CSE 379: Project Design

Matt Raporte and Greg Potter

10/13/2015

Intro:

We have decided to organize our project into three components: “Detection”, “Translation” and “Action” in order to facilitate project building, black box testing and future extensibility. The following Project Design document describes these components in terms of their functional overview, interfacing elements, and input/outputs. Our project design also details the technical requirements and specific testing methods used at each level of the software development process as well as a “technical” subsection for each component which will describe the technical implementation, what parts we have developed, and occurring problem areas going into development.

Components:

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.