6.835 INTELLIGENT MULTIMODAL USER INTERFACES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Andrew Moran
May 15, 2015

VIRTUAL REALITY AND LEAP MOTION FOR ENHANCED AIR TRAFFIC CONTROL (ATC)

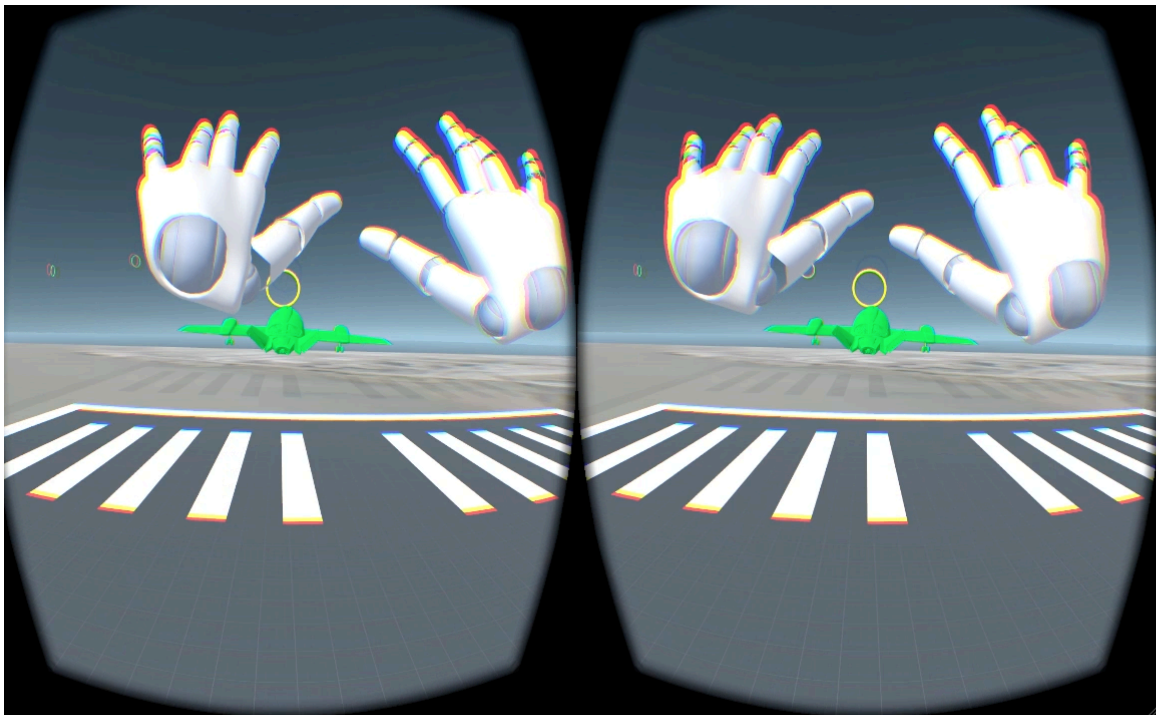


Figure 1: Stereoscopic view of system as seen from the Oculus Rift Head-Mounted-Display (HMD). Above scene depiction is an example of how a player can utilize the Leap Motion controller to navigate a model four-axis airplane in a VR interface.

ABSTRACT

This paper presents a new system that incorporates two emerging technologies (*Oculus Rift* and *Leap Motion*) to generate an immersive 3D scene to promote interaction and situational awareness. As a use case, I created a simulation to better monitor and coordinate model airplanes for local air traffic control. After multiple iterations in the prototype, gesture recognition has improved to be more accurate than expected. Results have shown that users do feel more involved when placed in a virtual geographical scene that can be dynamically manipulated by various sensor inputs. However, further improvements and features on the user interface need to be made to enhance the user experience and expand this system to a more real-world application.

TABLE OF CONTENTS

1. INTRODUCTION

Virtual and Augmented reality (VR/AR) have made generating 3D environments very realistic and visually appealing. There have been many advances in VR game development due to the improved performance of computer graphics and the amount of detail one can achieve when applying varying first person perspectives in a virtual reality setting. [4] and [5] have shown how VR has extended from game applications into other areas of research such as situational awareness and information visualization. These above works have shown how immersion helps scientists more effectively investigate their area of study and lead to better discovery. As a result, I wanted to exercise this concept more in depth by creating a system that helps manage Air Traffic Control (ATC). There have been previous studies that examine how simulating geographical settings for training purposes have increased ATC performance and attentiveness. This project is designed to be a hands-on experience of determining the pros and cons of this sort of research. In addition, I wanted to experiment with natural inputs such as gestures and head orientation. Here, I can test if this input is more intuitive for user exploration and how this impacts the way orders are executed for ATC.

1.1 MOTIVATION & RELATED WORK

As a Research Assistant at MIT Lincoln Laboratory, I have seen many divisions devoted to various projects in Research & Development and integrating new technologies for the Department of Defense. In particular, Air Traffic Control is a large sector in governmental research. I have had the opportunity to see current state-of-the-art applications and the most recent installments. I even had the privilege to talk to retired air traffic controllers. Most practical ATC monitoring relies on 2D displays and any other non-traditional systems are typically designed for training/simulation.

There have been many efforts in incorporating augmented and virtual reality based solutions in a variety of sub-tasks related to air traffic control. Reisman [7] conducted a project during the 2000s to test the utility of Augmented Reality (AR) for tower control. In general, air controllers "strongly agreed" that AR had potential to complement current ATC practices. Santos [6] did research on remote tower control, simulation, and air traffic control training. Advantages of AR tools mentioned were better low-visibility sight, remote capabilities, superposition of data, and more ATC engagement such as tracking functionalities.

As a result, there has been much progress in seeing how VR/AR could apply to ATC. However, some concerns in practical applications include latency, performance, user comfort, etc. which explains why this is still a topic in research rather than practice. Therefore, designing a prototype that fulfills some of the above functionalities will help evaluate the usefulness of this technology and if future integration is practical.

1.2 THE TASK

This system is designed to be a responsive ATC simulation tool as viewed from a remote air traffic location (for this example, San Francisco International). As viewed from the first person point-of-view, the objective is to fulfill the role of an ATC controller and manage/coordinate planes as they populate a local airport terminal. Described in more detail in the evaluation, there are two main modes of this application.

1. **Field Training** – The main role of an air traffic controller is to provide direction and commands to planes in the field from a remote watchtower. By performing gestures, you can signal planes and what course of action they should execute.
2. **Manual Piloting** – Sometimes you would like to know how it's like to control an airplane. Similar to traditional flight simulators, this game mode allows the user to fully control an airplane model on four axes (yaw, pitch, roll, thrust). Therefore, you attempt to execute the complete duration of a plane's flight.

By performing these modes in a VR application, a more realistic and enhanced situational awareness can be achieved which could benefit ATC performance. In the next section, I describe how a VR setting with gesture recognition is implemented.

2. SYSTEM IMPLEMENTATION (HOW IT WORKS)

2.1 CAPABILITIES

This system is a standalone computer application that can run on either Mac OSX or Windows. This application supports Oculus Rift DK2 and Leap Motion as supplemental inputs/outputs. The idea is to replace the standard I/O provided from traditional desktop applications (monitor, mouse, keyboard, etc.) with sensors and displays that translate better in a 3D virtual environment.

Input is provided as sensory data provided from both the Oculus Rift and Leap Motion. To start the simulation, you load the application like any traditional PC video game on your computer. However, instead of watching on your computer monitor, you plug in the HDMI cord to view on the external head-mounted display (HMD) in the Rift. Gyroscopes and sensors in the Rift determine the camera's position and player's orientation in the scene. Now, in order for the player to perform actions in the simulation, the user performs gestures that are recognized by the leap motion and translated to game commands. The default Leap Motion SDK and library provides the recognition of hands and simple gestures once viewed from the infrared camera. To provide feedback on hand and gesture recognition to the user, physical models of hands are rendered in the 3D scene as shown in Figure 3.

Once in the simulation, a player is able to view model planes at an airport. Gestures dictate the course of action a player is capable of communicating in order to manage and coordinate model planes in the scene. Once the gesture is executed, the plane responds and the plane's status is updated accordingly. As the output, the scene dynamically changes as planes carry out routes provided from the user. Tables 1 & 2 show current input and outputs of the system.

Table 1: Field Testing Gesture Input-Output relations

Input	Gesture Description	Result/Output
Selection - Determines which plane is currently selected	Point with finger	Plane highlights to a selected color. All other gestures will be applied to this plane.
Takeoff - Notify plane is free to takeoff	Hand in front, palm face down, fingers facing forward, swipe hand upwards	Plane initiates takeoff, starts thrusters, increases speed, and ascends leaving the airstrip
Roundabout - Direct plane to continue following circle route above	Hand in front, fingers facing upward, rotate hand either clockwise or counter-clockwise in a full circle.	Plane overrides of the map destination to follow pre-defined circuit above the airport
Land - Notify plane is clear to land	Hand in front, palm face down, fingers facing forward, swipe hand downwards	Plane initiates landing, once it approaches suitable entry point to runway, decreases speed and descends
Toggle Camera - Switch between cameras in scene	Hand in front, palm facing user, fingers facing towards other hand, vertical swipe closely over Leap Motion	Toggles user's camera between remote tower on runway and fixed camera attached to plane

Table 2: Manual Piloting Gesture Input-Output relations

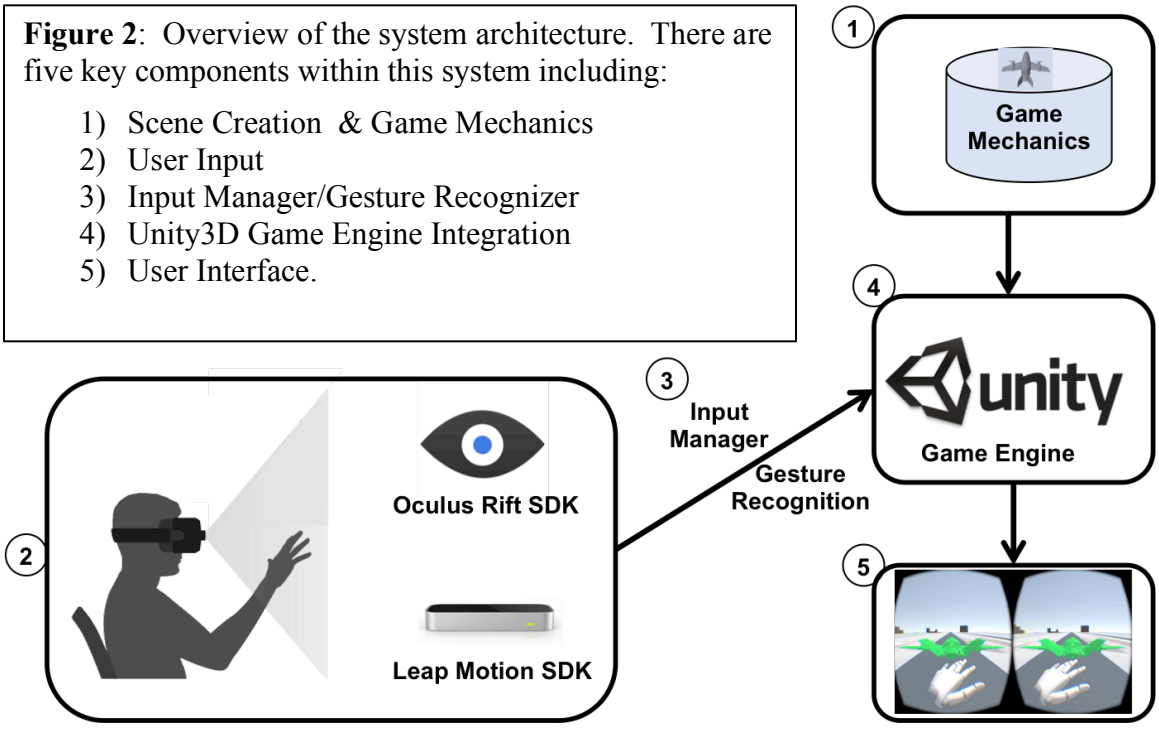
Input	Gesture Description	Result/Output
Enable - Toggle manual override of plane controller	Place both hands in front	Once activated, the plane changes color and AI script is switched to control script, which reads in player input.
Navigation - Adjust plane direction (yaw, roll, pitch)	Vary locations of hands relative to one another to adjust global X,Y,Z axes.	Plane changes direction along the yaw, roll, pitch axis based on orientation of both hands
Thrust - Adjust plane speed	Make both hands into a fist	Plane's speed adjusts to fist recognition. Extended hands match to a full thrust and plane it at full speed

2.2 ARCHITECTURE

This project was designed with the Unity3D game engine. Unity3D is readily available to developers and makes it a valid candidate as a modeling and 3D visualization tool. In this example, I used Unity3D to create a system that recognizes gestures in an Air Traffic Control virtual reality simulation. When designing this project, I incorporated ideas of good user interface design as well as game development to try and appeal to many attributes that define a good system from a first person perspective. This includes efficiency, learnability, user satisfaction, attentiveness, etc. Figure 2. Summarizes the overall layout of the system, which is described more clearly in the following.

Figure 2: Overview of the system architecture. There are five key components within this system including:

1) Scene Creation & Game Mechanics
2) User Input
3) Input Manager/Gesture Recognizer
4) Unity3D Game Engine Integration
5) User Interface.



2.2.1. SCENE CREATION & GAME MECHANICS

One of the key components of this system is the 3D geographical environment that needs to be created for the user. Developing this static scene first required some manual configuration before any further game mechanics could be applied. When doing so, it's important to keep note that Unity3D works as a physics engine. 3D objects can be created and instantiated in the scene, but the physical characteristics you attach to them (rigid bodies, colliders, etc.) helps guide how they are influenced by scene constraints (gravity, proximity, scene perimeter, etc.). For this ATC system, I created a simple scene where there is a 2048 x 2048 PNG image of San-Francisco International Airport (KSFO) supplied from the Google Maps API and rendered on a 2D texture representing the

ground. Next primitive shapes and Google Sketchup 3D models representing buildings populate the scene to give a more 3D feel.

Planes represent the dynamic objects in the scene that are viewed and manipulated by the player. Therefore, game mechanics needed to be applied to the scene to describe how planes are modeled and how gameplay is experienced. In particular, they too are 3D physical objects that populate the static scene described above. APPEDNIX 3 describes in detail more on the layout of the game hierarchy and scripts that I utilized to help control planes and their relationship to the scene.

2.2.2. USER INPUT

As previously mentioned, the application requires a user to place the Oculus Rift on their head. The Leap Motion controller is strapped onto the front of the Oculus Rift to register and any hands that appear in front of the user. Specifications on hand detection and gesture recognition are used to undergo game actions as described in the next section. Sensory data collected from the accelerometer and gyroscope of the rift translates to specifications of the camera's angle, direction, and position in the game environment. Please see APPENDIX SECTION for more information on the input devices.

2.2.3. INPUT MANAGER & GESTURE RECOGNIZER

When reading in data from the sensors and user input, there are two main components that then drive the system.

- **Gesture Recognizer** – The Leap Motion device has information on current frames supplied by the infrared camera. Once a hand is detected, advanced skeletal tracking improves the monitoring of specific attributes pertaining to the hand such as location, direction, finger extension, palm position, etc. As a result, gestures can be detected by looping through a sequence of frames and comparing defining metrics motions such as a swipe or circle for example. The Leap Motion SDK allows these gestures to be created, customized and processed via C# scripts.
- **Input Manager** – Similar to any application and game, there needs to be a global manager to launch events in the scene based on user actions. The input manager serves as a global script that updates and triggers events in the scene once they have been activated by gestures confirmed by the gesture recognizer. This includes camera views, plane selection, and signaling gestures that have previousle been seen in TABLE. Game objects and the scene dynamically adjusts to what the user is requesting at that time. The input manager transitions between different game states.

Take a look at Appendix for more information on the underlying structure of the gesture recognizer and input manager.

2.2.4. UNITY3D INTEGRATION

Unity3D is a fully capable physics engine that uses C#/Javascript programming languages. C++ Libraries can be bundled and imported in the engine provided available software development kits (SDKs). The Oculus Rift and Leap Motion SDKs are fully integrated into Unity3D. Therefore, all the prior sections can be combined by attaching classes and scripts to 3D objects in the scene. The editor allows for 3D scene customization, scripting, and play testing all-in-one until a binary is ready to be built.

2.2.5. USER INTERFACE

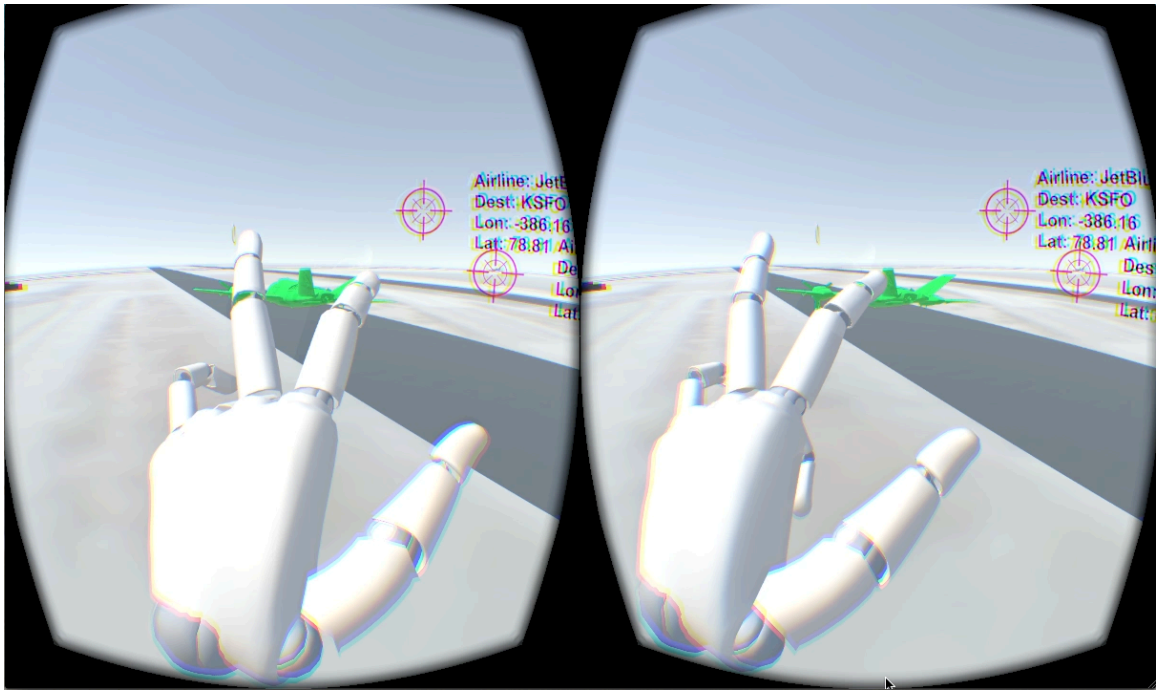


Figure 3: User Interface of ATC system

Figure 3. gives an small depiction of what the UI looks like. Designed to be seen from the Oculus Rift, two cameras that are rendered on each half of the screen to provide a stereoscopic view. Additional text and GUI elements can be added to annotate items in the scene. In this case, reticles with text descriptions are seen on the plane models. When recognized by the Leap Motion controller, hands are rendered as 3D objects.

3. EVALUATION (HOW WELL IT WORKS)

In determining how well my system performed, I executed an evaluation on both the Field Testing and Manual Piloting modes of the simulation. For each mode, I experimented with performance on gesture recognition and usability.

3.1.1 FIELD TESTING – GESTURE RECOGNITION

To evaluate the gesture recognition performance, I recorded corresponding accuracies over a sequence of three iterations. Training over a set of at least 30 or more samples, I recorded the average accuracy of detecting the right gesture at the time it was invoked. Accuracy percentages for gestures are also determined by a confidence factor influenced by a Euclidean distance metric.

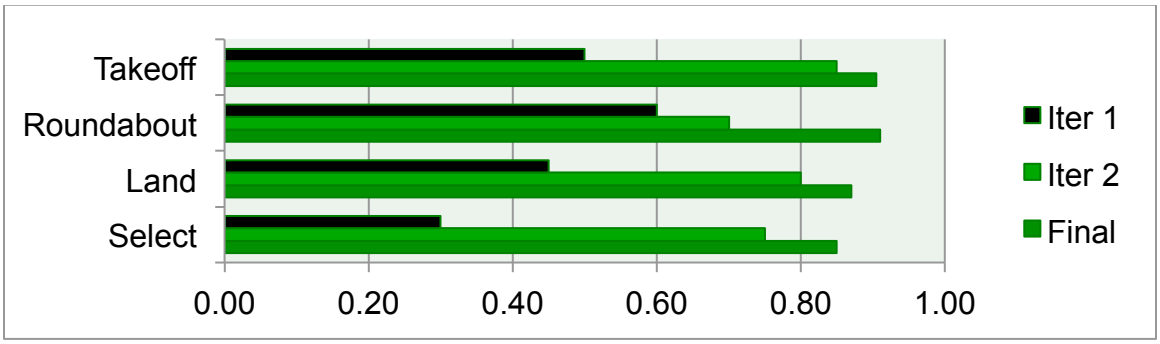


Figure 2: Summarizes gesture recognition accuracy per iteration for Field Testing

- **First iteration** – For my first prototype, I used the default Leap Motion configurations and simple gestures to detect swipe and circle gestures. Initially for all tasks, accuracy was pretty low (no more than 0.60). The Leap Motion was recognizing gestures, however, more frequently than expected. Gestures were being invoked when not initially intended by the user, resulting in many false positives. This could be because no thresholds/customizations were applied and therefore anytime the user placed and examined their hands in front of their face, multiple gestures were detected when a task was not defined yet. Also, raycasting for plane selection performed badly. I initially used a raycast from the direction in which the finger was pointing. However, if the finger was not stabilized or obstructed by the top of the hand, the leap had trouble detecting the right location. Therefore, most selections often went unnoticed.
- **Second iteration** – Much improvement has been made in making gestures perform when expected and more accurately in the second iteration. By comparing speeds and directions of the 'takeoff', 'roundabout', and 'land' gestures, I was able to determine thresholds that provided as additional parameters to when a swipe or circle gesture

was invoked. This helped better distinguish among gestures. However, there was still some error on when the intended gestures were executed. This was from the fact there was no defining bounding box in front of the user, which defines where gestures are completed. The most significant improvement in accuracy was updating the 'selection' method. This worked better because I updated the implementation of a raycast to originate from the camera and pass through the index finger tip position of the user. This helped define the space in which the raycast would occur as well as making this clearer to the player once in the 3D scene.

- **Final optimization** – The final iteration after optimization of gesture recognition had an overall accuracy of at least 0.85. This was because in addition to direction and speed, I decided to take into account stabilization, confidence, and gesture location relative to the player. The Leap Motion SDK allows for a creation of an interaction box in which gestures from the hands are only recognized from a defined space in front of the user. In addition, determining when hands and fingers are stabilized and not stabilized help detect of a gesture was active or not. Minor improvement also worked for 'selection'. Raycasting did better by increasing the colliders of plane models. This way, users can select a plane more suitably instead of hovering over the object exactly.

3.1.2 FIELD TESTING - USABILITY

To test the usability of the Field Training mode, I performed Over-The-Shoulder (OTS) Test scenarios. These scenarios were tasks and actions I wanted the user to complete throughout the duration of gameplay. This included single gestures such as 'select a plane' and 'plane is free to takeoff'. As they completed the tasks, I stood behind and took notes on their acknowledgment, execution, and overall reactions. In addition, I monitored qualities specifically gauged toward air traffic control such as attentiveness, plane coordination and situational awareness.

After the initial tests, there were some pitfalls that prevented an ideal gaming experience and proper ATC management. When trying to perform gestures, users were not sure the exact orientation and position in which a gesture needed to be performed. They also questioned how fast gestures needed to be completed and to what extent. So far the user interface only shows the 3D scene and a model of user's hands once its recognized by the Leap Motion. Once a task is complete or a state has changed, users wanted some visual confirmation like a popup window or text. They had trouble assuring gestures were recognized and executed by the system. When trying to perform gestures, there was some delay between recognizing what task needed to be completed and knowing what gesture correctly corresponds to that action. Most of this was due to a camera that moves in accordance to the user. So if the user is not necessarily facing

forward or draw their attention to another object, they have to re-orient themselves. These above problems combined to make it difficult for the player to complete multiple tasks in sequence consistently, which is a key operation in ATC.

Despite these problems, there were some promising results. Users first were so awe-struck with the virtual reality setting of an airport and seeing planes flying around. It placed them in a perspective that reminded them of a video game. They felt "more attached" with the environment and were able to map plane locations to noticeable landmarks in the scene. Users continued to remain alert and were more aware of their surroundings. Although gesture recognition was still fidgety, there was improved learnability over the course of the test scenarios. Users grew accustom to which gestures needed to be performed and when.

3.2.1. MANUAL PILOTING – GESTURE RECOGNITION

Evaluating gesture recognition performance for Manual Piloting followed the same approach as the Field Testing evaluation. The main difference is that the hand detection and input for navigation correspond to values representing the angle of rotation along a particular axis of the plane model. For simplicity, again, the below accuracy percentages for gestures are heavily weighted by a Euclidean distance metric from unit axis vectors.

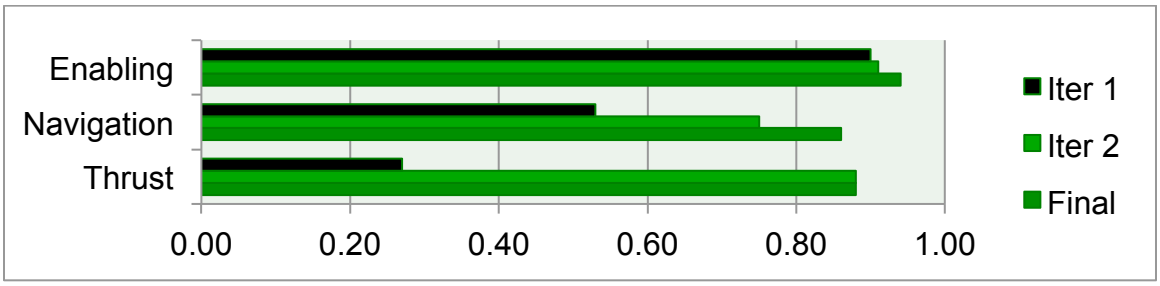


Figure 2: Summarizes gesture recognition accuracy per iteration for Manual Piloting

- **First iteration** – In the first iteration, there were less false positives than the previous Field Testing mode because activation is determined by hand detection rather than gesture detection. In fact, this describes why 'enabling' worked so well. The plane is taken off autopilot and expected to fly manually when two hands are in view. Therefore, once the user places their hand in front of their face, they are ready to fly/control the model plane. On the other hand, 'navigation' and 'thrust' were not working too well. By default, 'navigation' was determined by the angle difference and average location of the hands. No additional constraints were applied to help boost confidence. In addition, 'thrust' was actually performing he worst. In the first implementation, the speed was controlled by how far hands were in front of the user.

This resulted in many problems, especially when users reset their hands but not particularly in the same spot as before.

- **Second iteration** – By the second iteration, improvements had been made to have the 'navigation' and 'thrust' working better. Experimentation allowed for additional thresholds to be applied to 'navigation' axes to ensure the angles and/or locations of each hand do not exceed a maximum or minimum value. In addition, an interaction box was applied here. This way no input was recognized if hands seemed to fall off the screen or were not in the line of sight of the player. The best improvement was done on 'thrusting', which controls the speed of the aircraft. By transitioning to a fist gesture, the recognizer could easily detect whether any fingers were extended and determine if thrust should be on or off. With a closed fist, thrust is off. However, multiple fingers are extended, the user is expected to be willing to fly the plane.
- **Final optimization** – The final iteration after optimization of gesture recognition for Manual Piloting had an overall accuracy of at least 0.87. This is because most of the inputs that determined how the plane was to be controlled relied more on hand detection rather than gesture recognition. Adding into the thresholds produced from the previous iteration, incorporating other factors such as hand stabilization and distance of hands relative to origin helped increase the confidence of where the player wanted to control the plane.

3.2.2. MANUAL PILOTING – USABILITY

I took less of an active role when testing the usability of the Manual Piloting mode of my system. Here, I had the same users complete a Timed Obstacle course. The goal was to navigate a model plane through waypoints represented as floating rings in the air. This required controlling the plane in all three dimensions as well as speed. Users were rated on lap time, average speed, waypoints made, and time actively controlling the plane.

There were some noticeable drawbacks that made users uneasy. Since controlling the plane required constant reading of hand location and orientation for input, any slight movements directly impacted plane direction. Users felt the system needed to be more forgiving to learn the controls faster. After determining the proper way of using the controls, users arms got tired and experienced fatigue after three minutes of gameplay. In addition, constantly moving a plane adjusting speed and direction increased the feeling of disorientation and even motion sickness for some of the users.

This game mode did have some positive feedback from the users. Users this VR test similar to a flight simulator. In contrast to Field Testing, this user study seemed more game-like and players experienced more user freedom. Despite the time it took to learn the extent of the controls, users "got the hang of it". They eventually discovered the best way of completing the obstacle course more successfully and hit more waypoints. This

included less sudden movements in their arms, always remembering to reset themselves when transiting between different directions, and obtaining better control with lower speed. In addition, controlling planes in a VR setting as viewed from a pilot's perspective was well received and users made the connection of how this system could potentially enhance ATC.

4. LIMITATIONS/CHALLENGES

As mentioned in my evaluation on both the performance and usability, there were many noticeable challenges I had to overcome in my prototype. Most of the problems arose from the fact that I was working with a *head-mounted* Leap Motion. Therefore, traditional hand tracking was slightly off due to the new orientation of the hands being viewed top-down rather than bottom-up. This played a significant role in performing gestures in a 3D scene, which required Leap Motion input.

One component that initially took longer than expected to implement was the raycaster for plane selection. When in the Field Testing scene, my original intentions were to have a player select a plane by pointing directly to it. This requires getting the forward direction of the index finger when a hand is detected and performing a raycast to check if any selectable items/planes cross paths with it. However, issues occurred when users pointing directly in front of them. The Leap could no longer detect the index finger if pointing forward because the back of the hand obstructed the camera view. Even when the index finger was in sight, the raycaster was sporadic because the forward direction is constantly updating and does not settle to a consistent point in space. Therefore, I transitioned to performing a raycast directly from the camera to the stabilized tip position of the index finger. The new technique was that the index finger needs to remain in sight (which makes sense, you need to see where you are pointing!).

Some gestures needed to be reconfigured due to the production of many false positives. For example, performing a circle gesture for 'roundabout' sometimes triggered a selection of an airplane if the circle gesture was not registered properly. Therefore, I added some more constraints such as palm direction, speed, turns that help detect when a gesture is being made and when to ignore a raycast simultaneously. In addition, I spent some time on tuning the vertical swipe gesture. A vertical swipe is used in two different areas in my system (1) open palm face player for camera switching and (2) open palm face down for 'takeoff' and/or 'landing'. Since users tend to swipe at the same pace but not always the perfect orientation, these gestures often got switched. Therefore, a way to approach this problem was creating a separation of space where these gestures are to be recognized in 3D. By directly using the Image API, the camera switch gesture can be detected right in front of your face only and not conflict with the other gestures performed at a normal distance from he player.

Some challenges arose in the usability as mentioned in the evaluation. There were two features while initiating gestures that I did not account for. When completing tasks in the Filed Testing scenario, it was difficult to perform instantly since the gesture needed to be executed with enough confidence in order to register. Most often than not, users had to re-execute actions. This draws away from having a very responsive ATC simulation. On the other hand, the Manual Piloting stage was constantly registering the input of the user. It was challenging to make the input forgiving enough such that mistakes do not dramatically alter the planes intended direction.

As mentioned by Leap Developers: 3D Best Practices, there are ways of rendering and interacting with objects in a scene to make it more realistic and move past the conventional mindset of 2D paradigms of screen space. This was noticeable when I wanted to display text to the user. If the text was not directly in the center of the screen, the text got out of focus or not scaled properly. This was handled was by mapping GUI elements on 3D objects in the scene at a fixed distance so they seem more focused.

5. FINAL STATE & FUTURE WORK

As a result, I created an immersive and interactive ATC simulation. This system utilizes the Oculus Rift and Leap Motion to create a VR setting where actions and ATC commands are guided by your hands. After much configuration, most gestures are recognized and an enhanced situational awareness is achieved. However, much improvement needs to be made to make this a more useful product.

Even though the gesture recognition accuracy increased, there are times in which performing a gesture is still not recognized fully by the user. This is a critical problem since it impedes fluid gameplay and adds frustration to the user. A way to increase gesture recognition is to add more attributes and pay strict attention to more salient features that characterize a gesture. For example, in addition to just capturing speed and direction of a gesture, capture to global start and end location of a gesture. This could help narrow down the scope of which gesture is trying to be recognized. In addition, incorporating speech recognition like Sphynx-4 could help assure which command the player wants to execute and boost performance.

Improvements on the usability end could help make this a more practical application. User-specific calibrations could help improve the gesture performed and ease of using the system. One idea is to create a tutorial in which users learn the gestures that are available to them. By trying to mimic a playback of gestures in the scene, the system could potentially adjust to the user's natural hand movements such as speed, direction, etc. to improve the performance of the system. In addition, the user interface could be dramatically improved. So far, my first implementation just has the basic functionality and not the aspiring aesthetics you see in more-developed video games. By creating a heads-up-display (HUD) with appropriate text, better signifiers could let the

user know more about the scene and their state (e.g. whether an action is being performed, if a task has been completed, etc.).

Lastly, making this system more robust and scalable could increase the chances of this tool being used more practically for ATC. So far the tasks performed are very simple and does not directly correlate to actions performed by real air traffic controllers. By creating multi-dimensional levels that better describe real-world scenarios, ATC practice can be tested more clearly in this VR setting. With a larger gesture set, tasks can be executed on a larger scale that's more realistic of ATC during training or in the field.

6. BIBLIOGRAPHY

1. Unity3D. http://unity3d.com

2. Oculus Rift. http://www.oculus.com

3. Leap Motion. http://www.leapmotion.com

4. S. Djorgovski, P. Hut, R. Knop, G. Longo, S. McMillan, E. Vesperini, C. Donalek, M. Graham, A. Mahabal, F. Sauer, et al. The mica experiment: Astrophysics in virtual worlds. arXiv preprint arXiv:1301.6808, 2013.
5. C. Donalek, S. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake, et al. Immersive and collaborative data visualization using virtual reality platforms. In Big Data (Big Data), 2014 IEEE Inter- national Conference on, pages 609–614. IEEE, 2014.

6. A.P.d.O.Santos.Three-dimensionalvirtualenvironmentin air traffic control tower: A sistematic review. In Proceedings of the 2014 Fifth International Conference on Intelligent Sys- tems Design and Engineering Applications, pages 156–162. IEEE Computer Society, 2014.

7. R. Reisman and S. Ellis. Augmented reality for air traffic control towers. In ACM SIGGRAPH 2003 Sketches & Appli- cations, pages 1–1. ACM, 2003.

7. APPENDIX

7.1 HARDWARE/SOFTWARE SETUP

Above shows pictures of the complete hardware setup of the final system. The Oculus Rift DK2 acts as an external display through HDMI. Enabling Mirror display in display settings allows the application to be viewed on both the head mounted display and computer monitor during runtime. In addition, Rift sensory information is provided via USB. Typically, a Leap Motion VR plastic head mount is added placed in front of the rift. However, a simple hack I did was to just use packaging tape and being very careful when applying/taking off the Leap. A USB extension cord was necessary to have the Leap Motion cord reach the computer. Below summarizes the current hardware/software specifications that were used in this project.

Hardware	Software
Oculus Rift DK2	SDK/Runtime 0.5.0.1-beta
Leap Motion	SDK v2.2.4.26750
MacBook Pro (Mid 2010)	Mac OSX 10.9.5
Asus Gaming Laptop	Windows Vista

7.2 NOTABLE DOCUMENTATION

Due to size limits, final project resources and Unity project are located via Dropbox at https://www.dropbox.com/sh/itxvp38sar07eic/AAAUzd1FAymSuKTxaSikkb6Ba?dl=0 Remaining materials (paper, builds, slides) are posted under the .zip Stellar submission under account andrewmo@mit.edu. However, I have listed notable scripts that are used in the project implementation. Please refer to the README.rtf file on the submission site for more information.

- **AeroplaneController.cs** – input script for plane model that reads in controls from LeapFly.cs when in Manual Piloting mode to control plane directly.
- **AeroplaneAiControl.cs** – AI script for plane model when performing actions such as takeoff, roundabout, land when player is not controlling plane directly.
- **LeapFly.cs** – new script that extends leap motion SDK to recognize input for Manual Piloting mode
- **GestRecognizer.cs** – addition to Leap SDK simple gestures to cater to newly defined gestures while in Field Testing mode.
- **InputManager.cs** – manages inputs and changes state of scene if necessary (Field Testing or Manual Piloting mode, plane selection, camera toggle, etc.)
- **SelectionFinger.cs** – augments Leap SDK default hand physics model script to allow for raycasting and plane selection.