Final Report: Virtual Reality Real-time Robotic Vision System

Ryan Connolly, Dhruv Patel, Sachal Dhillon, Raphael Kim and Guangshen Ma

May 3, 2019

Abstract

This report discusses the design process behind the construction of a real-time vision system in wireless settings. Such a device requires users to perceive their surroundings with head-tracking for a fully immersive experience. It has wide applications that involve performing a remote task because of its intuitive control system. The system integrates a virtual reality (VR) headset, a 360 degree camera, and a 2D camera, which enables operators to visualize panoramic real-world scenes with relatively low overall latency and high field of view. WebRTC, an open source framework to build real-time communication between web browsers and mobile application, is used to establish data streaming between VR headset and the cameras. The hybrid implementation of both a 360 and 2D camera creates a careful balance that mitigates the detriments associated with either individually while improving the overall user experience. The details of the final design, as well as literature review, system performance measurements, and future works are detailed in this report.

1 Introduction

1.1 X-Prize Competition

The company All Nippon Airways (ANA) is hosting a worldwide multi-year competition called the Avatar XPrize, which challenges teams to develop an Avatar System that emphasizes a human's sensation and perception of real time tele-operation. The motivation for this competition is to inspire technology that enables a more connected world with potential applications of the Avatar system including medical care, disaster relief, space exploration, and numerous other utilities that require hands-on expertise.

As such, the Avatar system incorporates technologies that allow a human to see, hear, and interact with an environment as if they were physically there. The project discussed herein focuses on the specific scope of sight for the Avatar by designing and building a robotic vision system to be incorporated into Duke's Avatar submission to the XPrize competition. This vision system will live stream video from cameras onboard the Avatar to a remote user to provide a near real-time view of the Avatar's environment.

1.2 Available Alternatives

Tele-operation, or the remote control of a device, is already widely-used with the help of modern technology. There are already many options to communicate live videos from one endpoint to another via internet protocol. Some of the major services that allow for a one-to-many streaming include Youtube, Facebook, and Twitch, all of which are popular platforms to share one's camera input to remotely located viewers. Unfortunately, these services come with a major disadvantage in latency. These servers must distribute video to multiple clients, which can dramatically increase buffering. Youtube live streaming, for example, has delay on the order of full seconds, which completely dismantles sensation of real-time telepresence. One-to-many streaming services, while easy to set up, are unrealistic for the purpose of this project, which aims to lower overall latency to as close to 200ms as possible in order to achieve real-time vision. One viable solution would require the use of VoIP (voice over internet protocol) services such as Skype. However, commercially available services mainly specialize in enabling a two-way video chat, which is unnecessary because only the avatar would be sending a video to the client. To reduce latency as much as possible, we decided to develop a proprietary application to relay data from a local server directly rather than load-balancing through an existing streaming service.

1.3 VR as Display Device

As implied by the project name, choosing the right type of display for this project was essential. We considered using a 2D display with several monitors, which would be easier to implement while also reducing overall latency of the final product with faster rendering time. However, it was decided that the disadvantage of using a 2D display outweighs the benefits. A 2D monitor, while generally cheaper to obtain, would have a very limited field of vision. Overcoming this obstacle requires the use of a large monitor or multiple monitors, which have considerable implications for the overall cost. More so, a 2D monitor significantly reduces the immersiveness of the user experience due to the lack of peripheral vision.

Virtual reality (VR) headsets, while ostensibly expensive and difficult to develop on, were our alternative display option. Being a relatively new technology, VR development tools were sparse and had complex dependencies which would make our application difficult to generalize across platforms. However, the advantages of using a VR headset are immensely useful to this project. A stock VR headset comes pre-equipped with inertial measurement units (IMUs) as well as a myriad of other sensors to detect head-orientation. Additionally they provide a more immersing experience than standard monitors by covering the entirety of an operators field of vision. Though a variety of options of VR headsets that were available, the Oculus Go was selected for this project for several pragmatic reasons. It was cost-effective with its relatively cheap pricing, allowed for less restrictive movement due to wirelessness, and was developer-friendly with well established libraries. Although this selection was made with compatibility in mind, these tools were eventually deemed unnecessary with the discovery of A-Frame, which

enabled the app to be cross-compatible with any platform that contain an internet browser.

2 Methods

2.1 Software Architecture

The software for this project (and instructions on how to run it) can be found at:

https://github.com/dhruvkpatel/realtime

Our solution is built upon the Web Stack. We chose this route because it allows for quick and iterative development without the need to flash the VR device for each update. Additionally, it allows our software to be completely platform-independent. Our Robot-side software can run on any OS and most modern browsers. Our display-side software can run on any VR platform as well as mobile devices and computers (with compliant web browsers). Within the Web-Stack, we utilize two key tools:

1. **WebRTC**[1]

   WebRTC is an open-source web platform that can be used to negotiate and sustain real-time video, audio, and data connections between web sites. It is supported by Apple, Google, Microsoft, Mozilla, and Opera. Our platform uses WebRTC to stream video from the camera rig to the VR headset. We chose to use WebRTC for several reasons. Firstly, it allows us to have near-optimal video streaming. WebRTC, in most cases, negotiates a direct web connection between two clients. Thus, no extra latency is added via a data-relay server. WebRTC is also built on UDP for video streams. This means that it prioritizes new data rather than complete date. This is ideal for real-time applications because it ensures that the most recent video frames received are displayed with the lowest possible latency. WebRTC comes with traffic-adjustment features. If network traffic is high, WebRTC will adjust the resolution of the video feed to maintain low latency. Finally, WebRTC allows clients to locate one another in a secure fashion. This means that our project can be easily expanded to directly connect two clients located anywhere in the world - not just in a local network.

2. **A-Frame**[2]

   A-Frame is a web framework that can be used to build Virtual Reality experiences on browsers. It is supported by most VR and non-VR browsers. In our platform, we use A-Frame as our front-end framework to display the video-feeds and other information on the VR device. We chose this platform for two main reasons. First, A-Frame is easy to use. It allows us to create simple graphics like 360 video-spheres and 2D video planes with a relatively small learning curve. Second, as mentioned earlier, it is platform independent.

[1] Learn more: https://webrtc.org/
[2] Learn more: https://aframe.io/

Although our platform uses the Oculus Go as the primary display device, it can theoretically work on any VR device simply by connecting to the internet and going to a web page. We can also use a regular desktop web browser for debugging purposes.

Built using **npm**[3], our software package can be built with its dependencies and run with little extra effort. Our core package runs four **node.js**[4] servers locally on the computer connected to the camera rig. The servers are all initialize in the **"main.js"** file:

1. **Robot Client Web Server**

   This http server returns the Robot client web page to be used for camera selection. The web page should be opened on locally on the device connected to the cameras (and running the server). When the package initializes, the user is prompted to open this web page and can select the proper cameras for the 360 and regular video feed. The robot client web page, once opened, initializes a socket connection to the WebRTC Signaling Server and waits for the Signaling server to connect it to the Display client. Once connected to the Display client, the Robot client opens two WebRTC video channels - one for each video feed. It also opens a data channel to send video meta-data and a data channel to receive the Display device's orientation messages from the Display client (Each time it receives an orientation message, it posts this message to the local Servo Control server.

2. **Display Client Server**

   This http server returns the Display client web page to be used as the main user experience on the VR display device. The display device must be connected on the same local network as the camera rig's device (all four servers). In our instance, we connected both devices to Duke's local area network. If connected properly, the VR device can access the Display client web page through its browser. Note: the display client web page is meant to be opened *after* the robot client web page is opened on the other side. Once the display client is opened on the VR device, it initializes a socket connection to the Signaling server and waits to be connected to the robot client. Once connected, the display client displays the two video feeds from the robot client. It also reads from the display device's orientation and sends this through to the Robot client. The Display client also sets up any other user-experience features, such as mode control and out-of-bounds arrows.

3. **WebRTC Signaling Server**

   This http server is used to connect the Robot and Display clients via WebRTC. Each client makes a connect request at initialization. Once both clients are ready, the Signaling server relays WebRTC configuration information between the two. Throughout the connection, the signaling server also relays ICE[5] candidate information.

[3]Node Package Manager. Learn more: https://www.npmjs.com/

[4]Node JS is a web server platform. Learn more: https://nodejs.org/en/

[5]Learn more: https://developer.mozilla.org/en-US/docs/Web/API/RTCIceCandidate

4. **Servo Control Server**

   This http server handles messages from the locally-run Robot client that contain the Display client's most recent orientation data. From this data, it controls the camera rig's servo motors to move the camera accordingly. Upon initialization, the Servo Control server opens a USB Serial connection with the Arduino micro-controller embedded in the camera rig. Upon getting a new orientation method from the Robot client, the Servo Control server feeds this orientation into a control loop. This controller converts the device orientation into the servo angles needed to accommodate the current orientation. Finally, the server sends the servo angles through its Serial[6] connection.

2.2 Camera rotation system

With the WebRTC platform, the limitation for data streaming makes it difficult to transfer high fidelity 360 panoramic images in realtime. Instead, WebRTC will only allow a low fidelity live streaming for 4K 360 image. We address a novel solution to solve the low fidelity problem by integrating a Servo-Webcam rotation system with the 360 camera. The rotation camera system enables viewpoint adjustment based on the head orientation which can be fed directly from the VR headset. A-frame provides an API to poll the orientation of the Occulus VR headset which is then communicated to the local host system. The system sends commands to the control panel through use of the JavaScript Arduino API. When the camera system adjusts to the orientation based on human head movement, the video that is livestreaming in the VR headset is automatically updated in lock-step.

(a) The pan/tilt camera mount

(b) The Logitech webcam

Figure 1: Rotation camera system

The camera rotation system includes pan and tilt subsystems (SPT200 *PanTilt* Kit, Servo City) with 2DOF rotation. The rotation is controlled by two servo motors (HS-485HB Servo, Servo City) in along the yaw and pitch axes. The whole program was developed on the Arduino Nano panel and C language code was developed to control the yaw and pitch rotations. The camera system can produce high-fidelity images through WebRTC and this served as a viable

[6]More on our Serial API can be found in the source code: https://github.com/dhruvkpatel/realtime/blob/master/src/server/servo-device.js

solution to solve the low-fidelity data streaming problem intrinsic to the 360 camera.

2.3 Camera Mount

A simple stand to hold the cameras in place was designed with a 1"x$\frac{1}{2}$" 80/20 T-slotted aluminum bar. Such a bar allows for devices to be easily mounted and unloaded at any height with great flexibility. Thin strips of aluminum were used as platforms to bolt down the cameras. The 360 camera was mounted at the peak to provide a full line of sight in all directions except downward. The rotational platform for 2D camera was mounted right under 360 camera, because its use will be limited to imitating user's neck movement. Therefore, the vertical stand behind the camera would not be obstructing its vision in a practical sense. A custom 3D printed case to house the electronic circuit was mounted beneath the 2D camera. Finally, an aluminum plate was attached at the base for support, but this is currently a placeholder for the actual avatar if the two devices are ever integrated. It is expected that the camera system which will directly support the robot as its "head" and vision subsystem.

3 Results

3.1 Latency Testing

A major constraint on the viability of the proposed system was the ability to limit end-to-end system latency. Current literature suggests that motion sickness becomes onset at a system latency of 70-270 ms [**clemson**]. Although a small amount of overage was deemed acceptable, minimizing total latency was one of the primary directives of the project. Several tests were developed in order to ascertain the various sources of delay inherit to the system.

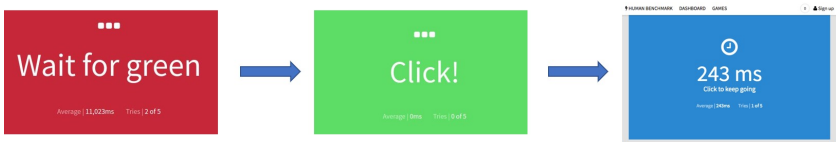

Figure 2: Displayed Test Screens

One of the primary metrics used to measure system latency was the reaction-speed test. This involved displaying the above wait screen to the tester and requesting that they click after a randomized interval. The time it took for the user to click and respond to the prompt was taken as the overall system latency indicated in **Figure 3**.

Figure 3: Reaction times with and without headset

A control was carefully measured by having the user repeatedly complete the test without use of the VR system at all. Another set of trials was also completed with the headset in the full-zoom and 360 views separately. Importantly, the order in which these three trials were done was randomized so as to reduce any confounding variables associated with becoming accustomed to the test.

By subtracting out the control as user-response time as well as camera capture latencies of 51 and 101 ms for the 2-d and 3-d cameras respectively, testing indicated that the overall latency for the system was 149 ms. Network delay accounted for a further 44 ms on average. The aforementioned should not be taken as an end-to-end latency because user response is an intrinsic part of the experience, but it does set an upper bound to the amount of possible improvement. Further considerations present are factors such as image-rendering latency on the headset as well as response delays associated with age.

Figure 4: Practical Usage

In order to ascertain whether or not the above timings translated to real-life performance, testers were asked to drive an RC car around a course in a random order of no-headset, 2d-zoom, and 3d-view.

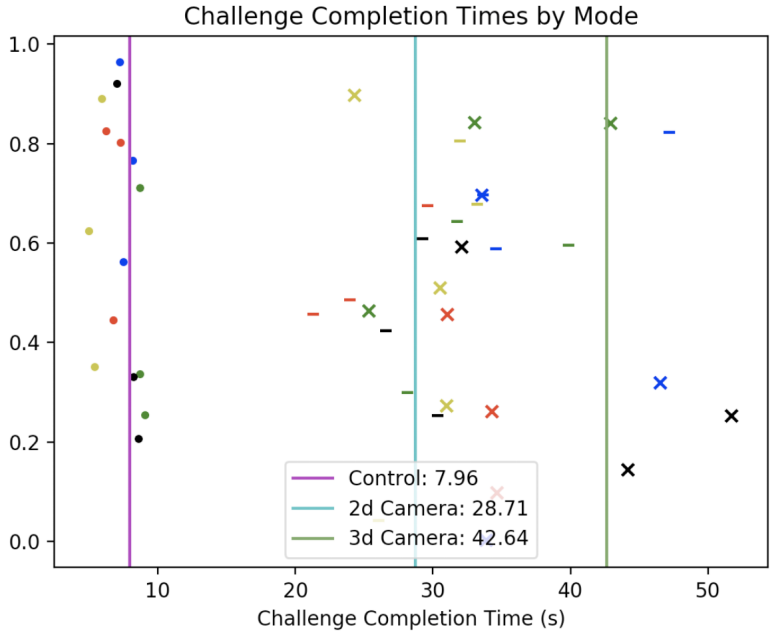

Figure 5: Course Completion Timings

Here, it is apparent that though the use of the headset did have a significant effect on course completion times, it was still usable enough to complete complicated tasks that required careful hand-eye coordination.

3.2 Resolution Test

One of the proposed benefits to adding a high-definition 2D camera to the system was improved field-of-view resolution compared to that provided by the 360 camera. To evaluate the improvement in video clarity, the team performed the Snellen Eye Test using the 360 camera and 2D camera individually. The **Snellen Test**[7] is commonly used to evaluate a person's visual acuity, which is a measure of one's sharpness of vision, and the corresponding chart shown in Figure 6 contains decreasingly sized rows of letters with an associated fraction value next to each row.

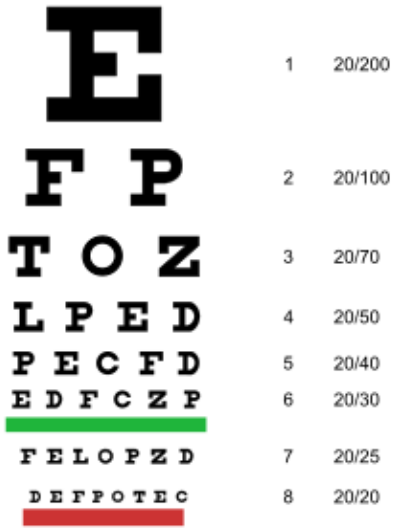

Figure 6: Snellen Chart for Testing Visual Acuity

Standard vision is assigned a baseline fraction value of 20/20(ft), meaning that letters on this row are the smallest size that a person with "20/20" vision can distinguish when standing 20 feet away from the chart. For comparison, a person with "20/20" vision should be able to distinguish the larger sized letters on the 20/80 row at a distance of 80ft. When viewing the 360 camera video within the Oculus Go, the team members could only distinguish the letters located at the maximum chart value of 20/200. The visual acuity improved by over 2x using the 1080p 2D camera view, which allowed users to discern letters in the range of 20/70 to 20/80. While this resolution is still approximately 4x worse than standard vision, the team believes that a better camera could further improve results without modifying any implementation details of the system.

Aside from resolution, the team observed several factors that contributed to the performance of the system's video clarity, notably image brightness and camera jitter. For instance, the testing environment contained several whiteboards with overhead lighting, which was too bright to distinguish any writing on the boards using the 360 camera view. With the 2D camera, however, the image lighting was automatically adjusted when the user faced one of these boards, which

[7] Kaiser PK. Prospective evaluation of visual acuity assessment: a comparison of snellen versus ETDRS charts in clinical practice (an Aos thesis). Trans Am Ophthalmol Soc 2009;107:311–24

allowed for the letters to become more discernible. Overall, selecting a camera with more fine-tuned control of color and focus adjustments could improve system performance. Additionally, camera jitter due small changes in measured head orientation also decreased video clarity, since these oscillating movements would have a blurring effect on the letters being observed. Filtering out these jittery measurements would improve camera stability and consequently improve the usability of the system.

3.3 Distance Test

The ANA Avatar system is intended to be remotely controlled by a human operator at a significant distance from the robot. As such, our team performed tests evaluating the viability of our system at a distance s well as the potential impact on system latency.

First, system latency was measured at the Foundry in Gross Hall, where both the user and the camera host computer were located. Next the user moved to the upstairs lobby of Gross Hall and again performed latency testing. Lastly, the user performed testing in Hudson Hall and in the Edge at Bostock Library. The measured latency and associated distance from the Foundry (camera host location) are displayed in Table 1 below, and an overhead view of the testing locations is illustrated in Figure 7. Note that distances indicated are measured horizontally "as the crow flies", and do not account for any changes in elevation.

Location	Latency (s)	Distance (ft)
Foundry (Camera Host)	407	0
Gross Hall Lobby	391	100
Hudson Hall	492	1500
The Edge at Bostock	480	1900

Table 1: Latency Testing Results at a Distance

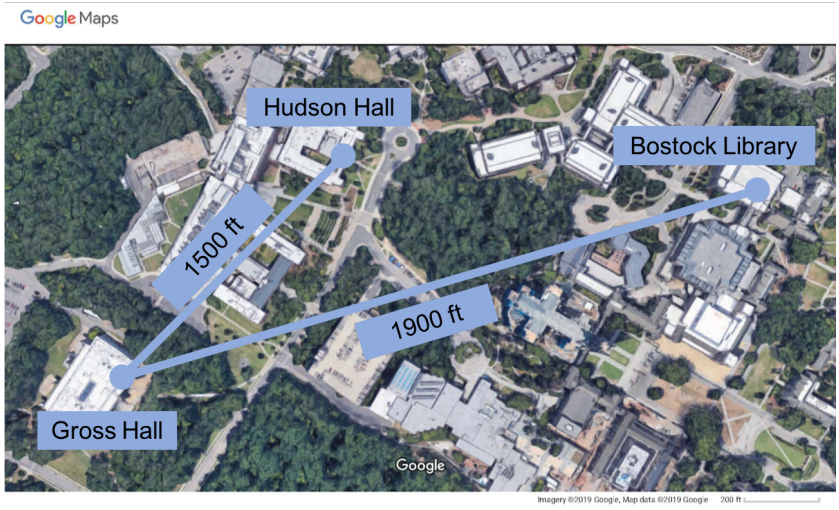

Figure 7: Aerial View of Distance Test Locations

From these measurements, it is observed that there is no statistical difference between the Foundry and Gross Lobby locations and between Hudson Hall and the Edge. Between these two groups of locations, there is an approximate increase of 90ms in system latency, however, due to the unknown behavior of how data is sent across Duke's WiFi network, no direct conclusions can be drawn regarding the effect of distance between the camera host and display on the system's latency. More meaningful results may be obtained if one can measure the distance between WiFi routers that the data is sent between. Furthermore, potential performance improvements could be made if data is sent across a privately hosted network that handles only the communication between components used in the Avatar system, as opposed to a much larger university network with massive amounts of data transfer.

3.4 Performance Testing

In addition to quantitative evaluation of the system, our team performed several qualitative performance tests to determine the ability of a user to interact with their environment while using the vision system. As mentioned previously, users were asked to navigate a small remote-controlled car through an obstacle course. This test was first performed using the original 360 camera system, and later repeated on a new course using the hybrid system. The importance of this test was that the user relied almost entirely on the vision system to sense their environment, with the remote control eliminating any haptic feedback regarding the car's position. Users were able to complete the task using both the 360 and hybrid views of the vision system, with the hybrid system providing a slight performance improvement over 360. This improvement was due to the 2D camera's lower latency and higher resolution, which allowed the user to more quickly correct the vehicle's trajectory and estimate its position relative to obstacles.

The second performance test was to individually place 5 straws into thin vertical PVC tubes on a table, and this test evaluated system usability at a close-range using both hands separately. Haptic feedback was also limited for this test because only the end of the straw would contact the tube and any significant contact with the tube would cause it to fall over resulting in a trial restart. As shown in the image displayed in Figure 8, the user would take a single straw from one of their hands and attempt to place it in one of the tubes. They would repeat this task placing each straw in a tube as quickly as possible without knocking over any tubes. Results comparing the completion times without the system and with each viewing mode are displayed in Figure 9. Users performed significantly better with the hybrid view over the 360-only view, which can be attributed to a few factors. First, is that the improved resolution allowed the user to better discern the entry of the tube and tip of the thin straw. Additionally, the yellow straw proved especially difficult with the 360 view since its color was similar to that of the table, making it hard to distinguish the location of the straw's tip. With the improved resolution and color adjustment of the 2D camera, the user did not experience this issue in the hybrid view.

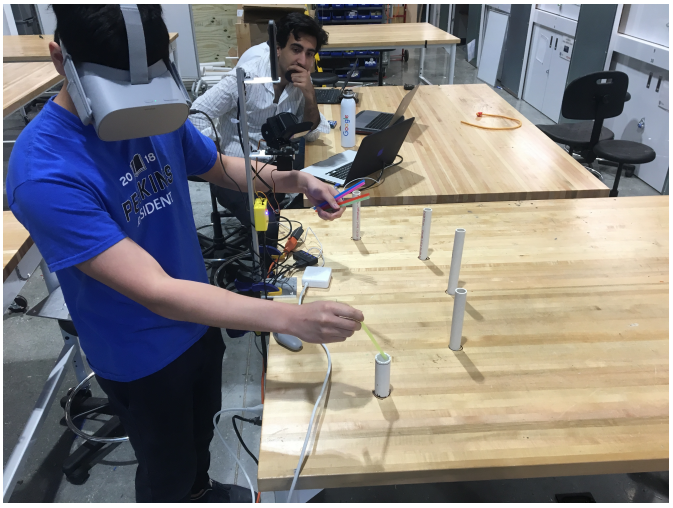

Figure 8: Straw Placement Performance Task

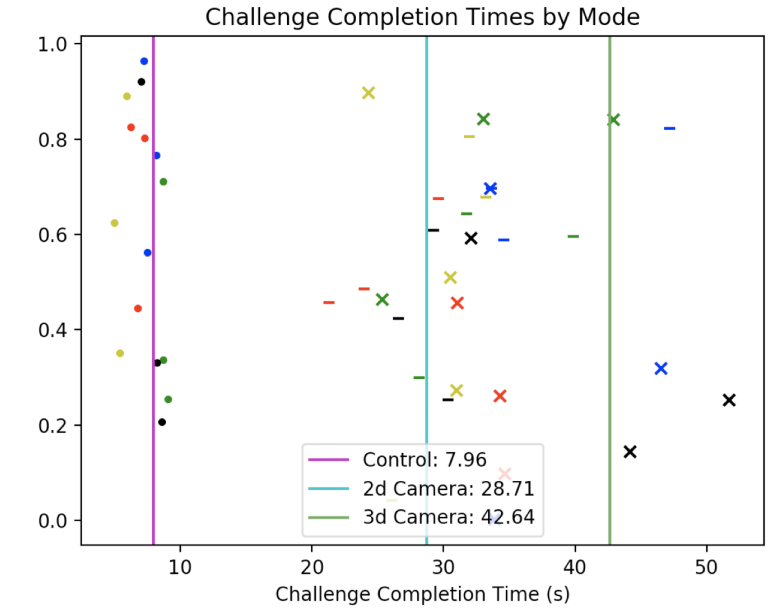

Figure 9: Straw Placement Task Completion Times

4 Discussion

The advantage of our WebRTC-Aframe based software program is the generalization to any operation system and web-browser, i.e we can use this system anywhere with a Wifi or mobile network. This enables a lot of new engineering and research directions related to VR-based realtime remote control. However, one problem of web-browser based live streaming is the limitation for data transmission. In this project, the 360 image is shown with low resolution in the web-browser due to the data streaming limitation, which is the main reason for integrating with a webcam. Realtime data streaming is a task in our future work since it is important for robotic guidance. For example, one interesting question is whether we can transforming the

colorized point cloud data from RGBD camera and show it in the VR headset. This can be easily done in Unity (C) but it is still an open question to finish it with our system.

The system latency measured is about 149 ms which can be used for applications without realtime requirement. For usability testing, this latency is acceptable since all the tasks are not required to finish in realtime. However, the lag during the visualization process may still cause issues when performing tasks that required accurate localization and reflexively quick response to stimuli. It is difficult to make great improvements to the system performance since the latency depends on factors that are difficult to ameliorate, such as Wifi network stability, camera system latency, servo system latency and etc. Possible solutions include a high-speed 360-webcam system that can speed up the live streaming process.

5 Conclusion

We developed a wireless VR-control vision system that enables 360 panoramic and 2D high fidelity visualization in realtime. The software platform is mainly built with WebRTC, A-frame and other functional servers and it enables a platform-dependent feature compatible with any operation systems and web-browsers. This software platform can be easily applied to other VR-based developments and Robotic applications.

Different evaluation tests are conducted in this project. The resolution test validates the importance of integrating a webcam to a 360 camera to generate a high-fidelity image view in a 360 scene. In addition, the latency of our system is not sensitive to the distance between the camera system and the VR operator. This has further applications for remote communication using 4G/5G mobile network. The system latency is about 149 ms and it can be used for multiple near-realtime applications in Robotics. However, it is not satisfied for realtime application and future work will focus on reducing the system latency.

References

[1] https://tigerprints.clemson.edu/cgi/viewcontent.cgi?article=2689context=all_dissertations