**EGRE 429 Team: ECE 409**

**Final Project Report**



**Modular Wireless Xbox Kinect Data Transfer Device**

**Date: May 10, 2018**

**Advisor: Yuichi Motai -** [**ymotai@vcu.edu**](mailto:ymotai@vcu.edu)

**Names: Dat Doan, Millad Nooristani,**

**Eric Olson, Ayush Rohatgi**

**Signatures**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Table of Contents

**Problem Statement2**

**Overview of the Concept System4**

**Major Subsystems12**

**Integration13**

**Implementation Plan15**

**Project Management16**

**Testing, Final Results20**

**Appendix A26**

**Appendix B30**

# 

**1. Problem Statement**

The purpose Modular Wireless Xbox Kinect Data Transfer Device, shorted MWXKDTD, is to counter hardware obstacles faced when attempting to operate multiple Xbox Kinects on a Windows device.

        The idea for the project originated from a computer vision project hosted by Dr. Motai two years ago. The core of the research was using three Xbox Kinects with a Windows PC in which the team encountered hardware difficulties when trying to plug in each Kinect into the same PCI-E card. They deduced that each Xbox Kinect required its own PCI-E card for multiple reasons. First, each Kinect draws enough power to only allow for one device to operate on the card. Second, each Kinect required a unique serial ID, therefore, it was unable to share the same serial bus as another Kinect. Lastly, a commercial motherboard for regular PC’s usually allocate anywhere from two to four PCI-E slots, so a limitation presents itself in the number of Kinects that can physically be connected to the PC.

A diagram of the problem is shown below:

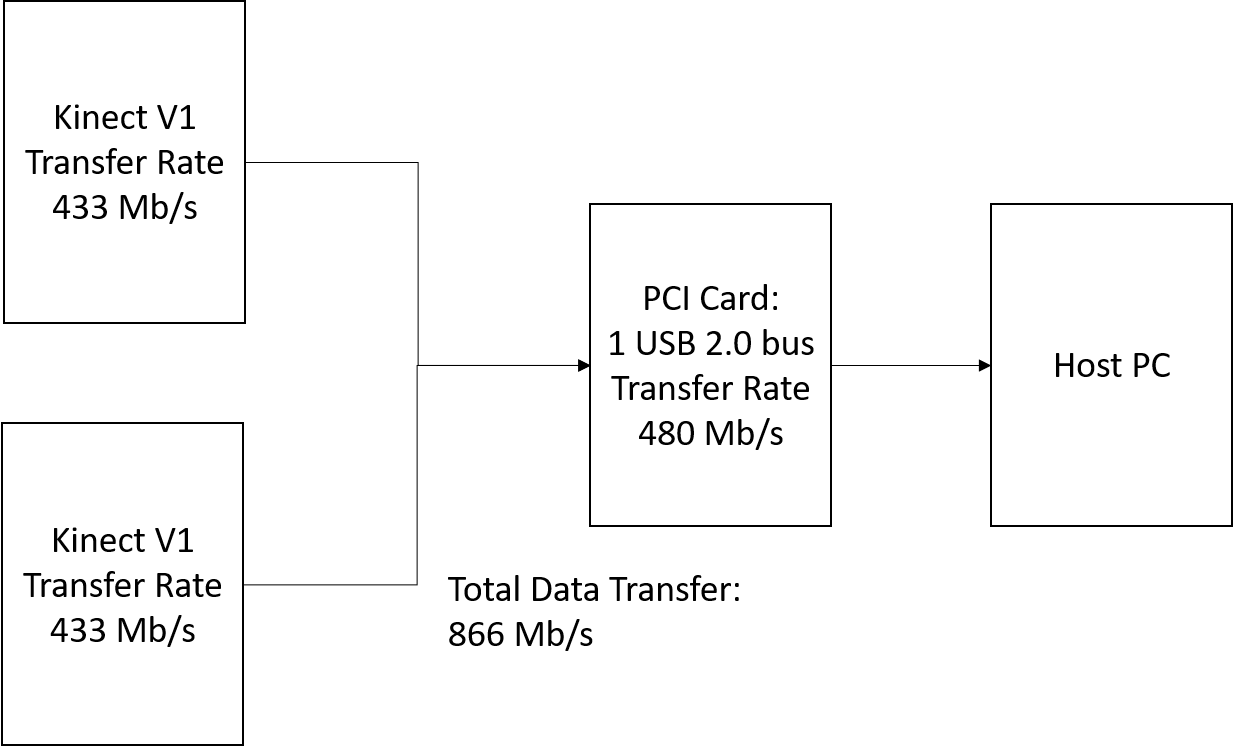


Figure 1: A diagram of the problem that this project aims to address. As can be seen, even two Kinects on the same USB 2.0 bus will overload the maximum data transfer rate. The system cannot operate in this way.

In simplified form, we are presented with the following complications when utilizing a Kinect on a PC:

•       Installation of PCI card per Kinect device

•       Motherboard may not have multiple PCI card slots

•       Each Kinect device requires unique serial ID

•       Each Kinect utilizes full power capacity of PCI card

A diagram of the proposed solution in the past to the project problem is shown below:

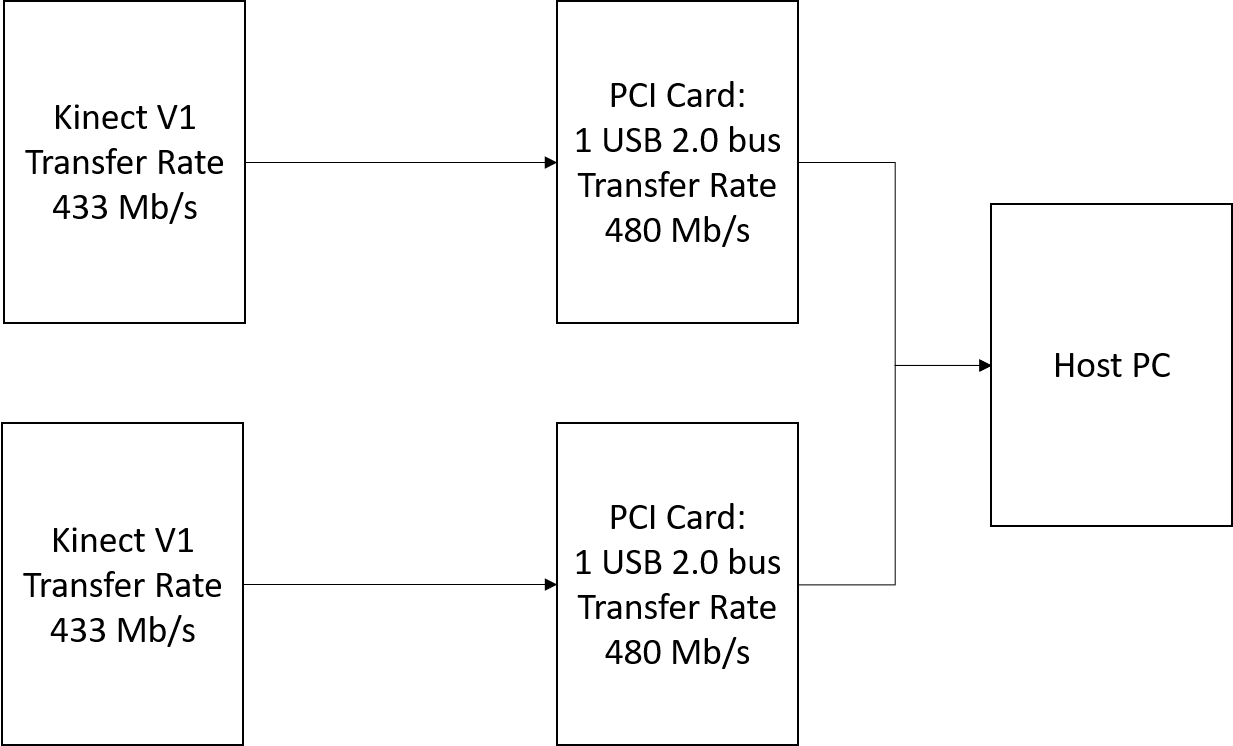


Figure 2: Proposed solution for using multiple kinects on the same PC in the past.

This flows into our customer Requirements: Humans will interact with our product purely as a safe and legal manner if the user follows the specifications provided. Some factors that have been considered are: Safety, manufacturing, operation, and maintenance. Safety: the product serves as a relatively lightweight addition to an existing product. The safety concerns are minimal and are similar to other small electronics.

Metrics for customer satisfaction: Customer satisfaction will be based off of if the customer found as much value from the product that they perceived during purchase. The value that the user derives will be based on what use the user finds. The potential uses of this product are too many to count, and new uses are sure to develop. This is due to the fact that any previously existing use of the Kinect 1.0 with Windows 10 will now be able to be completed using Wi-Fi. As an example, the most obvious use of this product is to use Wi-Fi Kinect 1.0 cameras to setup a point cloud for 3D mapping buildings/rooms.

What should the product do: This project seeks to alleviate issues a user may encounter when trying to set up multiple Microsoft Kinects with their PC. Instead of having to install multiple USB hub PCI cards in each computer they want to use, a user would rather have an easy, modular solution. In our project, we virtualized a Microsoft Kinect on a host PC by transmitting its data from a Raspberry Pi. The Kinect is connected to the Raspberry Pi, and the data is received through an API. The Pi will then transmit the data to the host computer via Wi-Fi.

Performance metrics: Video quality, connection range, data compression and decompression speed, GUI quality and software responsiveness.

In short, our project allows a user to utilize multiple kinects on the same system without the limitations listed above. Our project also has the added benefit of being wireless, adding additional functionality to the project.

**2. Overview of the Concept System**

The system that we created fully addresses the problems listed in the above section. Our project allows a user to have a system of multiple kinects without the downsides of past multi-kinect systems. We designed our system to subvert these disadvantages by using wireless communications instead of using the Kinect’s standard USB input.

The goals we seek to accomplish from our research given these problems are as follows:

•       Develop software to interpret data from Raspberry Pi

•       Adapt existing Raspberry Pi Kinect API for wireless data transmission

•        Implement software to virtualize Kinect video

•       Test functionality by operating our solution with an existing computer vision project that uses multiple Kinects

        We created a modular, wireless device that alleviates those issues. By having each Kinect communicate with a Raspberry Pi over a serial connection, then having the Raspberry Pi communicate to a local Wi-Fi network wirelessly, which then sends the information to the host PC connected to the same local network.

        The open source software used to make this possible is Robot Operating System and RVIS, a 3D visualizer for the ROS framework. RVIS allows the user to see what the robot is seeing, in our case, a point cloud image was developed. Along with that, we were able to get the functionality of the Kinect itself working, which consisted of the RGB image, depth image, and infrared image. All these images were streamed to the Raspberry Pi to the router Wi-Fi local network to the host PC on the same network, which created a video compiled of nearly 30 frames per second.

Evolution of Final Design Solution:

Our final product was the culmination of two semesters of testing, prototyping, adapting, and debugging. At the beginning of the second semester, we decided to make some radical changes to our design due to unforeseen limitations in our prototype. In the prototype, we sought to use a combination of Bluetooth and the Kinect for Windows SDK to realize our product. Unfortunately, Bluetooth had a transfer rate that was below our tolerance level, and the Windows SDK was locked down behind proprietary code. These original specifications restricted us from moving forward, until we were able to come up with not only suitable, but better alternatives. We swapped Bluetooth for TCP/IP based communications, and we swapped the Kinect for Windows SDK with the ROS Kinetic Kinect SDK.  
 Our design has always revolved the idea of modularity. In our original iteration, we associated this modularity with portability. This portability meant creating a solution which minimized wires, had as few pass through devices as possible, and limited hardware. The idea behind this was as the need for using multiple 3D Sensors (specifically Kinects) arise, the setup process becomes much more tethered to a single location. This is due to the sensors requiring additional hardware (PCI/e cards) to be placed inside the host system so that the full bandwidth of data can be passed through. This is only feasible with a desktop computer, and is a costly solution. We wanted to be able to run projects that required multiple sensors on a laptop, or other desktops, without needing to move the hardware out from host machine to host machine.  
 Our very first design was a hardware based rate limiter which would take in the input of three or more kinects into a device. We would then code some software for the device that would filter data out from in the input stream, and send the combined feed to the host machine. Some noise would have been added to the stream, and used to initialize three virtual kinects on the host. This design was deemed unsuitable by Dr. Elks. It required an advanced knowledge of hardware design, and in hindsight, would not have worked with the final tools (ROS, Linux) that we ended up using.  
 Our second design carried on the general idea of the first design, but approached it in a radically different way. We wanted to keep the idea of the host machine doing all the processing, but realized that the data needed to be transferred from the kinects in a different way. In keeping with the idea of modularity, we decided that the information should be sent wirelessly, instead of through a wired hub. In lieu of a direct connection to the host computer, the Kinect would have to be connected to some microcontroller which could process the data. With wireless, there exist two main options: Bluetooth and TCP/IP. We went with Bluetooth, as Bluetooth doesn’t require any intermediary devices (I.E. Wireless Router), and the microcontroller we selected to do the processing (the Raspberry Pi) has Bluetooth built in. Once this specification was decided, we had to decide how we would receive the information. As this project was supposed to be an extension of a previous Kinect project (which was completed and implemented using the Kinect for Windows SDK), we decided to stick with Windows. As with the previous prototype, this meant we needed to reverse the Kinect for Windows SDK to learn how to create a virtual Kinect.  
 As the semester progressed, we ran into issues with the Bluetooth protocol, specifically the interaction between the Raspberry Pi and a Windows host machine. The Windows host machine viewed the Raspberry Pi as an audio device, and was only able to control the volume. Though we were able to get a single file transfer initiated, we were not able to replicate the behavior continuously. We made the decision to change the protocol from Bluetooth to TCP/IP as we were able to easily initiate Wi-Fi transfer between the Raspberry Pi’s and the host computer. Additionally, Wi-Fi afforded us the opportunity to increase the bandwidth from a maximum of 10Mb/s to 434Mb/s. This negated the need to rate limit any of the streams from the Raspberry Pi’s. Switching to Wi-Fi was the catalyst we needed to change other features of the project.  
 Upon switching to Wi-Fi, we began to notice the problems with the Kinect for Windows SDK. Our initial design relied on being able to send image data in real time, process it using a custom software, and then pass that to the virtual Kinect instance. This idea had to be scrapped as we were unable to create the virtualized Kinect. Virtualizing the kinect would have required the reverse engineering of the Kinect SDK. As with the original hardware solution, this implementation required an advanced knowledge of programming that was out of our expertise. Additionally, the SDK is not publically available to edit, so we were unable to access the code. This mandated the change of SDK from the Windows default to the ROS (Robot Operating System) Kinetic SDK. This SDK is intended for Linux based OS’s, so we had to change the operating system we used from Windows to Linux (E.G. Ubuntu). Alongside this change  came a change with the software used to display the images. We switched to a combination of RTABMAP and RVIZ (both of which can be found in ROS). This combination of software lets us view the depth data, infrared data, and image data in one consistent view.  
 Finally, after some input from our professor, we were able to settle on our final design. The final design solution consists of 3 Raspberry Pi’s, each running Ubuntu Mate 16.04 and paired with a 802.11 WiFI AC with MU-MIMO adapter. The Raspberry Pi’s connected to a AC3150 Wireless Router, which was connected via Gigabit ethernet to a Host Computer. The host computer was running Ubuntu 16.04. Both the Pi’s and the host computer had ROS Kinetic installed. A Microsoft Kinect V1 was connected to each of the Raspberry Pi’s. ROS was used to process the data on the Raspberry Pi side, and then was used to interpret the data on the host side. The information was sent over a 802.11ac WiFi connection (through the wireless router). The Pi’s and the Kinect’s were mounted onto a Pioneer 3 mobile robot. On the robot, a monitor was mounted and connected to our host machine, so that the 3d image could be viewed as the robot was moving. Additionally, a mini computer was located on the robot as its controller. The robot was a late-stage addition, and came about due to a suggestion from our professor. He thought by mounting the project on the robot, we would be able to show a real application of our solution (i.e. 3D scanning).  
   Our final solution was truly a result of the mistakes we learned from our earlier designs. Without first trying the far-fetched and improbable ideas that we originally considered, we may not have been able to appropriately search for alternatives. As indicated, we built up from the ideas of each prototype, and changed key resources and tools as it became clear we could not use them. Our final product delivered on all the promises we originally made. Additionally, due to the way it was designed, the pain points upon which the project could be improved are clear. These improvements are straightforward, and could be easily continued on by another group.

Below is a diagram of the hardware configuration:

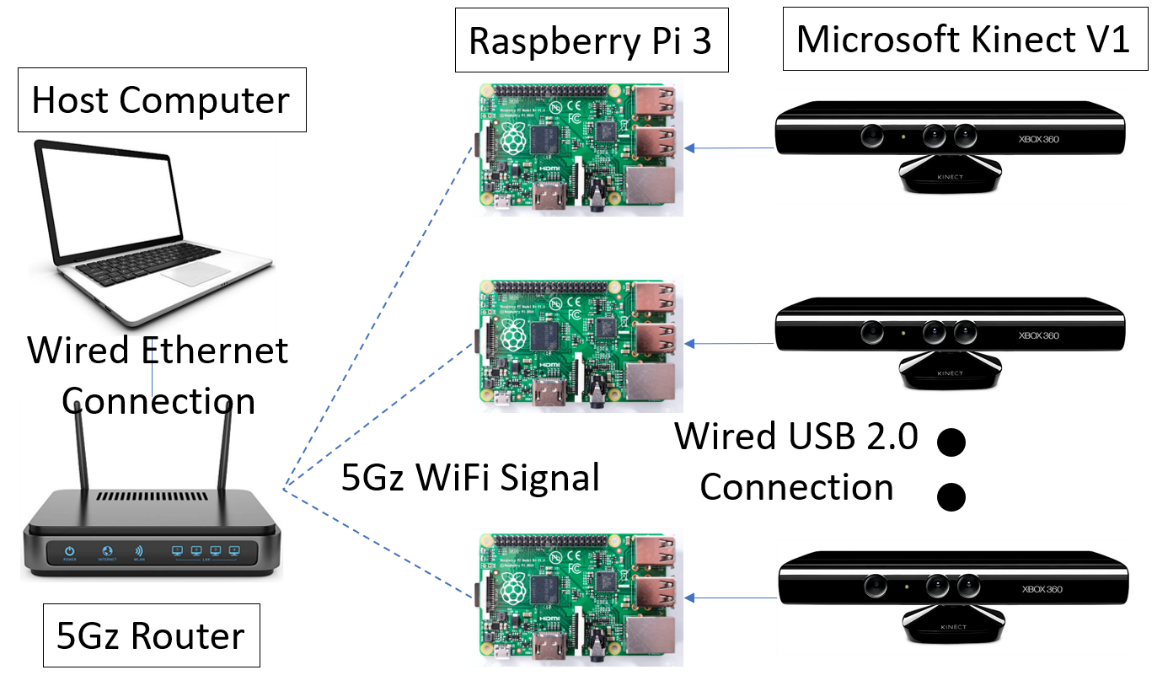


Figure 3: Diagram of the how the different hardware systems are interconnected and through what media/protocol they communicate data through.

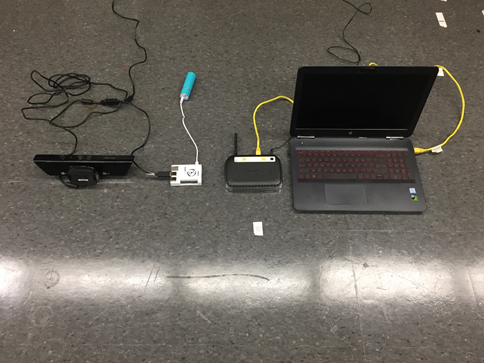


Figure 4: Photograph of an early iteration of the project in actual space. The photo was taken on the floor of lab/workspace. The computer and router were both replaced in the final iteration. The camera and pi share a large power source in the final iteration as well. For ease of understanding, only one camera system is shown in the photo.



Figure 5: Flow chart of the interconnections of the different systems. This flowchart is an overview of the entire system, which encompasses the cameras, Pis, and host computer.

The only part of the final system that has not been discussed in depth in this section is the mobile robot that the camera system is placed upon. The robot subsystem works independently of the rest of the system. More information on the robot is shown below in the Major Subsystems section. A figure of the final project all assembled is shown below:

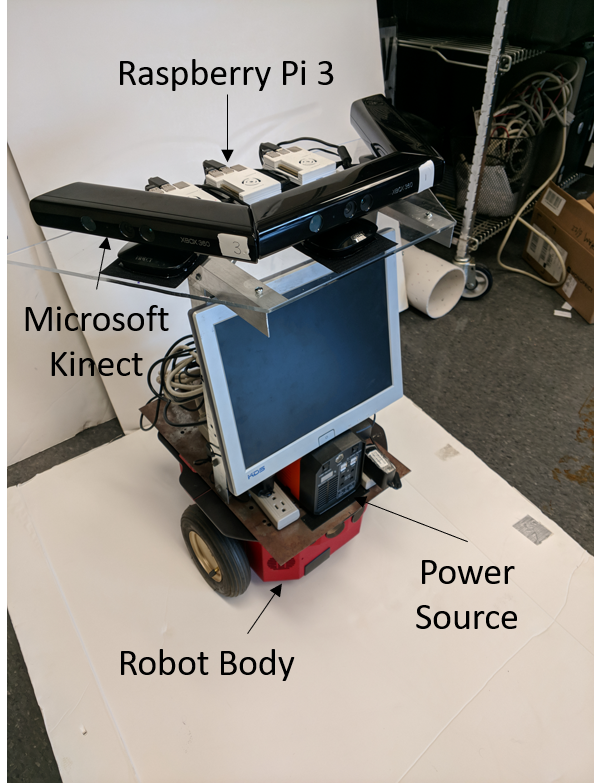


Figure 6: Photo of the entire assembled system. Not pictured is the Host PC and router, which with the photographed robot make up all aspects of the project.

**3. Major Subsystems**

        Our system incorporated 5 major subsystems: The Xbox Kinect, Raspberry Pi 3, Router, Host PC, Pioneer 3 robot. The components were interfaced together via each Kinect wired to a Raspberry Pi, each Raspberry Pi wirelessly connected to a router via the local Wi-Fi network, followed by the router wirelessly transferring the image data from each Raspberry Pi to the host PC on the same local Wi-Fi network via Ethernet connection. The images were then stitched together using RVIZ, a subsystem of Robot Operating System. Robot Operating System is the middleware for robot systems, where it provides services, such as RVIZ, designed for hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. The open source RVIZ subsystem developed the point cloud image, RGB video, infrared video, and depth video.

        The Xbox Kinect for Xbox 360 system was used because Microsoft has released a software development kit which allows users to use its functions and write applications on a Windows device. There also exist open source implementations of the Kinect that allow users to directly modify and operate it features. The Kinect device is also inexpensive and abundant, meaning we were able to find software for the device easily. The Kinect connectivity operates with a USB 2.0 port which is why we were limited to using the Raspberry Pi 3 rather than a better microcontroller with higher serial port speeds. The only need for the device is power.

        The Raspberry Pi 3 is a single-board computer with wireless LAN and Bluetooth connectivity. Initially, we wanted to transfer data over a Bluetooth network; however, we quickly learned that the Bluetooth bandwidth is insufficient in terms of data transfer so we decided to implement the LAN capabilities. The device requires a power source and includes a quad core 1.2 GHz CPU, 1 GB of RAM, 40-pin GPIO, 4 USB 2.0 ports, network capabilities, micro SD port, and HDMI input. It is a multipurpose device with many capabilities for a low cost, making it the most suitable device for our research. The only limitation was the 480 mbps data transfer cap we had using USB 2.0 ports, which was due to the outdated technology from the Xbox Kinect V1.

        The router and host PC both are of standard components. The router had wireless speeds of 1000 mbps over a 2.4 GHz bandwidth and 2150 mbps over a 5GHz bandwidth. These specs were more than sufficient as we were limited in our incoming data stream from the USB 2.0 ports. The PC had standard features including a graphics card and a network adapter which allowed for data to be transferred over an Ethernet connection. Both, the PC and router required a power connection.

**4. Integration**

In our early stage, our first iteration was to mount each of the three Xbox Kinects to their own tripod and attach each of the three Raspberry Pi’s to one of the three legs of the tripod. The USB portable charger will also follow the same mounting setup as the Raspberry Pi’s. The Xbox Kinects were mounted to the tripod via a clip-on mount that attaches to the underside of the Xbox Kinect.

The tripod design allowed our team to manually pick a certain direction and was easy to transport/portability. In the previous computer vision project, they utilized this method because it was adjustable. Since their project had facial recognition capabilities, this was the ideal setup as it allowed for height adjustment of the Kinects and easy mobility. For us, along with the previous benefits, since our research is wireless, we will be able to move the device around to different locations. For example, each Kinect can be setup on a grip handheld tripod system to be capable of using the functionality while remaining mobile.



Figure 7: First prototype iteration

Our second iteration was to keep all three Xbox Kinects stationary, while velcroing them on top of a trapezoid plexiglass. The plexiglass was then mounted on two aluminum support beams via screws and bolts and the support beams are attached to a bottom metal plate using L brackets via screws and bolts. The metal plate is velcro on to the Pioneer 3 robot. Each of the three Raspberry Pi’s are inside a 3D printed case, which are also velcro to the plexiglass. We had to opt out of using the USB portable charger and used a heavy duty portable power supply, due to the fact the Raspberry Pi’s weren’t getting enough power. The heavy-duty power supply was placed in between the two aluminum support beams.  We also attached a Ourlink 600mbps AC600 Dual Band USB Wi-Fi dongle to each of the three Raspberry Pi’s. The monitor was velcro onto the heavy duty portable power supply in the front with the microcomputer placed in the rear.

The purpose for switching to the stationary, tabletop setup on a robot was to increase stabilization when using the functionality of the device, while remaining mobile via robot. This opened the door to opportunities such as 180 degree point cloud imaging where the device could develop a point cloud image of whatever was directly ahead of it. This would allow for higher range of marketability and functionality. A user could setup 3 more Kinects on the system in order to develop a 360 degree point cloud image. Imagine that you were renovating a house and you wanted to have a better understanding of the environment or specific room you were working in. The device could move into a room to develop a 360 degree point cloud image of the room in which could become interactive. You would be able to determine where outlets, studs, piping, etc exist to further reinforce the ‘measure twice, cut once’ principle.



Figure 8: Second prototype iteration

**5. Implementation Plan**

As described earlier, the project suffered from some initial setbacks, which can be described as our “high risk areas”. These can be classified as such because they were the core concepts of the project that determined what technologies we were going to use, and what software solutions we were going to build and follow. These areas include:

* Network Technology
* Operating System & Primary SDK

Network Technology is a key part of this project, as it determined how the information is made modular and how it is transferred. We considered two of the popular wireless technologies: Bluetooth and 802.11/Wi-Fi. We initially decided on Bluetooth, as there exist more stacks on how to directly connect to devices (such as between Android and Linux). We tried to work on Bluetooth for a while, but due to OS limitations in how Windows views the Raspberry Pi Bluetooth, as well as difficulty in doing file transfer without making OS calls, we had to scrap it. This decision came towards the end of the first semester, and was especially considered a pain point due to how much time we invested trying to figure it out, and how key it was to the success of the project. The switch to Wi-Fi proved to be a great move, as any device is able to connect. The initial reason we rejected Wi-Fi was due to the requirement of the router, which made the situation a little less modular.

The Operating System was another high risk area, as it determined how we would implement the rest of our software stack. This is directly tied to the SDK as well, as many Windows SDK’s are not compatible with Linux, and many Linux SDK’s are not compatible with Windows. The SDK’s dictated what kind of software we would write, as well as how our Raspberry Pi’s would communicate with the host computer. We initially decided on using the Windows SDK, which requires us to do a lot more software processing on the host side. The issue with this solution was that we were unable to properly implement the virtualization of the Microsoft Kinects which would have been necessary with this approach. This approach proved impossible, as there was no way to open the Windows SDK to make the proper function calls. We switched our solution to a Linux compatible solution, which allowed us to progress and create a final product. Without this switch, we would have been stuck with no working product.

**6. Project Management**

One benefit of our project design is the clear upgrade path that came as a side effect of the tools we used. As described earlier, our project was adapted from the original vision to fit our goals. Rather than adapting our goals, this meant we found alternative solutions, and some of these were lacking in their implementation. The project consisted of a software solution and a hardware solution: implementation of the goal in ROS and the Kinect Sensors by which the solution was achieved. Some upgrades and improvements which are clear because of this include:

* A better sensor, such as an upgrade from the Kinect 1 to the Kinect 2
* A more universal SDK, or a parallel SDK implementation
* A better microcontroller, which offers better processing power
* A better Wi-Fi router

In our initial design, we had planned to use the Microsoft Kinect 2’s. These Kinects offer a higher bandwidth and better features, so we sought to use these as they were also available at a similar price. Unfortunately, the microcontroller used (the Raspberry Pi 3) has no support for USB-3. This meant that we had to use the original Kinects. In a future iteration of this project, the setup can easily be upgraded to use the Kinect 2, as the software we used is meant to be used with any compatible sensor. Furthermore, if another sensor is released which is more affordable and offer similar or better functionality, that could be switched out. One possible solution is the Intel Realsense cameras.

The Raspberry Pi’s are another clearly upgradeable point in the system. We made use of the Raspberry Pi due to the extensive amount of documentation available, and the multiple distros and software that can be used on it. The use of the Raspberry Pi was encouraged by our instructors as well. The Pi’s performed well for most of our tasks, but due to the high amount of data that was actively being sent, the Pi would overheat often. This led to system crashes and visual instability. Though we were able to remedy this quickly, these issues ideally should not have occurred. There are many other microcontrollers available that give better performance and offer on-board cooling, while still retaining ARM and Linux compatibility.

Though we picked a Wi-Fi router which provided fast peak speeds, we were unable to make use of one key feature: Upstream MU-MIMO. This specification will not be available until the release of 802.11ax, which releases this summer 2018. It allows multiples devices to connect to a single router and transmit data without having to worry about signal interference and airtime splitting. This issue caused our Pi’s to manage their bandwidth and randomly prioritize one connection over the other.

In addition to these obvious upgrades, the project can be upgraded to incorporate more functionality. Another project in our lab focused on just tracking the motions of users and identifying them using a neural network. It then used these motions to automate some tasks. This kind of feature tracking is already built into the Kinects suite of capabilities. By merging the Kinect’s motion detection with a neural network, similar functionality of improved motion detection and task automation could be implemented. Additionally, the Kinect offers facial tracking. By tracking single users, the kinect could follow the users and indicate to the robot upon which it is mounted to follow them. This could be especially useful if the user wants to control what is being 3D mapped as being only what they have seen. Finally, the 3D mapping could be uploaded to some cloud server which would directly turn the mapping into a CAD editable model. This model could then be overlayed with different points of interests, like users and objects, which could then be separately identified, classified, and removed.

Economic issues

The Modular Wireless Xbox Kinect Data Transfer Device, shorted MWXKDTD, can be considered in either a global or local manner. Let us compare the design at a global, or higher level. Currently, there exists many hardware solutions for wireless data transfer for the Internet of Things that fall under working group 802.15 of the IEEE: WPAN/Bluetooth, Coexistence, High Rate WPAN, Low Rate WPAN, Mesh Networking, Body Area Networks, Visible Light Communication, Peer Aware Communications, Key Management Protocol, and Layer 2 Routing. Currently, at the higher level, our product, which falls under the 802.11 protocol, utilizes the wireless networking aspect (Wi-Fi). Let us examine why MWXKDTD can be examined as the most cost-effective proof of concept to be released commercially in comparison to the other 802.15 protocols.

Bluetooth, being the driving idea behind our project, deemed itself to be slightly cheaper due to the lack of a Wi-Fi router priced at $150, currently being used. However, the price-performance ratio unanimously eliminated a Bluetooth device as an option due to its deficiency in frame rate, therefore, we cannot justify Bluetooth as a proper comparison to our current approach. Now, let us compare the current model to the remaining 802.15 IEEE protocols. WiMax, being the next viable option for our product, did not fit our criteria in terms of the range of the bandwidth being too wide and the data rate slightly lower than required for our simple proof of concept model.

Furthermore, if our project were to progress to the next stages of development and find an adequate application, WiMax could be feasible option to achieve ranges greater than Wi-Fi capabilities. By deductive reasoning, we can conclude that if WiMax is the nearest alternative option but it not a possible one, thus, eradicating the possibility of the other 802.15 protocols from being an option, as they become exponentially more expensive.  
Financial

        The Modular Wireless Xbox Kinect Data Transfer Device seeks alleviating the cost of implementing multiple Kinects on a PC. The previous computer vision project attempted to develop software for 3 Xbox Kinects running on a PC. To do so, the team needed to purchase a PCI card for each Kinect that connected to the PC. Each Kinect required a unique serial device ID and used the entire power output from the PCI card it was connected to. Also, most motherboards do not have multiple PCI card slots, which meant the user had to purchase a motherboard advanced enough to support multiple PCI card slots. The cost of implementing the previous project accumulated mostly from hardware upgrade requirements rather than the software to actually implement their solution. This triggered the idea to develop our modular solution, eliminating the need for various hardware changes. An estimated cost for hardware changes to implement the previous project can be found in the table figure 1, below.

|  |  |
| --- | --- |
| Hardware | Cost |
| Gigabyte/Asus Motherboard with 3+ PCI | ~$65 |
| PCI Card | ~$25 |
| 500W Power Supply | ~$80 |

Figure 9: Estimated Hardware Cost for Alternative Project

In figure 1 we find most 4-PCI motherboards to be about $65, each PCI card to be $25, and a power supply, we assume 500W, sufficient enough to provide enough power to each PCI card along with other components, $80. The base total cost would run the team $220, assuming the cheapest parts are purchased, not including other miscellaneous hardware upgrades to be compatible with the motherboard such as processor, ram, graphics card. The previous project also had to purchase the Kinects, each Kinect costing $100. A rough final cost estimate could easily run the team $800-$1000, taking into account all possible components.

        The cost of our project was the cost of the each Raspberry Pi, a router, and small miscellaneous components listed in table 2. We do not include the cost of the Kinect because they were already available from the previous project.

|  |  |
| --- | --- |
| Hardware | Cost |
| Raspberry Pi 3 with Heatsink x3 | $40 |
| Router | $150 |

Figure 10: Hardware Cost of MWXKDTD

        We can roughly estimate the total of our project to be $300, including any small, miscellaneous components we had to purchase such as cables. We can confidently say we have cut the cost of implementing the hardware of the computer vision project by at least half by eliminating the need for PCI hardware upgrades and implementing a modular, wireless data transfer solution.

**7. Testing, Final Results**

Performance Testing

The first performance test we performed was a network speed/transfer rate benchmark on each of the three Raspberry Pi’s to test the Ourlink 600mbps AC600 Dual Band USB Wi-Fi dongle when streaming the point clouds. We installed a speed test application to monitor the upload and download bandwidth from each of the Raspberry Pi’s to the master computer. This benchmark resulted in discovering that all three Raspberry Pi’s network channel will cause some collision/interference with one another. The average network transfer rate for all three Raspberry Pi’s was 8-15 Mbps for upload and download.

The second performance test we completed was a second network benchmark test with the same setup, but this time it involved in changing the channel frequency for one of the Raspberry Pi’s to 2.4 GHz instead of 5 GHz and have the other two Raspberry Pi’s on the 5 GHz frequency. This resulted in less collision/interference between all three Raspberry Pi’s and an increase in average network transfer rate from 8-15 Mbps to 10-18 Mbps.

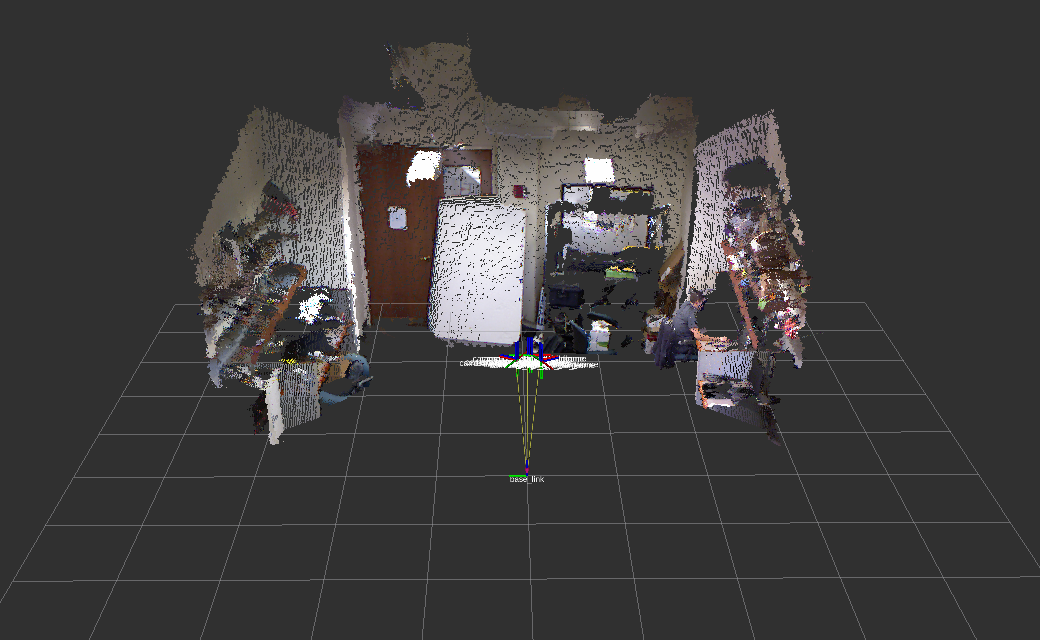


Figure: 3D Point Cloud with one Raspberry Pi on the 2.4 GHz and two Raspberry Pi on the 5 GHz.

The third performance test we performed was to check the response time it took from when a command is sent from the onboard minicomputer and received to the Pioneer 3 robot. This performance test resulted in a 4-millisecond delay. The last performance test we performed was a distance benchmark. This distance benchmark was to check the maximum range our wireless router could communicate with the Pioneer 3 robot. This resulted into a maximum range of 30ft from the wireless router.

Once the unit is assembled i.e. three Raspberry Pi’s each connected to their own Xbox Kinects mounted on top of the Pioneer 3 robot with the onboard minicomputer, we will perform a 3D mapping benchmark. This benchmark will test the resolution of the 3D point cloud, how well the unit can map in low light settings, the duration time for mapping a 10ft x 10ft room, and the maximum time before the Raspberry Pi’s begin to start throttling performance due to overheating.

Outputs of the system:

The following figures show the different outputs that the system was configured to do. The following are shown below: 3-D Point Cloud, RGB Camera, Depth Sensor, IR Sensor, and Facial Recognition.

3-D Point Cloud:



Figure : 3-D point cloud using the wireless Kinect array system

RGB Camera:

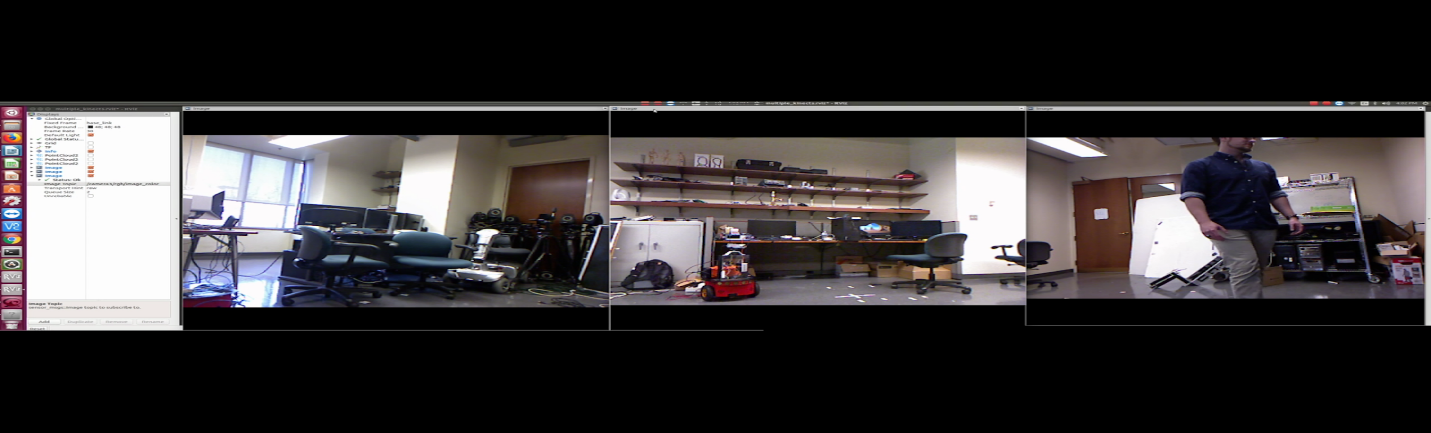


Figure : RGB Camera stream from all three wireless cameras

Depth Sensor Stream:

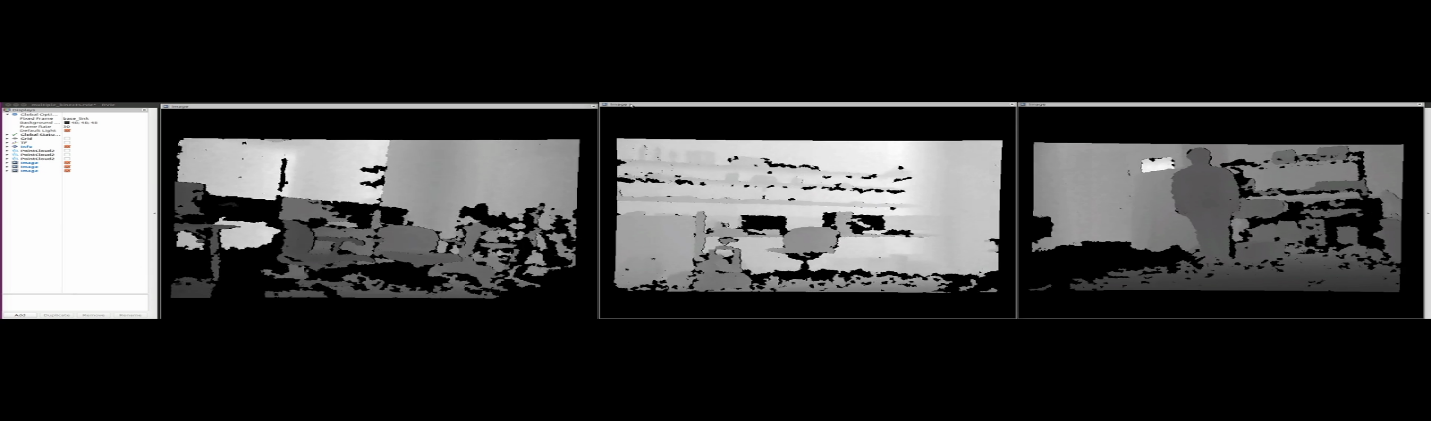


Figure : Depth Sensor Stream from all three wireless cameras

Infrared Sensor Stream:

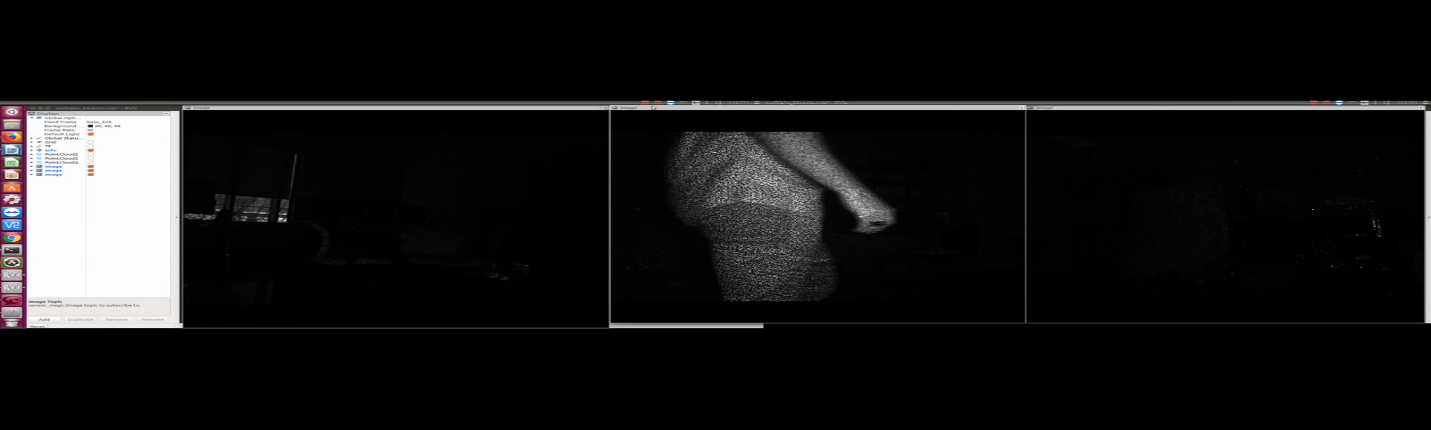


Figure : Infrared Sensor Stream from all three wireless cameras

Facial Recognition:

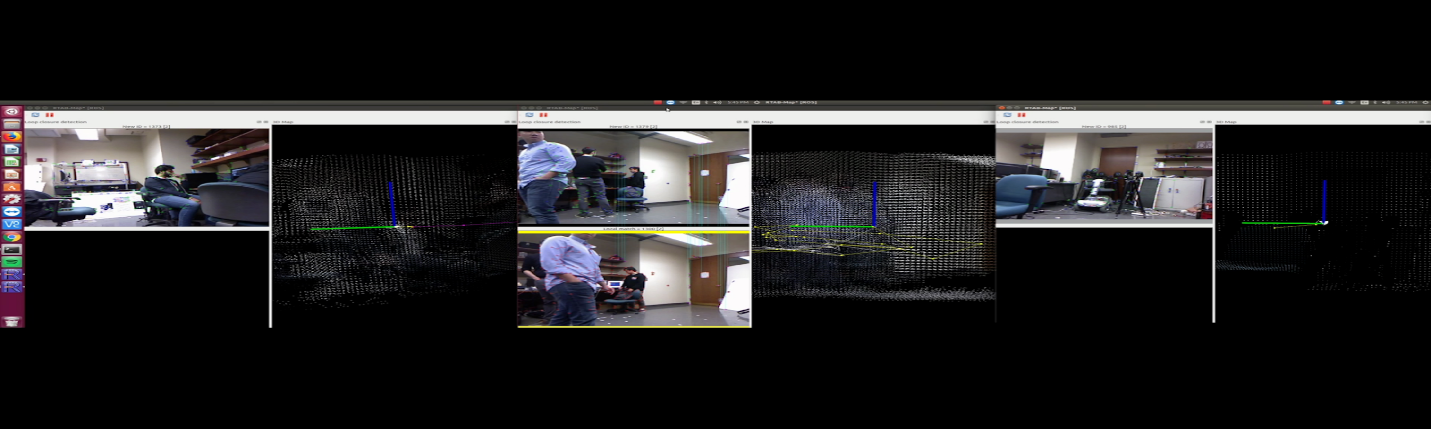


Figure : Facial Recognition software streamed wirelessly through each camera simultaneously

Discussion, Conclusions, Recommendation

The project was a huge success, as we met all of our goals set forth in our Project Description. Our original design of having a wireless Kinect transfer device was conceptualized and brought into and completely through the design and prototyping phase. Using our projected concept of using an array of three Raspberry Pi 3.0s and three Xbox Kinect cameras, we were able to make a system of wireless cameras that would send data through 2.4 and 5 GHz band Wi-Fi to a host computer for data processing.

        The prototype was thoroughly tested with expected results, which can be found in the Performance Testing Section. Our results cover most of the goals we created when the project was started, and the speeds we were able to achieve were greater than expected.

        The goals that we set out to achieve were as follows:

•        Develop software to interpret data from Raspberry Pi

•        Adapt existing Raspberry Pi Kinect API for wireless data transmission

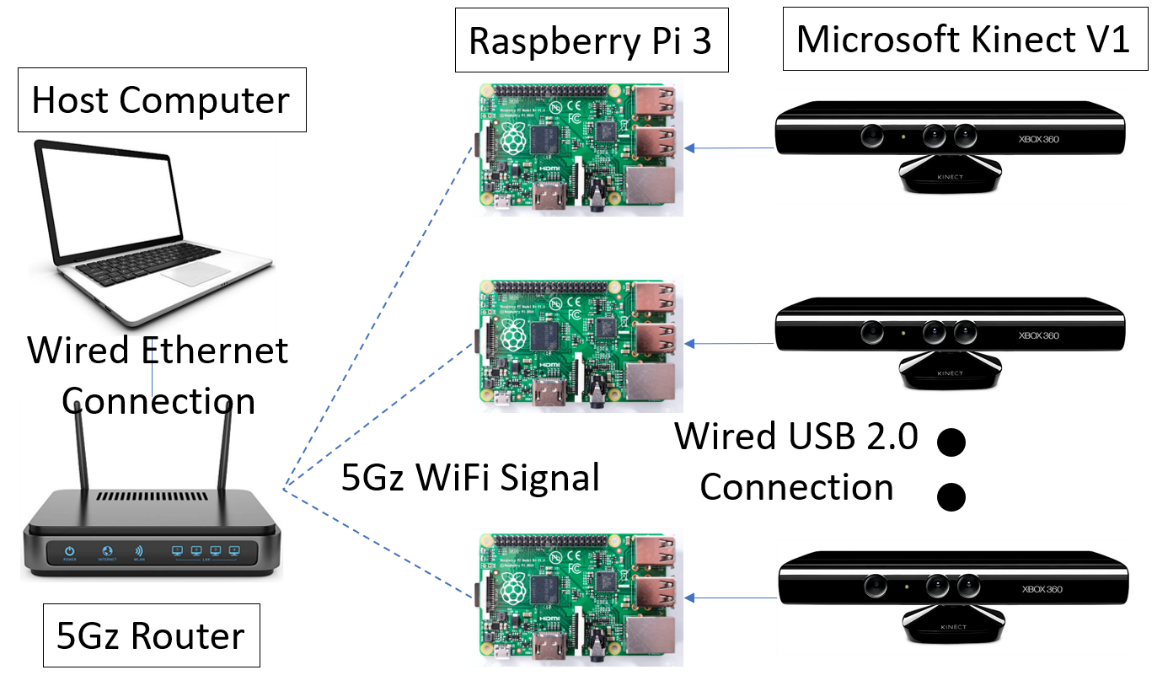
•        Implement software to virtualize Kinect video

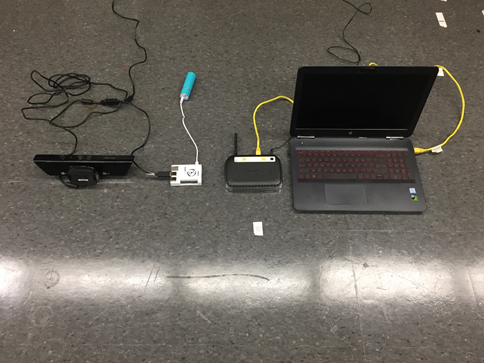
•        Test functionality by operating our solution with an existing computer vision project that uses multiple Kinects

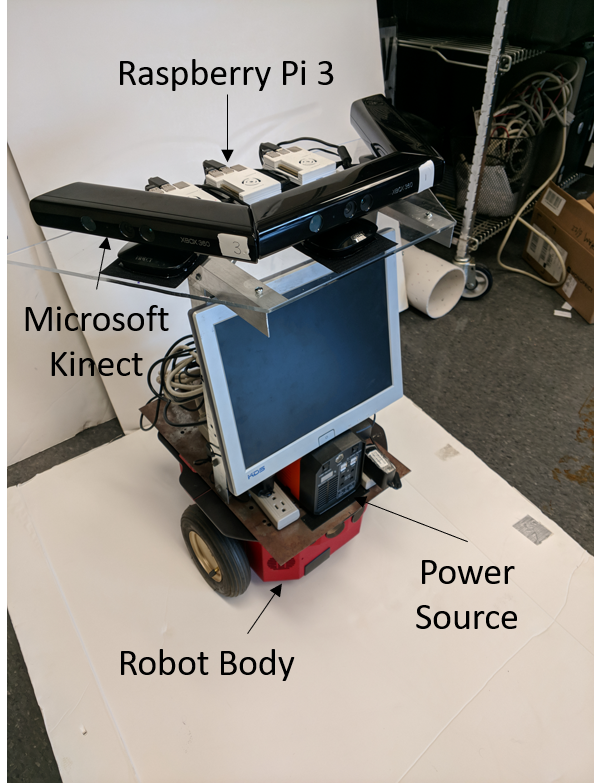
All of these goals were met except for goal number 3. We developed software that would interpret the Kinect data, we adapted an existing API for the Kinect for wireless data transmission, and we were able to recreate the facial recognition project of the year before, as well as some other functionality that was displayed through our many demos. We did not, however, virtualize the Xbox Kinect. While the task sounded simple when we set the goal for ourselves at the project interim, it was a much more complex task than we initially believed. Instead of directly virtualizing the Xbox Kinect, we decided to further modify an existing Xbox Kinect library called libfreenect. This library is made for Linux OS and does not interface with windows.

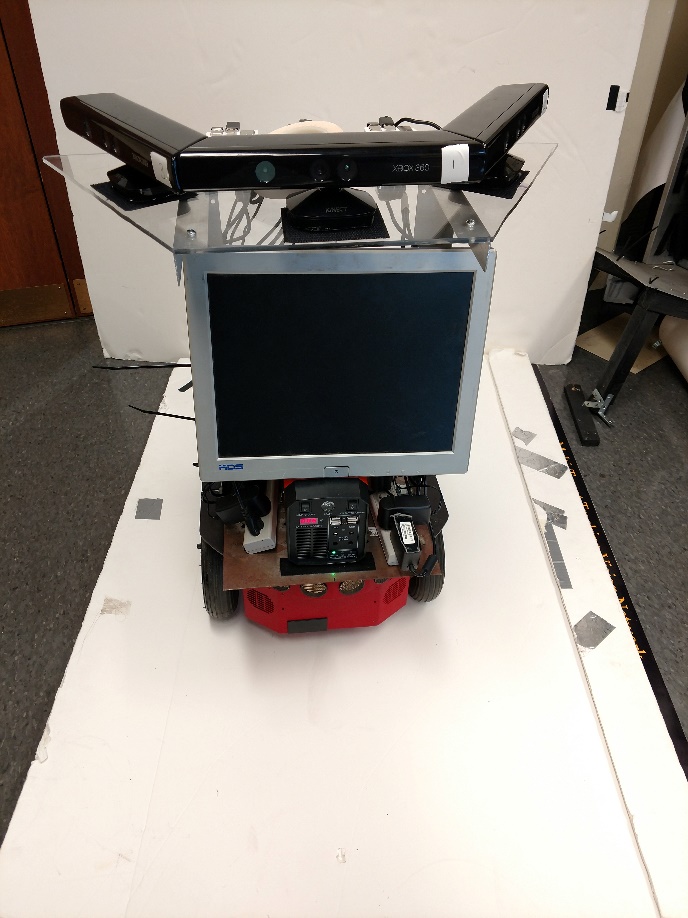
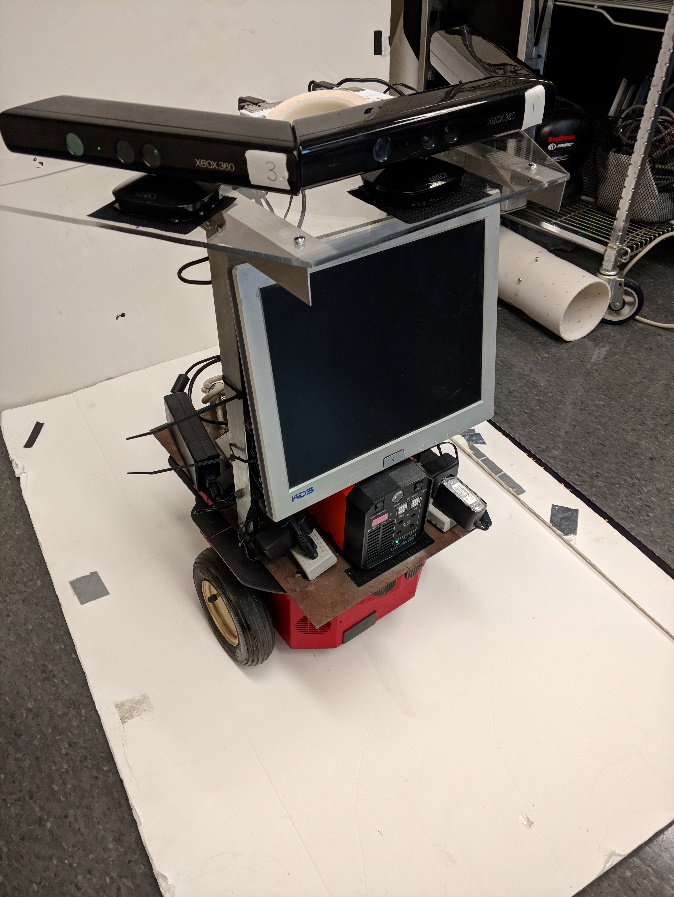
Some recommendations we have for next year are as follows. The most obvious improvement would be to virtualize the Kinect in a much more direct way. This would allow the system to be run with a windows host computer. The next upgrade that could be done would be to increase the data transfer speed with multiple user multiple input multiple output (Multi-user MIMO) upstream. This would help eliminate collision between the data transfer coming from each camera. The technology was not available during our iteration of the project, so we could not implement it. The last recommendation we have is to use a stronger microprocessor in order to support the Kinect V2, which is a superior camera. The raspberry pi 3 cannot support the Kinect V2, as it only has 2.0 USB support.

**Appendix A**









**Appendix B**

https://github.com/rohatgia/VCU\_Capstone\_2017\_2018\_ECE409