**CSE379 - Senior Project - 3D PowerPoint**

**Matthew Raporte, Greg Potter**

**Advisor: Professor Huang**

**a. Executive Summary:**

In our senior project we wanted to explore practical uses of affordable new technology. The field of AR/VR development is very promising, however it is commercially underappreciated because of a lack of affordable hardware and lack of “every day” uses for a casual consumer. We created a product that allows users to launch and present a PowerPoint presentation using only natural hand movements, by utilizing Leap Motion’s 80$ 3D sensor to wait for and detect gestures, and using Microsoft Message Queueing to relay them to an application that issued PowerPoint commands using an API.

During our project we encountered latency and accuracy issues in our gesture detection component, but after testing were able to build an efficient and highly extensible detection architecture with minimal API calls.

**b. Introduction & Motivation:**

i. What is the problem?

Development and spread of AR/VR technology is being bottlenecked by processing power limitations, public perception, and lack of accessibility.

ii. Why is it interesting/important?

Because this field has tremendous scientific (telepresence, data visualization, simulation) and commercial (gaming, social media, 3D movies) potential. Additionally, hardware and software solutions are starting to catch up to the public’s expectations and more AR/VR products are being released, making this a very exciting field to develop for.

iii. How did you propose to solve the problem?

By exposing an audience of people to 3D computer vision, gaining experience in this field, and creating a final product that is equally practical and exciting.

iv. What are the merits of the proposed solution?

Our final product will allow users to present slideshows using natural hand motions instead of a keyboard and a mouse. More importantly, it will expose Lehigh University students to this technology and create an extensible architecture for Leap Motion gesture mapping.

**c. Project Planning:**

i. Project Summary

Our project will utilize Leap Motion technology to detect unique hand gestures during a Power Point (or an open source alternative) presentation and execute associated commands, including moving to next/previous slides, skipping slides, pressing embedded buttons, and flipping between slide show mode and outline mode. This will help popularize a very promising new technology and allow for a more fluid and natural presentation. Our “reach” goals include image resizing/movement, drawing shapes onto the screen and directory browsing/presentation setup.

ii. Project Organization

Team Members:

Matthew Raporte, Gregory Potter

Client:

Professor Huang

Who on the team will be interfacing with client:

Matthew Raporte, Gregory Potter

Client meetings and status updates:

-Progress Reports:

Emailed out to Professor Huang every Sunday.

-Deliverables:

9/29: Project Planning

10/06: Progress Report 3

10/20: Project Design

10/27: Progress Report 4

11/03: Project Implementation

11/10: Progress Report 5

11/17: V&V and Documentation

12/01: Final Posters

-Face-to-face meetings:

Wednesdays at 3PM, Prof. Huang’s office.

iii. Requirements:

Customer level requirements:

-Easy to setup and use

-Intuitive and easy to remember commands

-Strong gesture detectability, doesn’t “miss” gestures or falsely detect gestures

-Quick detection mapping and command execution

-Robust but useful features

Mapping to technical requirements:

*This maps the user requirements identified in the project definition into specific technical requirements*

Development Platform (Linux, Windows, Android,…):

Windows because many of the APIs we want to connect to (like PowerPoint) are currently in Windows only

Development environment / software / languages:

C#, Java, Leap Motion SDK

Performance requirements:

Efficient processor usage to ensure timely command execution, good garbage collection in order not to clog up system with constant Leap Motion input. It should be able to seamlessly discard data where nothing is happening so that we aren’t constantly calculating data

Resources required:

Leap Motion Controller

PC/Laptop running Windows with applications like PowerPoint

Testing methods:

We will cycle white-box testing during development and extensive black-box testing after we have a working prototype. We will possibly utilize our product for one of our in-class presentations after sufficient testing.

iv. Project Management

Brief description of project execution

We will start by researching the different APIs that we will use to get an idea of what is possible and how easy it is. From there, we will try to implement some of the simpler controls which are mostly around advancing/skipping slides, etc.

Clear enumeration of customer requirements

– Identify a minimum scope you must accomplish to meet customer requirements (and pass the course)

* Go to next, previous, and skip slides
* Play media or otherwise interact with the slide itself (highlighting perhaps)

– Prioritize remaining tasks of the desired scope

* Integrate with the operating system (window management)
* Operating system login screen

Identify risk areas, and how these risks will be minimized

The bulk of the time will be spent doing the investigation into what we can actually get done with the Leap Motion device and the PowerPoint API.

Interfaces:

– Leap Motion and the PowerPoint API

Self-evaluation of progress

By the end of the house, we should be able to deliver a presentation about our project using our project. Along the way we will want to keep in mind that this should be able to accomplish the productivity tasks faster or easier than with traditional keyboard or mouse. At each stage, we will need to make sure that it is performant and accurate.

**d. Software Design:**

Detection

Overview: The *Detection* component will be written in Java and will use Leap Motion SDK tools to recognize the physical gestures we will be implementing, parameterize them accordingly and call corresponding *Action* components using the *Translation* process. *Detection* will also initialize processes we’ll be using for large-scale efficiency and correctness testing.

Input: Physical hand gestures.

Output: Live-time gesture identification, parameterization and forwarding information to *Translation*.

Interfacing: *Detection* interfaces with physical hand gestures using infrared sensors and cameras built into Leap Motion and translating their coordinates into JSON/Java objects then uses a listen loop to detect predefined motions. *Detection* will interface with *Action* by signaling corresponding *Action* functions using a (signal based?) process we will develop during *Translation*.

Prototype functionality: Base level prototype will detect 4 unique “minimum-scope” gestures (“Next Slide”, “Previous Slide”, “Skip Forward ‘X’ Slides”, “Skip Backward ‘X’ Slides”) and print the name of the command to the console in live time as well as store raw Leap Motion input data/interpreted commands to a text file for troubleshooting/data gathering. The prototype will include: *(1)* False-positive error handling in order to prevent false command execution [performing a software action when no physical input is given]. *(2)* False-negative error handling in order to prevent physical actions going unnoticed. *(3)* Incorrect read error handling to prevent physical actions being misinterpreted into incorrect software actions. Additionally the prototype, and subsequent version, will include functions/classes for large-scale testing and logging.

Once we have built a prototype we will test it for “correctness” and “speed”. First we will design uniform descriptions for each action, an extra one to “wait” and several non-implemented actions that might randomly occur during a presentation. These could be in video-demonstration form but will most likely be a text string, as they should mimic our final product’s user instruction documentation. We will sample at least 5 total volunteers and in order to target a wide demographic we will empirically sample testers of various ages (18-50), hand sizes, technological proficiency and public speaking/Power Point experience. Subjects will be given a list of 100 descriptions randomly picked and ordered from possible actions in list format (ie: “1: wave hand parallel over sensor from left to right 2: wait 10 seconds 3: hold hand over sensor with fingers spread then close hand into fist”). We will analyze and statistically manipulate this data in order to ensure/demonstrate the highest “correctness” level possible because of the tremendous importance of accurate input reading to our product. While running correctness testing we will also need manually keep track of [what we perceive to be] human error where subjects misinterpret an instruction, we will factor out human error cases from our results and also use results to re-define action descriptions (and include video demonstrations if this becomes a large scale issue).

The second part of *Detection* prototype testing will judge latency between physical action and corresponding command execution. Running latency testing in parallel with correctness testing will give us a large sample size without unnecessary work, but more importantly it will allow us to analyze processing speed for correctly processed inputs separately from processing speed of error cases and optimize/handle them separately. We will test the time from when Leap detects a hand

Alpha functionality: The full scale Alpha version *Detection* component will handle the same 4 commands the prototype does and include the same methods for testing, however after parametrization it will call corresponding *Action* components (functions) using whatever protocol we find most effective during *Translation*.

Beta functionality: The full scale Beta version *Detection* component will detect every gesture mapped in our customer requirements, include toggle-able testing methods and call the correct *Action* component with excellent precision and speed.

Because full-scale logging will be existent in the Alpha version, new gestures will be created and tested separately from the rest of the product in order to determine their individual efficiency before being integrated with the whole and being ran as full-scale tests.

Extensibility: *Detection* will be easily extensible because each gesture is separately defined and the testing/listening loop of the code will never change. New gestures can continue to be defined, and we will have the option to write new gestures as combinations of previous ones.

Technical:

Leap API has four built in gestures (circle, swipe, key tap, screen tap) that are defined by acceptable ranges for each of the Hand object properties: isLeft/isRight, Palm Position, Palm Velocity, Palm Normal, Direction, Grab Strength and X-Y-Z coordinate position. A start and end point hand position has to be defined, as well as time frame and any intermediate positions.

We used sample.html (<http://goo.gl/B3YSzK>) and JSONViewer.html (<http://goo.gl/PXNFWT> ), SDK tools to analyze different gesture properties, and Leap Trainer- an open source project that allows users to record physical gestures and then matches them back- to develop an understanding for gesture recognition (<http://goo.gl/9jV3YI>).

Built an early version of the prototype in IntelliJ using built in “Rotate” and hand-written base level “Swipe Left” and “Swipe Right” commands (<http://goo.gl/9nudGY> ) which work about 75% of the time. Had to try multiple compilers before I found one that was able to link the Leap libraries properly, not sure why but there were a lot of reported issues online about this.

Translation

Overview: This is a conceptual component in order to emphasize the importance of how we will handle interaction between detection-action components in order to instantly execute detected commands without compromising ability to detect ongoing gestures (and therefore impose a detection pause between gestures). This could be relatively easy, but it depends on efficiency of *Detection* and *Action* components. Either way we need to research and explore different methods of signaling/passing/execution between Java based *Detection* and C# based *Action*.

*Translation* will also include full-scale testing/data gathering for Alpha and later versions, we will have to be careful not to issue too many file open()/close()’s due to their relatively high overhead. We will also implement a global option to toggle this logging in order to maximize efficiency outside of testing.

Input: A parameterized version of a physical gesture in Java generated by *Detection*.

Output: A highly efficient call to an *Action* function corresponding to the input

Interfacing: We will most likely use messaging to connect the detection to the action.

Prototype functionality:The prototype portion of *Translation* will be the last prototype designed because requirements will be heavily dependant on *Detection* and *Action* efficiency/cost. We will analyze the prototype testing of other components and identify weaknesses. Then we will research and explore several implementation possibilities, test the most promising ones and possibly consult approach Professor Spear about a multithreaded approach.

Alpha functionality: In our products’ Alpha version we will implement the most effective *Translation* prototype in order to match necessary behavior mentioned in *Translation - Overview*. We will use the full-product testing methods we implemented to test full-product correctness and efficiency, and find *Translation* effectiveness by comparing full-product results against individual *Detection* and *Action* results.

Beta functionality:In our product’s Beta version, *Translation* will be a tested and improved version of the Alpha. Additionally the Beta version will include a simple option to dispose of runtime data instead of saving it.

Action

Overview: *Action* is called by *Translation* and executes its’ C# code and uses Microsoft PowerPoint API to generate a specific function inside the currently active PowerPoint presentation.

Input: The command from *Translation* such as “next slide” (in testing, this command could also come from another source such as a keyboard shortcut)

Output: The desired action occurring to the PowerPoint presentation.

Interfacing: Will listen to messaging queue (or whatever the input needs to be) and will respond to the commands.

Prototype functionality: During Prototype *Action* will be an independent program with 4 functions (corresponding to each Prototype implemented gesture) that will execute their designated actions in PP. The functions will be repeatedly tested in a main() method to ensure they work under correct circumstances and that exceptions are properly caught and handled under incorrect circumstances (such as PP not being installed, being closed, etc).

Once individual *Action*’s are working correctly we will mass-test their execution speeds and gather data. Outliers may be redesigned using different objects/calls if we find it necessary.

Alpha functionality: The full scale Alpha version *Action* component will handle the same 4 commands the prototype does and include the same methods for testing, however action calls will be determined by *Detection* gestures using whatever protocol we find most effective during *Translation*.

Beta functionality:The full scale Beta version *Action* component will include every action mapped in our customer requirements, include toggle-able testing methods and execute the correct PowerPoint action with excellent precision and speed.

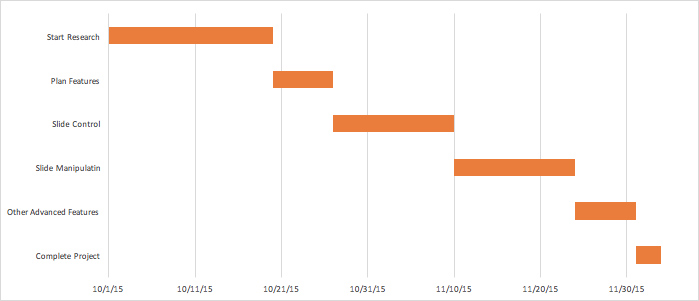
Because full-scale logging will be existent in the Alpha version, new functions will be created and tested separately from the rest of the product in order to determine their individual efficiency before being integrated with the whole and being ran as full-scale tests.

Technical:

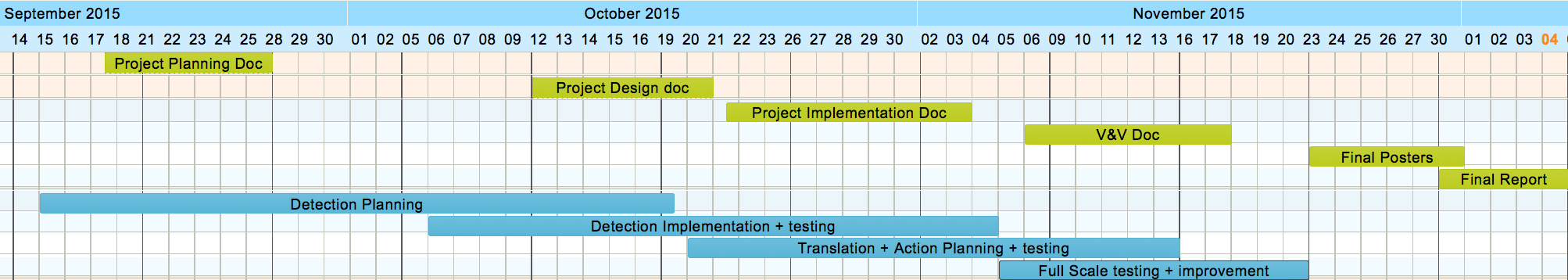
Using the Visual Studio Addin API for Microsoft Office, we are able to tie in our application with PowerPoint. We will be able to add configuration to the “ribbon” (see<https://msdn.microsoft.com/en-us/library/aa942955.aspx>). Most of the slide control functions are defined in the MSDN docs for PowerPoint addin. By tapping into the “Slide” object, we can control the current slide as well as interact with media elements on the slides themselves.

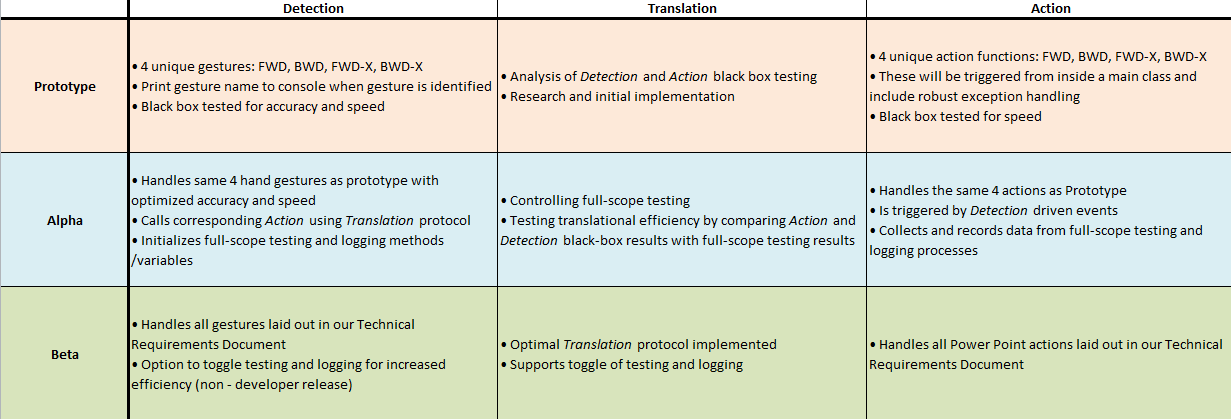
**e. Implementation Results:**

Original Gannt chart:



Updated Gannt chart:



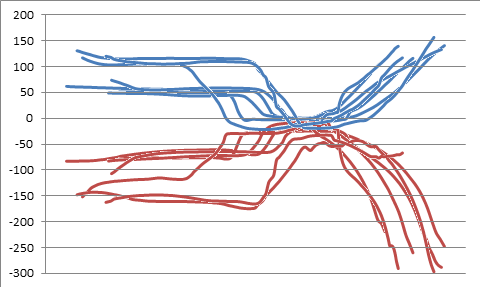


We had to spend more time researching and testing/improving our code than originally expected, however we were still bound by submission deadlines so we followed a similar documentation schedule. Splitting the project into components and developing/testing them separately helped us develop this on schedule.

i. Show results of what you did.

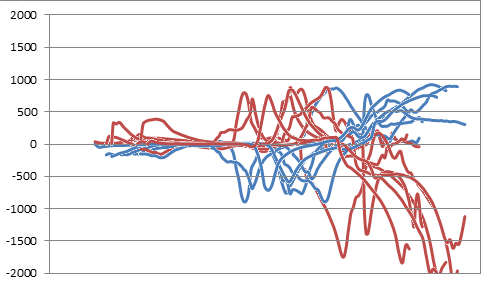
Created tool to map selected gesture data to Excel in order to better define gestures:

Clap Gesture X-position - Time Mapping



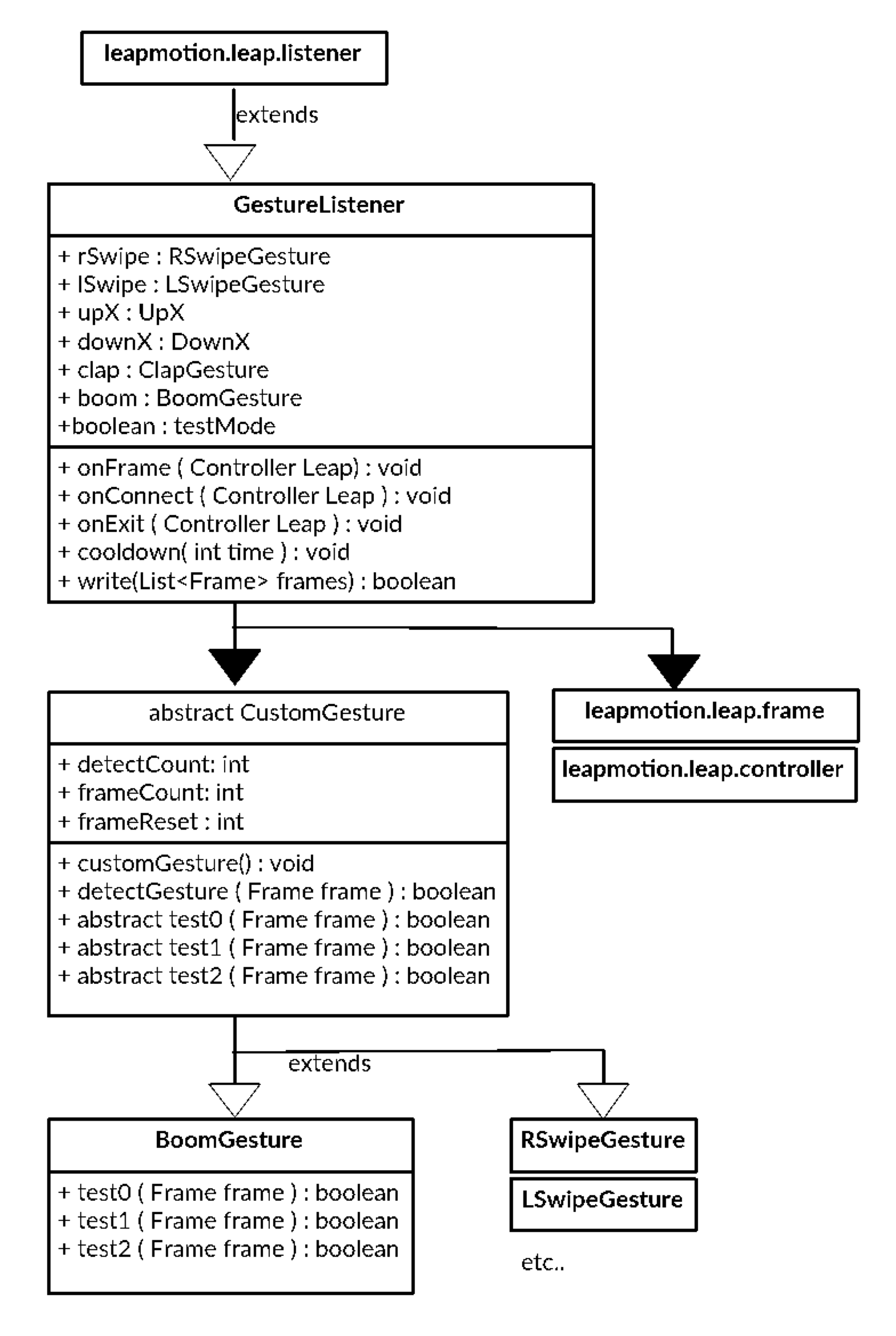
X position is in mm, time is measured in frames (at 80-110 fps)

Clap Gesture X-velocity - Time Mapping



X velocity is in mm/s, time is measured in frames (at 80-110 fps)

UML Class Diagram of Detection Architecture



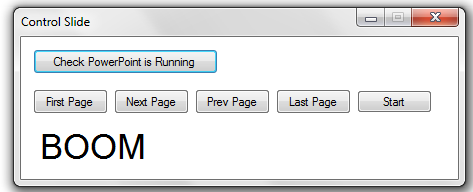
UML Explanation:

Detection implements an extended API Listener object to test each detected Frame for gestures in prioritized order using detectGesture calls. detectGesture calls the gesture’s start, midpoint and end condition tests according to detectCount, and increments it when a test is passed. Each custom gesture extends CustomGesture and implements it’s own logical tests.

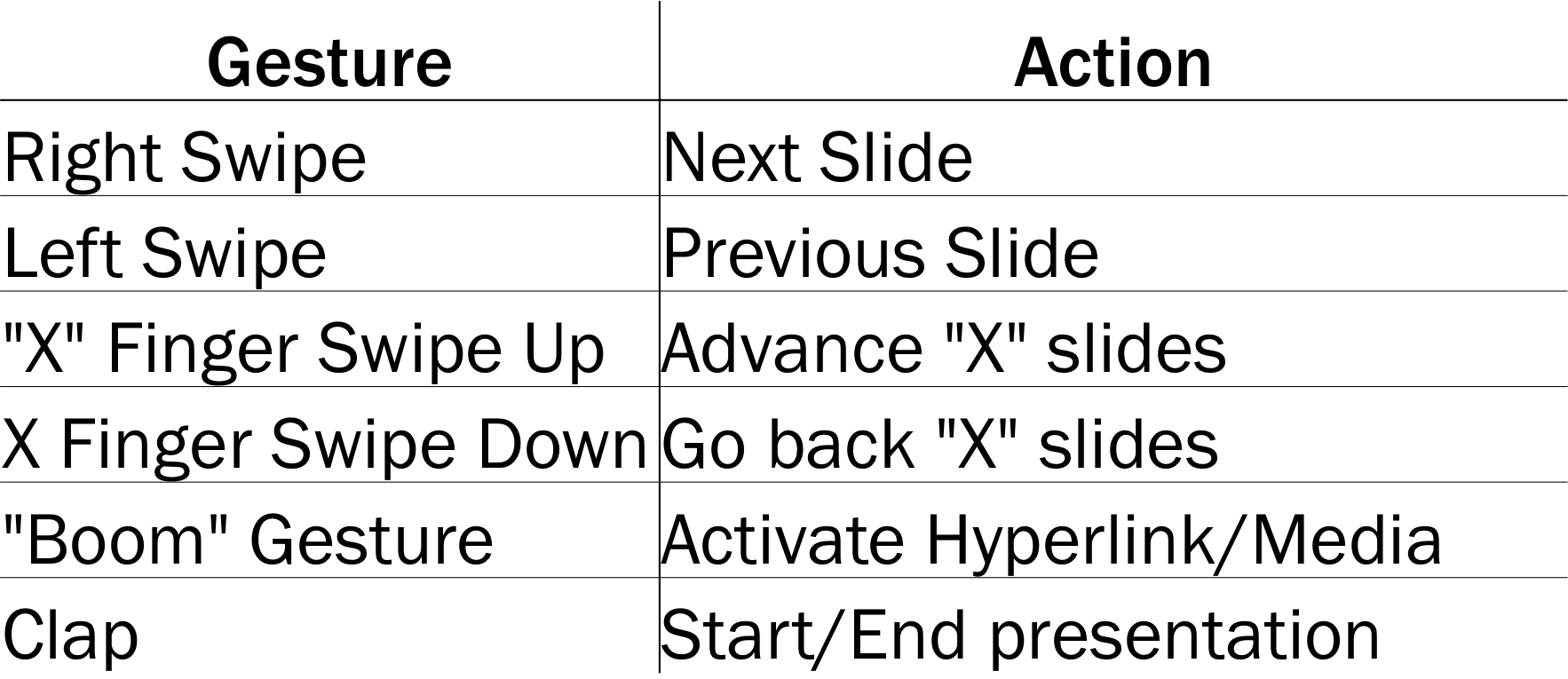
For example, Clap gesture test0() checks if there’s 2 hands present in the frame, checks that their palms are facing each other (absolute value of the sum of palm normal x vectors is below a threshold) and checks to make sure all fingers are extended, then stores hand position and hand id’s if the test is passed and increments detectCount. In the next frame, Clap gesture is tested first because of it’s higher detectCount, and test1() is called. test1() checks the hand id’s to make sure neither hand has left the frame, once again checks palm normals and fingers, and then compares current hand position to the ones stored by test0() to see if the hands got closer. If test1() is passed, test2() is called and follows a similar procedure. If a test is failed the frameCount variable is incremented, and once enough (30) consequent frames fail a test the detectCount is reset.

GUI for standalone C# PowerPoint Controller

We created a standalone GUI for the PowerPoint “Action” component in order to facilitate testing.

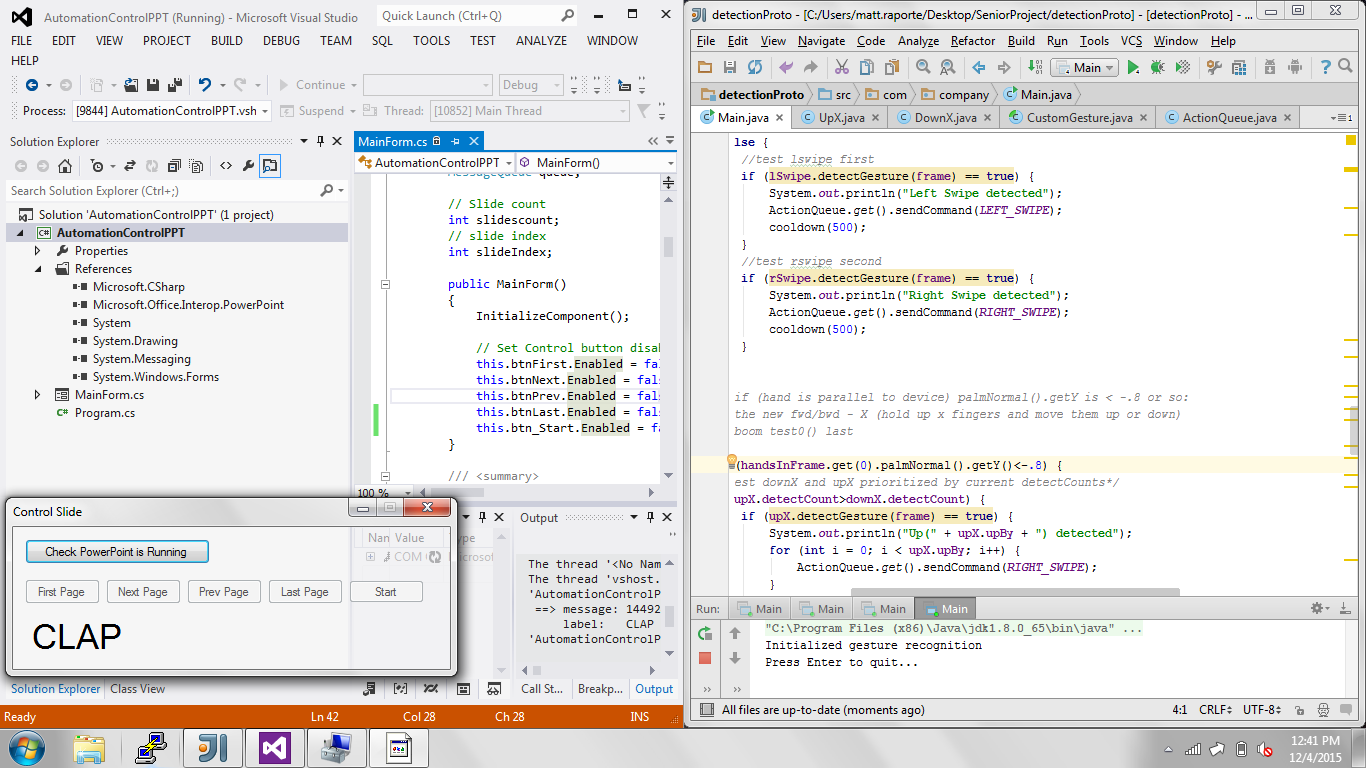


Final Product:



Code base: <https://github.com/matt-raporte/Senior-Project>

Screenshot of *Detection* and *Action* running in parallel with *Translation* relaying commands (the processes running in the background of our project):



**f. Verification & Validation:**

*i: Detection* Test: (White-box)

1. Each Gesture will be tested for accuracy by having random users issue gestures, and having the *Detection* component print gesture names to console; each Gesture will be simultaneously tested for speed using a built in clock to time latency between recognition (object detected in frame) and gesture recognition (outprint). Running these tests concurrently will allow us to evaluate speeds of false-detection, mis-detection and correct-detection separately. *Detection* accuracy will also be judged and improved by testing and assessing edge cases, and the *Detection* process speed will be further evaluated using Big O notation.
2. The test will be passed once accuracy for every gesture is >90% and latency is <1.5s, or we have proved hardware or software limitations.

ii: *Translation+Action* Test: (White-box)

1. Each gesture/command will be dropped on a queue with a timestamp so that events that the *Action* phase might miss aren’t all of a sudden batched up with
2. We might add something to make sure that there is a “cooldown” period between commands which we can test by manually firing many events and seeing how many (and which ones) actually get through the queue.

iii: Full-Scope Test: (Black-box)

1. Full-Scope of our project will be tested after each component passes its white-box testing. Full-Scope will be tested for accuracy by having testers issue random gestures with Power Point running, and corresponding PowerPoint behaviors (or lack of behaviors) will be recorded and quantified per Gesture-Execution pairs. Full-Scope will also be tested for speed by simultaneously recording intervals between object recognition in frame to *Action* execution while accuracy testing is conducted.
2. This test will be passed when every gesture results in its corresponding Power Point *Action* being executed at <2s from the Leap Motion controller capturing a “hand” frame. With at least an 85% accuracy.

iv: Full-Scope Live Functionality Test: (Black-box)

1. After Full-Scope testing is passed we will conduct a live test of our full project by using it to control an informative slideshow we will present alongside our Poster Presentation. In order to conduct this test before the final presentations on 12/04, we will have to pass our Full-Scope testing phase before the practice presentation dates of 11/24 and 12/01.
2. This test will be passed when we are able to complete the whole presentation without any non-Leap input device assistance, use each gesture at least once, and receive a positive response from the class and Professor Chuah.

Testing the detection component caused us to prioritize speed and design an architecture that avoided unnecessary operations and minimized API calls. We originally planned to use Leap Motion API’s “Swipe Gesture” and “Circle Gesture”, but after testing them chose to implement our own versions.

Translation and Action component testing passed without issues, and Full-Scope testing helped us prepare a functioning demo for the presentation. As a result of Full-Scope testing we had to implement a longer “cool-down” period to prevent double gesturing, and relay FWD/BWD-”X” as a string of repeated FWD/BWD commands

**g. Documentation:**

The project was documented through a series of document submissions: Project Planning, Project Definition, Project Design, Project Implementation, and Project V&V, all of which are summarized in this document.

Code is also extensively commented, with notes about implementation, progress and issues at the beginning of each class and method.

Lastly, the README file packaged with the project includes basic instructions on how to download, setup and use our product.

**h. Future Work:**

We successfully completed our project. Future groups may build on our project by implementing a similar gesture detection/queueing architecture in their Leap Motion powered projects, or by developing new gesture-action functions to run alongside ours. New gestures can be easily implemented by creating new gesture classes that extend CustomGesture and override it’s test methods. Existing or new gestures may be refined and made more precise by adding test methods to CustomGesture, and having simpler methods increment detectCount past unused methods.

Additionally, our detection component may be further improved by testing a more robust set of conditions (to improve detection accuracy), using less API calls and utilizing multithreading to improve speed.

**i. Discussion:**

Working on this project taught us a lot about project planning, especially when building projects using new technology/tools. We’ve also learned about the benefits of componentizing large scale projects, and building through testing.

Specifically to our project, we found the Leap Motion controller to be less accurate/efficient than envisioned, and we were challenged to meet accuracy and speed goals. In hindsight we would have liked to spend more time planning and researching before committing to a project idea.