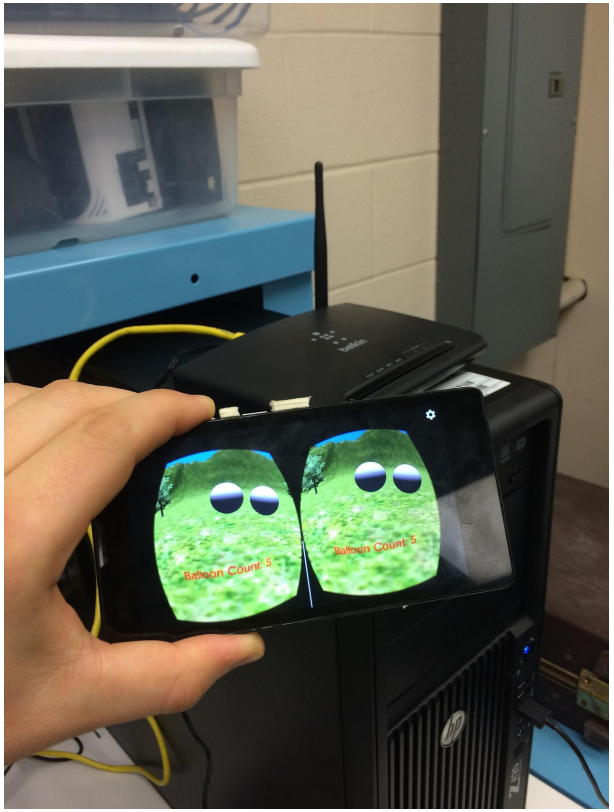
MDR Deliverables Met?

- ✓ Precise speed control with elliptical
- ✓ Adequate Kinect motion sensing
  - Data processing and transmitting to android
  - Programmed VR environment using inputs



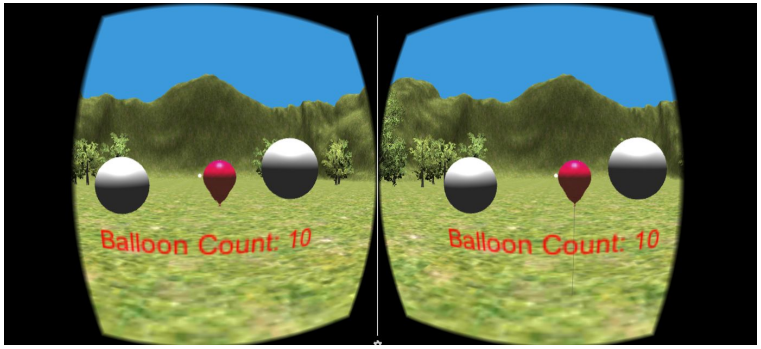
MDR Deliverables Met?

- ✓ Precise speed control with elliptical
- ✓ Adequate Kinect motion sensing
- ✓ Data processing and transmitting to android
- Programmed VR environment using inputs



MDR Deliverables Met?

- ✓ Precise speed control with elliptical
- ✓ Adequate Kinect motion sensing
- ✓ Data processing and transmitting to android
- ✓ Programmed VR environment using inputs



Our Solution

- Create a 360 degree, 3 dimensional virtual environment on the android
- Create a system that will detect the user's movement like walking, running, and arm motions
- Create a structure in which the virtual world can be mimicked

Overall Requirements

PDR Feedback:
Quantifiable requirements missing

- User is able to freely move in virtual environment and control movement speed
  - Speed Accuracy within .5 MPH
- Hand and arm motion is translated to in-game action
  - Depth Accuracy, standard deviation within 1 inch
- Control latency less than 200 ms[1]
- User does not have to wear any equipment beyond VR headset
- Maintain framerate at 60 FPS

[1] Leadbetter, Richard. "Console Gaming: The Lag Factor." Eurogamer.net. Eurogamer, 09 May 2009. Web. 01 Dec. 2016. http://www.eurogamer.net/articles/digitalfoundry-lag-factor-article.

Overall Requirements

PDR Feedback:
Quantifiable requirements missing

- User is able to freely move in virtual environment and control movement speed
  - Speed Accuracy within .5 MPH
- Hand and arm motion is translated to in-game action
  - Depth Accuracy, standard deviation within 1”
- Control latency less than 200 ms[1]
- User does not have to wear any equipment beyond VR headset
- Maintain framerate at 60 FPS

[1] Leadbetter, Richard. "Console Gaming: The Lag Factor." Eurogamer.net. Eurogamer, 09 May 2009. Web. 01 Dec. 2016. http://www.eurogamer.net/articles/digitalfoundry-lag-factor-article.

Design

Elliptical

Raspberry Pi

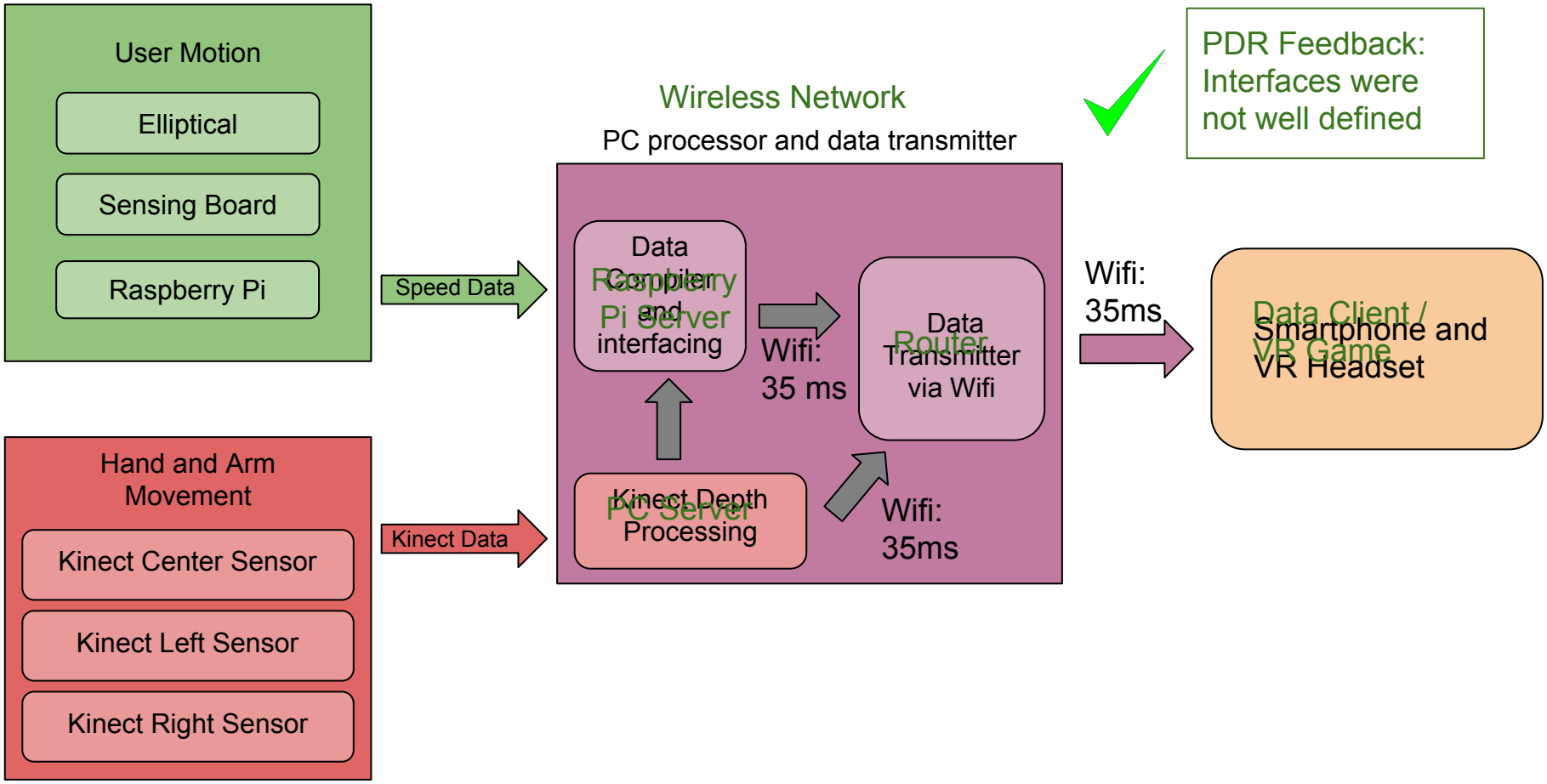
Router

Android Phone

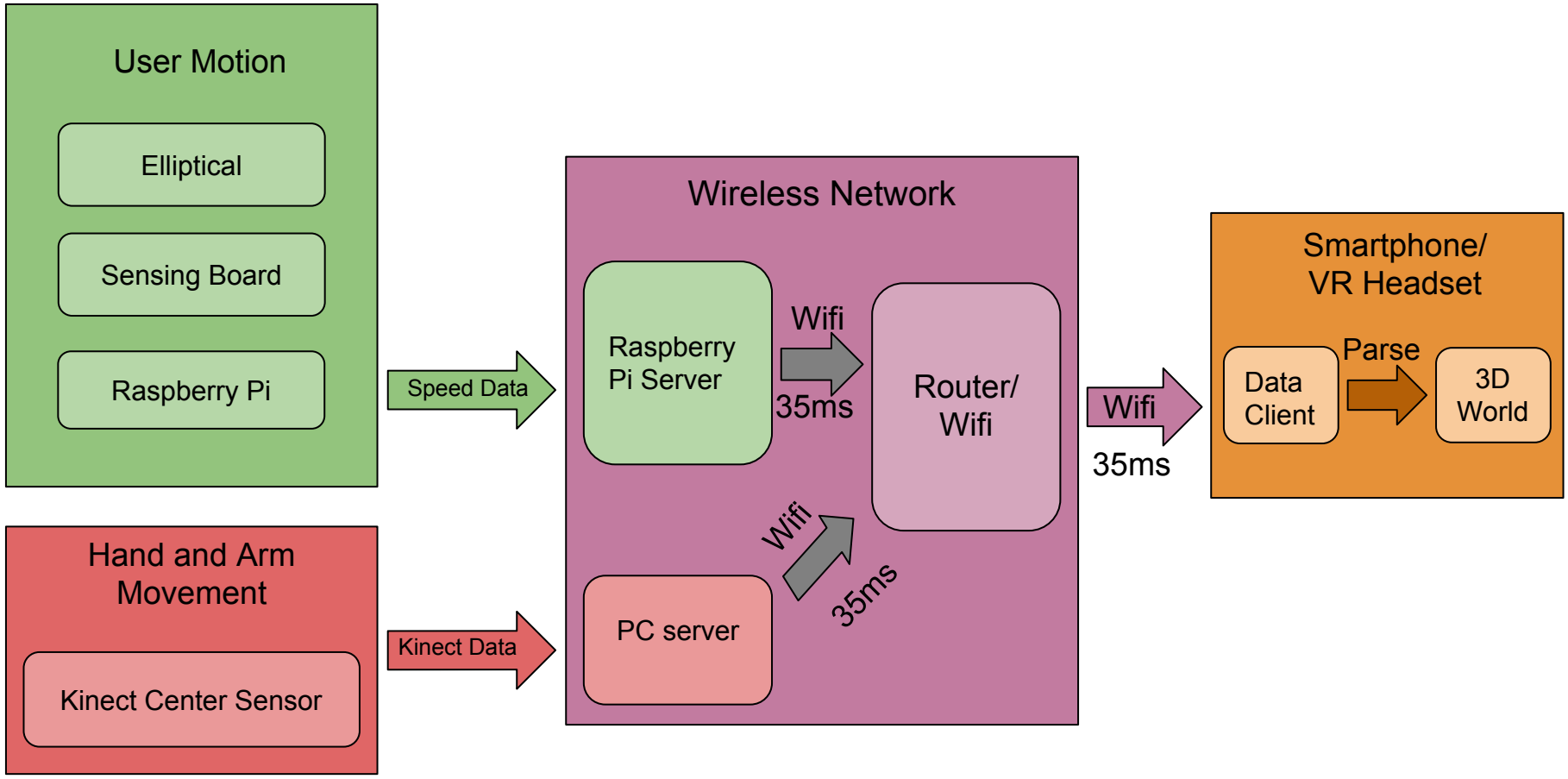
Kinect

PC

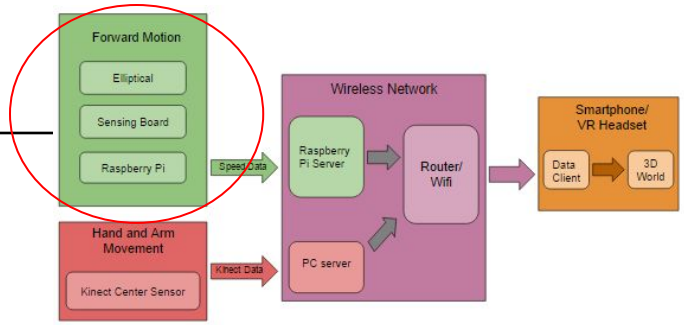
PDR Block Diagram



MDR Block Diagram

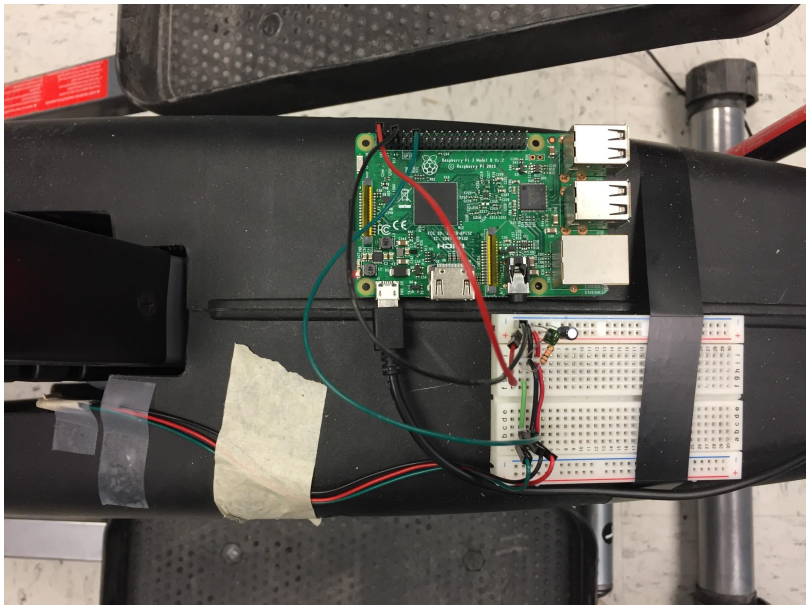


User Motion



Implementation

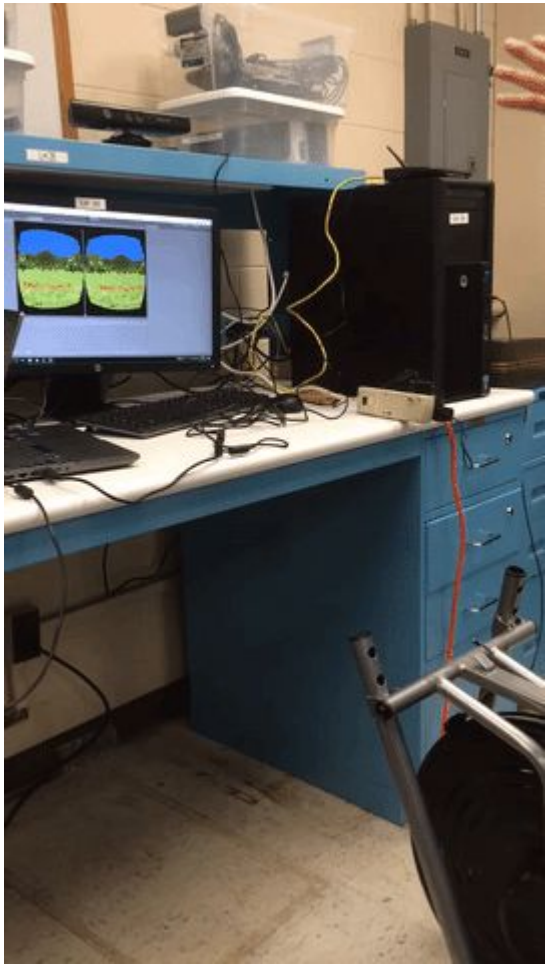
- Raspberry Pi with attached sensor board
- Rotational speed measured using magnetic sensor
- Data transmitted wirelessly to smartphone



Changes from Initial Design

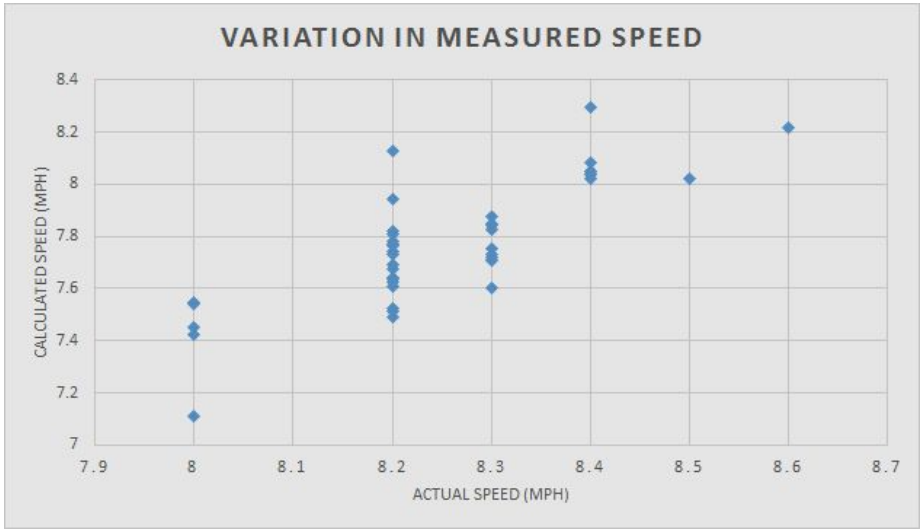
- Multi-magnet sensing
- Data sent directly to smartphone, not to PC

User Motion Functionality

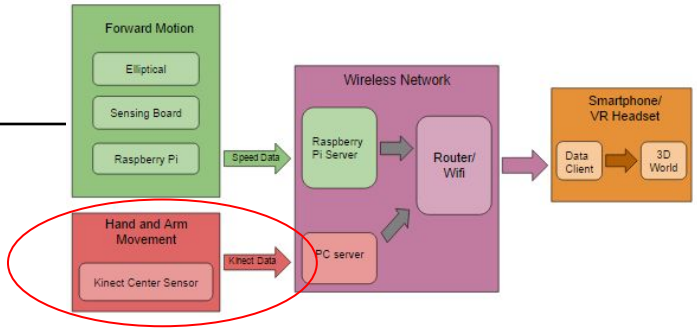


User Motion Functionality

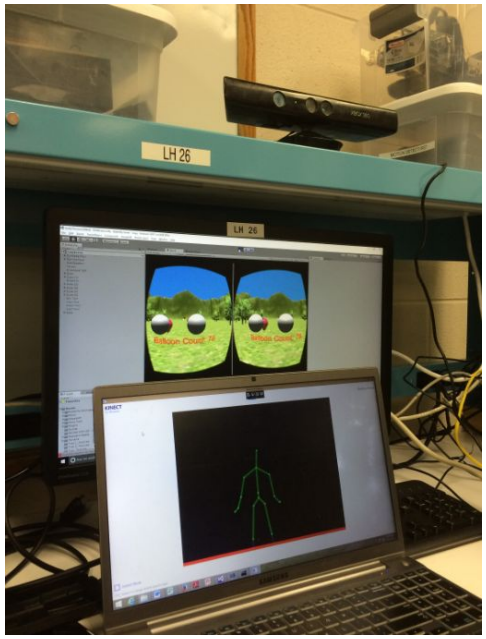
- Calculated speed is accurate ±.5 MPH
- Standard Deviation = .152 MPH (n=40)
- Latency is under 200 ms requirement
- Average latency = 70.08 ms (n=100)



Hand tracking - Kinect

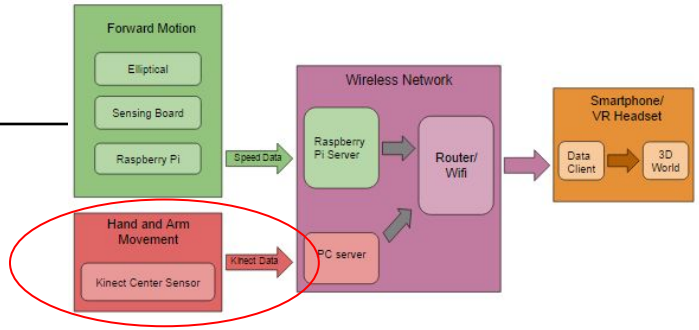


- Requirements
  - Depth within 1” standard deviation for arm movement
  - Real-time processing and transmitting

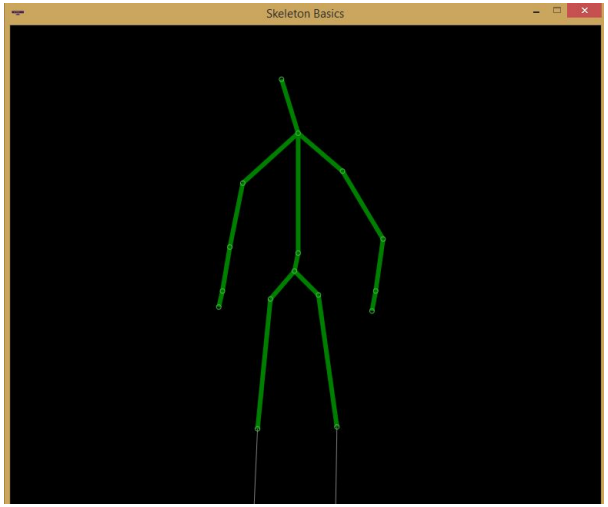


- Steps
  - Create server
  - Extract Position
  - Wait for request
  - Write position as string to server

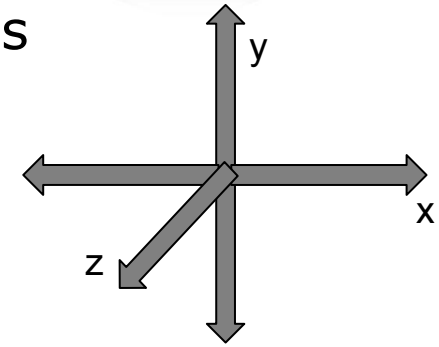
Kinect Joint tracking



- Skeleton
  - Each node is a "Joint type" object
  - Using left and right hands
    - Shown as balls in balloon popper



- Position
  - (x,y,z) coordinates
  - Unit: meters



Kinect Depth Results

- Depth was thought to be an issue
  - Collected data and measured depth performance

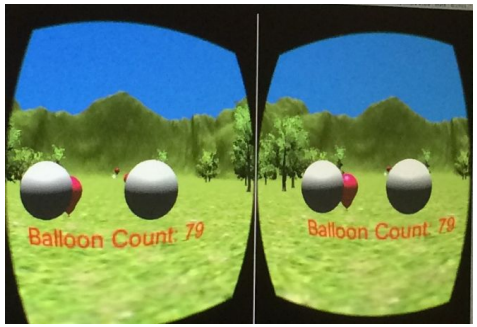
Tape Measured	Calculated Kinect mean (n=100)	Standard Deviation (m)	Standard Deviation (in)	Performance <inch (=0.0254 m)	Performance <4cm
1.0 m	1.0021 m	0.0214	0.844	80%	93%
2.25 m	2.2452 m	0.0218	0.859	77%	96%

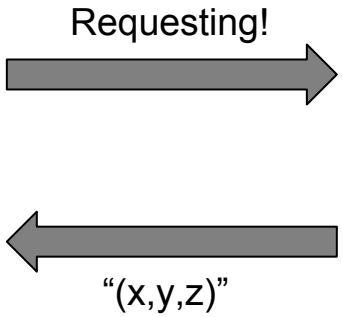
- Experiment setup
  - Measured points from Kinect with tape measure
  - Compare with what Kinect returned

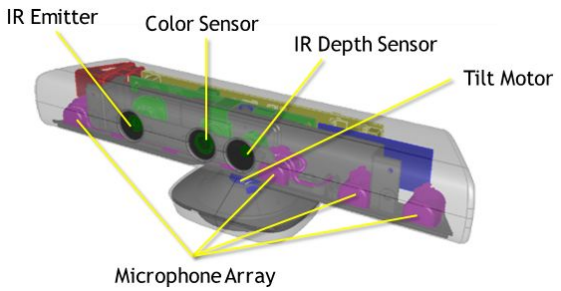
Kinect Processing and Data Transfer

- Internal frame rate [2]
  - 30 fps

- (x,y,z) coordinates extracted
  - Coordinates converted to string
  - String written to server stream when requested

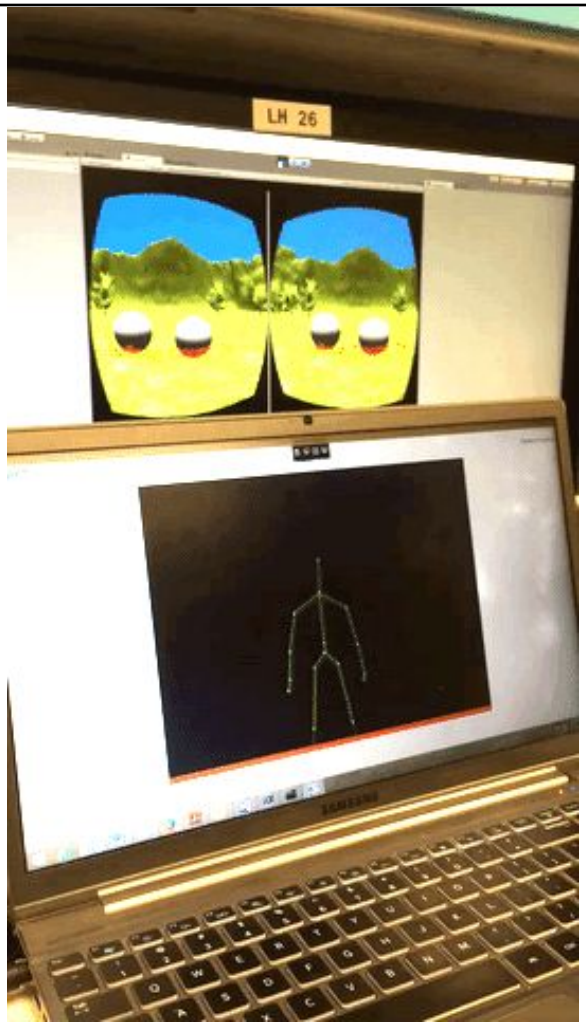




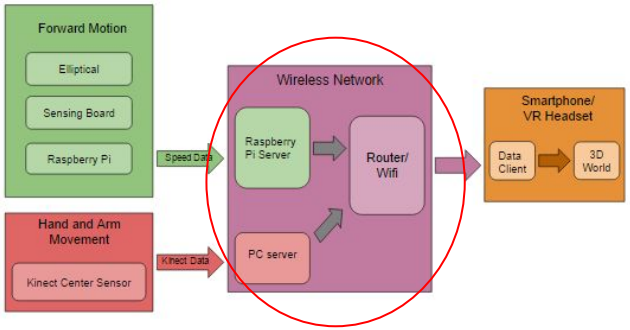


[2]"Kinect for Windows Sensor Components and Specifications", *Msdn.microsoft.com*, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx. [Accessed: 30- Nov- 2016].

Kinect Used in the Game

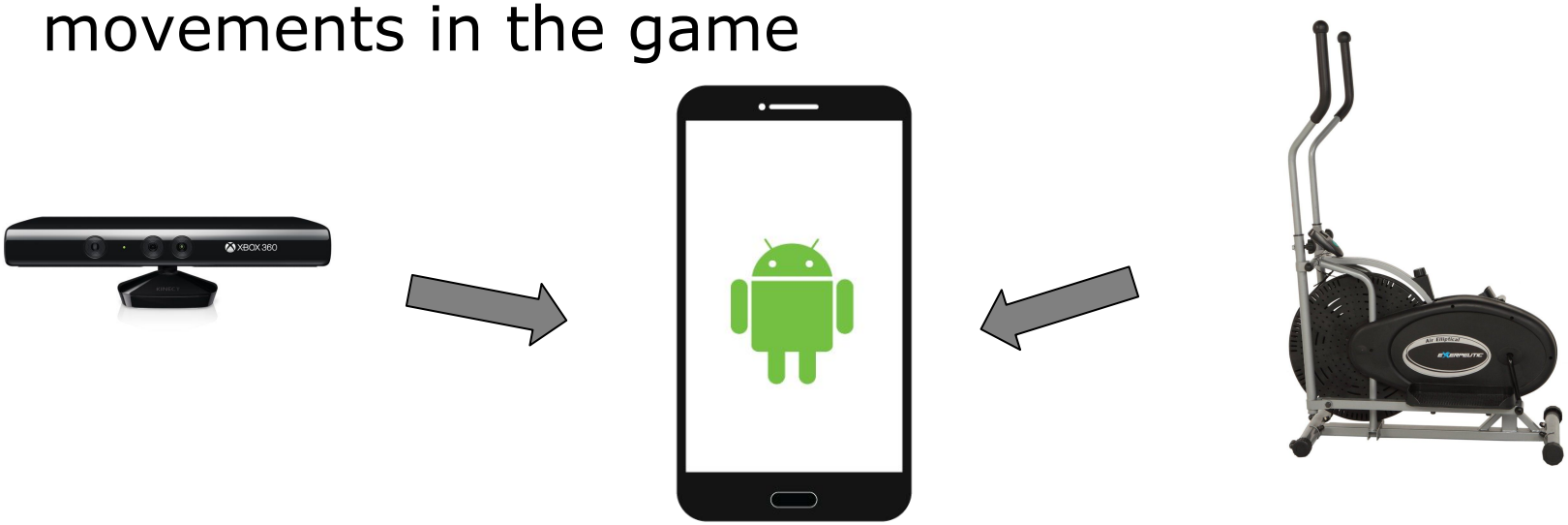


Wireless Network



What it needs?

- Should allow the data from inputs to communicate with android
- Minimal latency in order to have accurate movements in the game



Wireless Network Blocks

Servers

- The raspberry pi
- PC reading Kinect data

Client

- Android

Router

- Wirelessly connects the Servers to the Client

Does this part work?

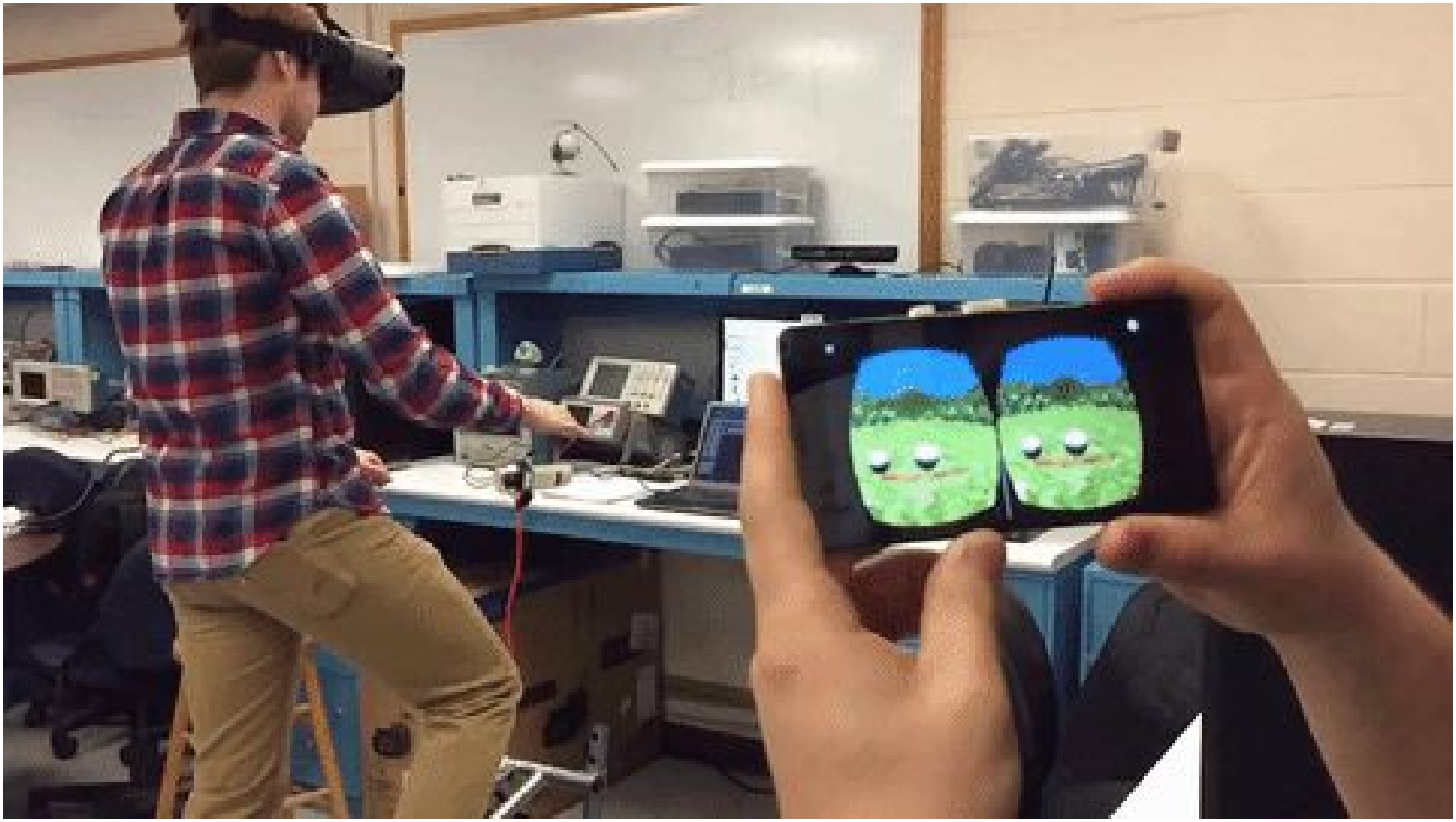
Very clearly the Kinect and the Elliptical can communicate with the android phone through the router since the phone can see

The latency

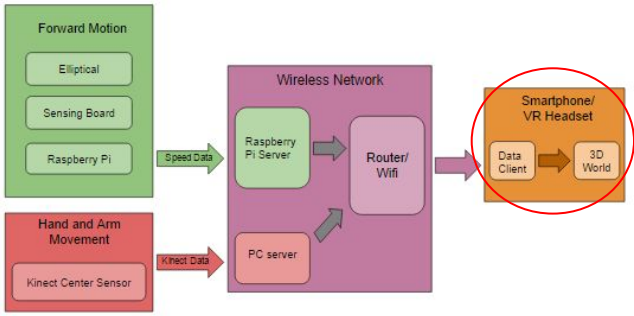
Used slow motion camera and matlab to calculate

End-to-End Latency: ~163ms

Latency : end-to-end



Smartphone Application

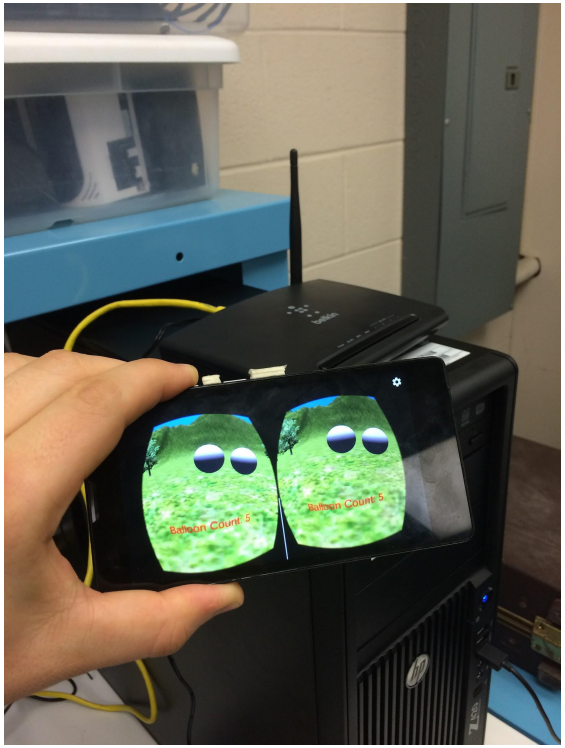


Requirements

- Render a 3D virtual world
- Receive and translate data sent through network
- Framerate = 60 FPS (Limited by VSYNC)

Implementation

- Virtual 3D environment developed with Unity
  - C# scripting
- TCP client requesting data through router
  - Data parsed to be usable
- Ensure TCP servers are not capping the framerate through slow data availability



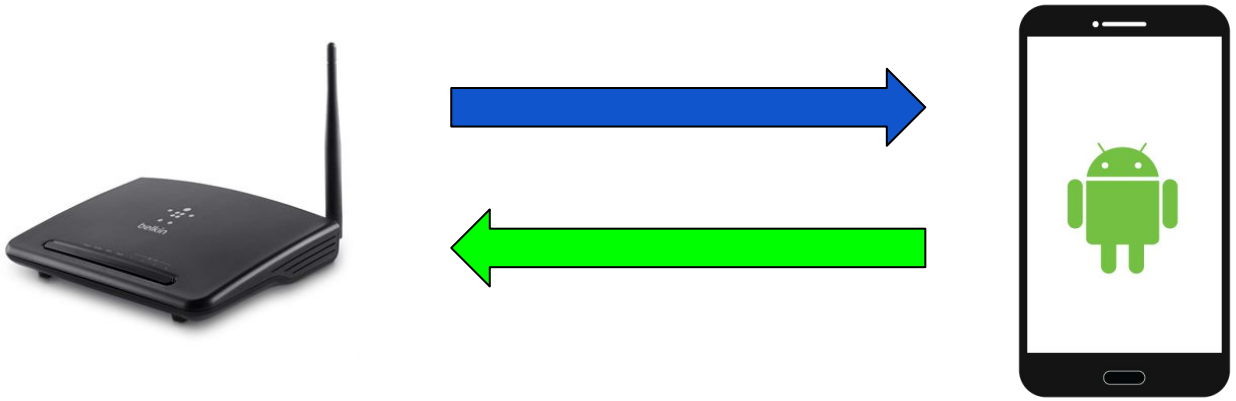
Unity


Unity Personal (64bit) - FirstScene.unity - Balloon VR - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component GoogleVR Mobile Input Tools Window Help
Hierarchy
FirstScene
GvrViewerMain
Main Camera
EventSystem
GvrFPSCanvas
Terrain
Directional light
Cube
Cube (1)
Cube (2)
Cube (3)
Cube (4)
Cube (5)
Win Text
Lose Text
Reset Text
Start Text
Body
Scene
Game
Asset Store
Persp
Inspector
TCPclient Import Settings
Imported Object
TCPclient
using UnityEngine;
using System.Collections;
using System;
using System.IO;
using System.Net.Sockets;
using UnityEngine.UI;
public class TCPclient : MonoBehaviour {
//variables for client 1
private Text displayText;
private TcpClient client;
private NetworkStream stream;
private Byte[] data = new Byte[2048];
private String responseData = "1";
private Int32 bytes;
//right hand variables
private String rightHandResponseData = "1";
private Int32 bytesR;
private Byte[] dataR = new Byte[2048];
//variables for writing
private String please = "please send data";
private String please2 = "1";
private Byte[] send = new Byte[2048];
private Byte[] send2 = new Byte[2048];
//variables for client 2
private Text displayText2;
private TcpClient client2;
private NetworkStream stream2;
private Byte[] data2 = new Byte[2048];
private String responseData2 = "0";
private Int32 bytes2;
private Text displayText3;
private Text displayText4;
private int frameCount;
//left hand
private String[] handCoords = new String[3];
public static double xHandCoord;
public static double yHandCoord;
public static double zHandCoord;
private int i;
//right hand
private String[] righthandCoords = new String[3];
public static double xHandCoordR;
public static double yHandCoordR;
public static double zHandCoordR;
//elliptical velocity
Asset Labels
Project
Console
Favorites
Assets
Scripts
TCPclient.cs

TCP Server → TCP Client → Virtual World

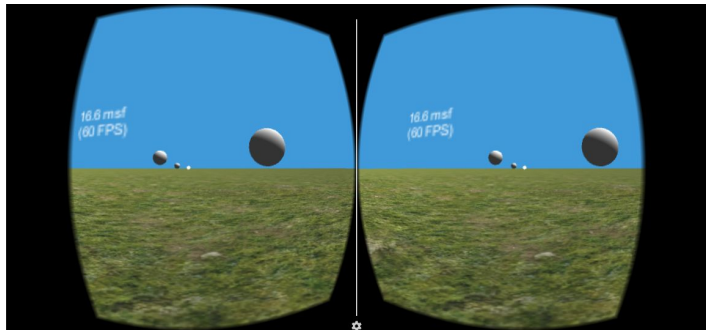
- Uses Microsoft's .NET Framework

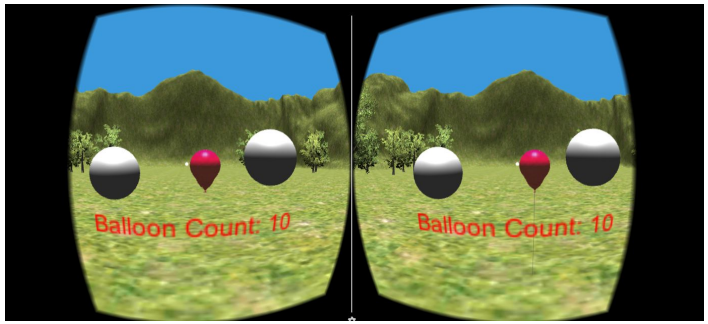
- Steps:
  - Establish client, open stream
  - On every frame render, request data
  - Parse byte data into variables
  - Use variable coordinates and variable speed



MDR Deliverables

Interactive VR game in which the user can walk along a straight path whilst controlling their speed of travel, and using their arms to pop bubbles.





- Demonstrates:
  - ✓ a. precise speed control with elliptical
  - ✓ b. adequate kinect motion sensing
  - ✓ c. data processing and transmitting to android
  - ✓ d. a programmed VR environment using inputs

PDR Feedback:
User has no incentive

→ Game score ✓

Individual Parts

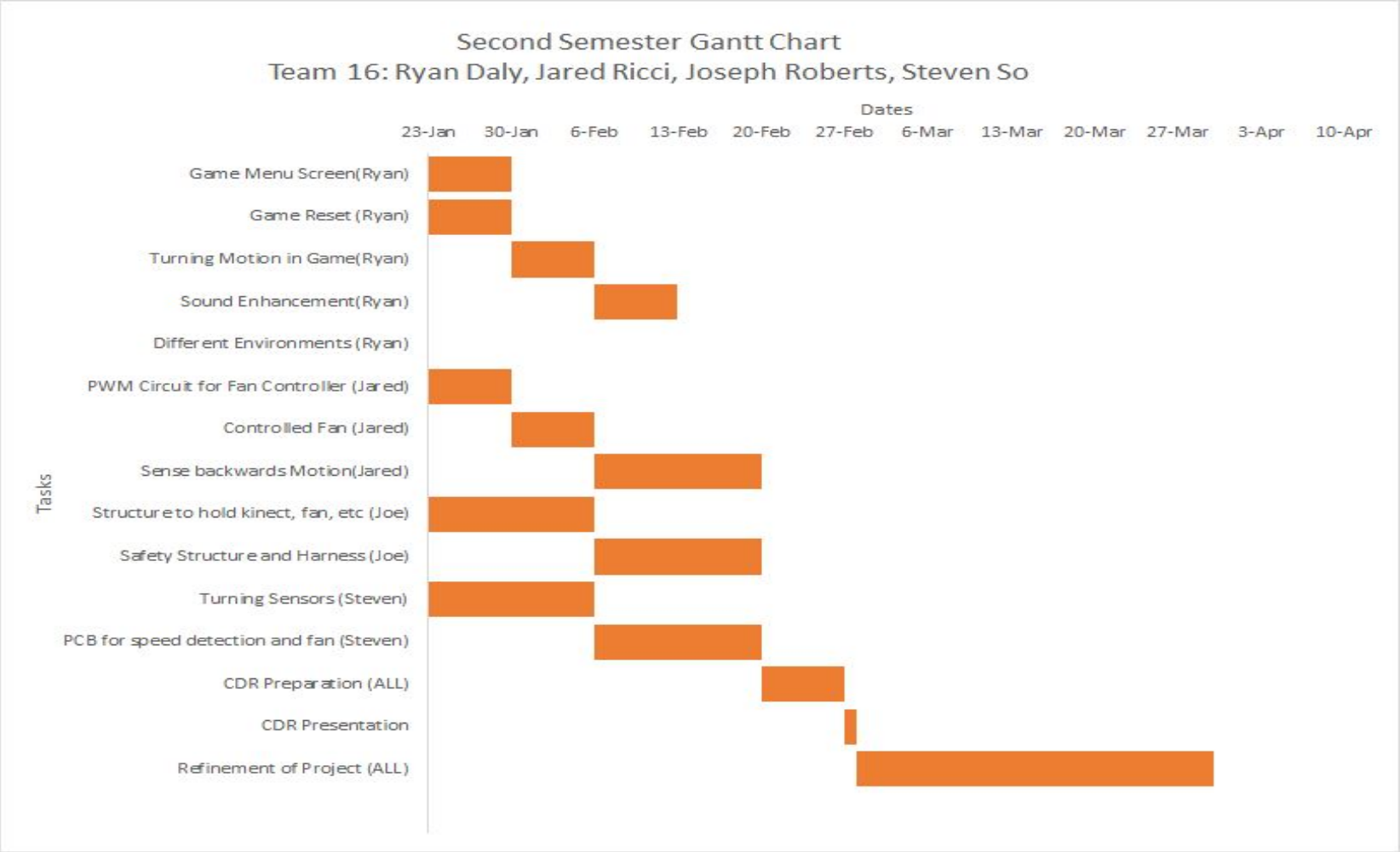
Jared Ricci: ~~Forward Motion~~ ✓ → PCB Design
Fan Controller

Joseph Roberts: ~~Kinect~~ ✓ → Structure
Safety Harness

Steven So: ~~Wireless Network~~ ✓ → Turn sensing

Ryan Daly: ~~VR Application~~ ✓ → Refined Game

Gantt Chart


Second Semester Gantt Chart
Team 16: Ryan Daly, Jared Ricci, Joseph Roberts, Steven So
Dates
23-Jan
30-Jan
6-Feb
13-Feb
20-Feb
27-Feb
6-Mar
13-Mar
20-Mar
27-Mar
3-Apr
10-Apr
Tasks
Game Menu Screen(Ryan)
Game Reset (Ryan)
Turning Motion in Game(Ryan)
Sound Enhancement(Ryan)
Different Environments (Ryan)
PWM Circuit for Fan Controller (Jared)
Controlled Fan (Jared)
Sense backwards Motion(Jared)
Structure to hold kinect, fan, etc (Joe)
Safety Structure and Harness (Joe)
Turning Sensors (Steven)
PCB for speed detection and fan (Steven)
CDR Preparation (ALL)
CDR Presentation
Refinement of Project (ALL)

CDR Deliverables

1. PCB design / Fan control
2. Refined Game (Menu/Reset)
3. Structure / Safety Harness
4. Turn sensing

Questions?