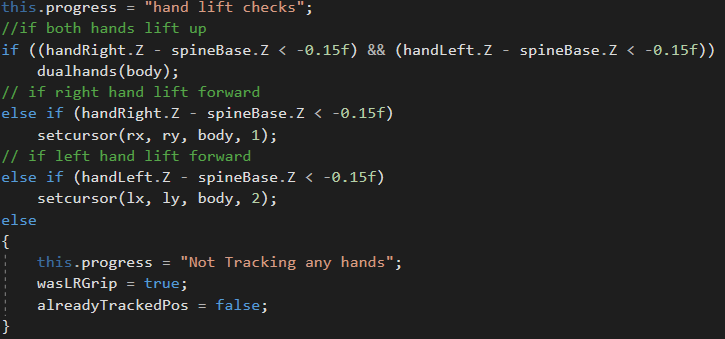
dupliAbstract:

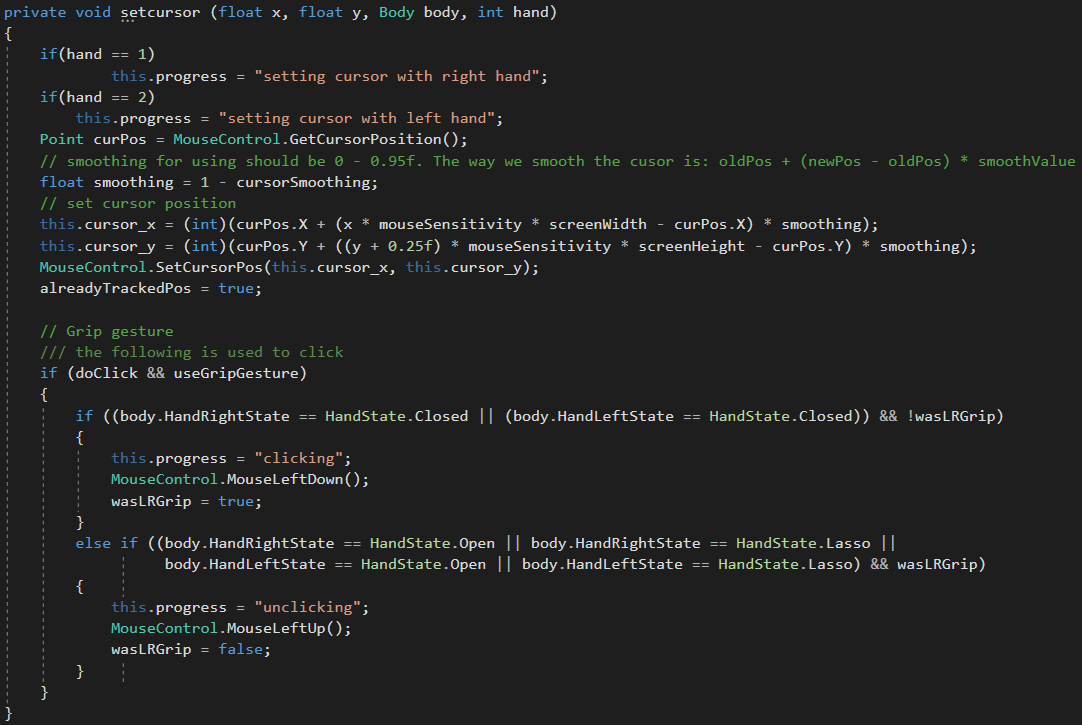
By taking the project, Smart Mirror started in Spring 2016 and continued in Fall 2016, we are improving the functionality of the gesture software named *Kinect V2 Mouse Control (KV2MC) [1]* in Summer 2017*.* Improved functionality includes the following: left and right hand (dual-hand) recognition and a debug menu that returns values read by the Xbox One (XB1) Kinect to the window form of the program.

Body:

Initially, we were going to program the application from scratch, but have decided to use the source code from the *KV2MC* software, which is stored in the GitHub. From there we researched on how the Kinect was used inside the *KV2MC* software and see how to incorporate dual hand recognition. There was another software that we were also looking at to work with the smart mirror, but was not open-source.



[A]

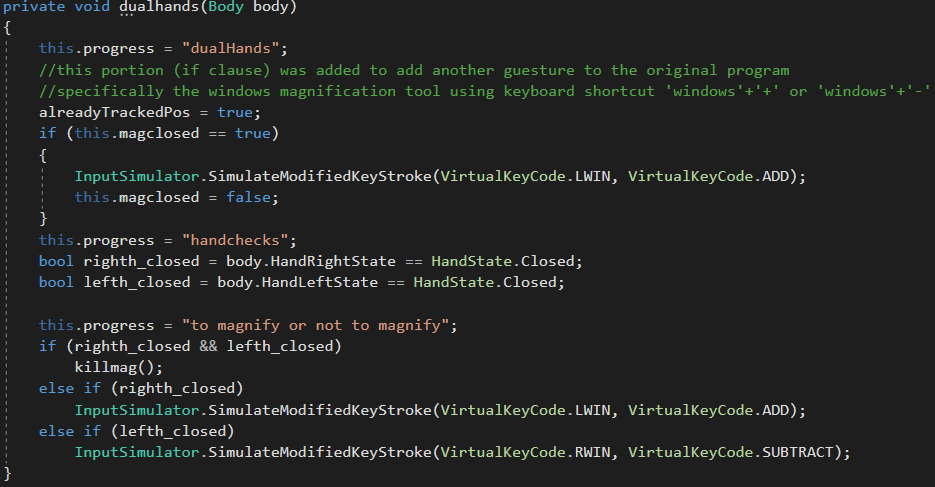


[B]

The code snippet above shows the improved hand checks in the developers application [A] from the original program. In the original program, it only recognized the right and left hands separately. In addition, where the "setcursor" function resides in picture [A], the original program had "duplicate" code for each left and right hands for setting the cursor on the screen. As shown in picture [B], it displays the "setcursor" function and is for the left and right hands to set the cursor on the screen. The code was moved into a function to help the functionality of the program perform better.

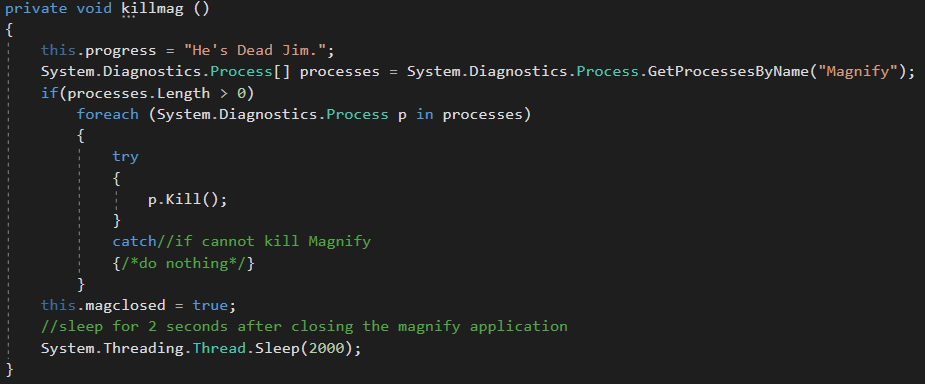
We did have some issues in implementing the dual hand recognition. One issue was when the program would hang when the program recognized the two hands. This was due to having to check whether either hand was raised inside the dual hand recognition portion of the hand checks.

First, we tried to implement a gesture such as gripping both hands close together then pulling both hands away from each other for a "zoom in" gesture and vice-versa for a "zoom-out" gesture. Unfortunately, we could not implement this as the program would bog down. To remedy this, we just checked whether or not if both hands were closed or either hands were closed.



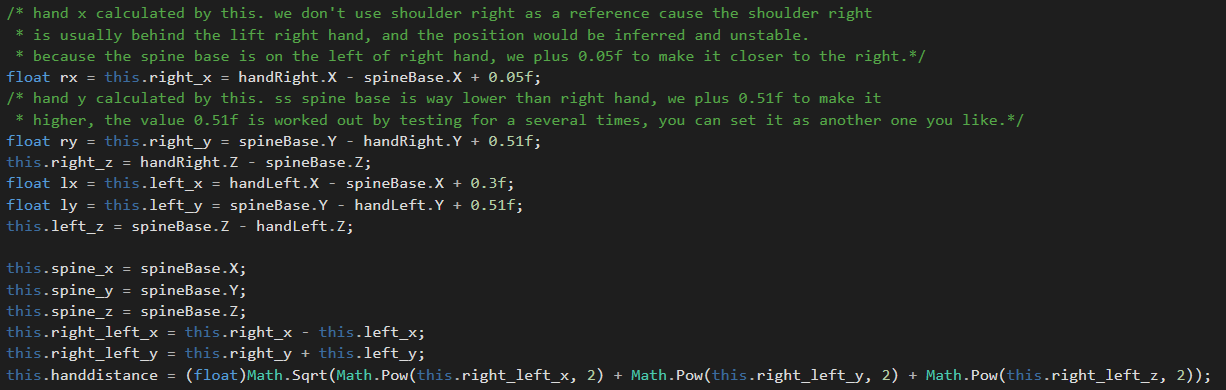
[C]

As seen in picture [C], when both hands are recognized (not closed and are raised forward towards the Kinect) the program would launch the application "Magnify.exe". This Magnify application would zoom in or out of the screen in specific position where the cursor is currently at. When both hands are forward and are closed, the program Magnify is closed with the function "killmag()". When both hands are forward and the right hand is closed, the program will send a key-combination "Windows" with the "+" keys to zoom in onto the screen at the last known cursor location. When both hands are forward and the left hand is closed, the program will send a key-combination "Windows" with the "-" keys to zoom out of the screen.



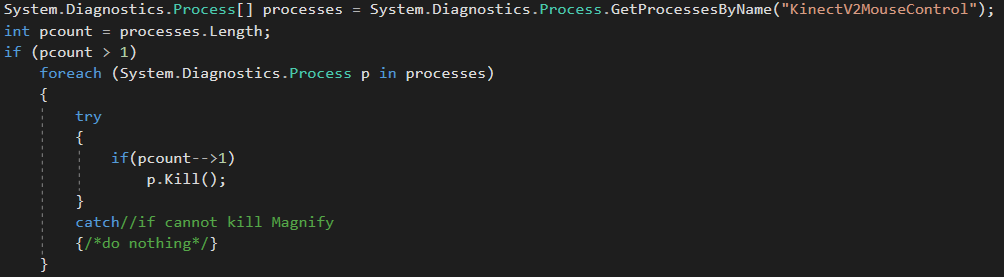
[D]

The killmag function is shown in picture [D]. A try-catch is used as a measure if the program cannot close down the magnify application. The reasoning to closing the magnify application is to not have the magnify glass on the screen when the magnify application is not in use.



[E]

The code above lists sets up the Right hand (X,Y,Z), Left hand (X,Y,Z), and Spine (X,Y,Z) values. These values are used in the set cursor and the dual hands functions.



[F]

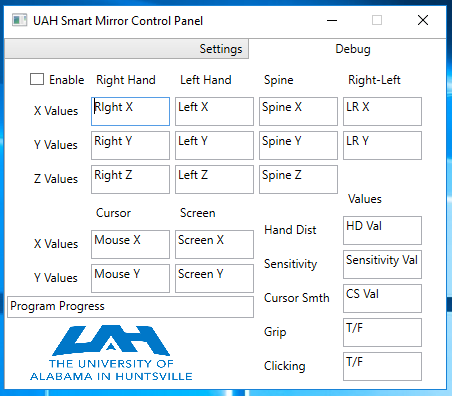
The snippet of code shown in picture [F], was added into the code to prevent duplication of the same application running. On the Intel Compute Stick, if the phrase “Control” was said multiple times, multiple instances would launch and would require multiple phrases of “Stop” to be said in order to close all of the instances of the application.



[G]

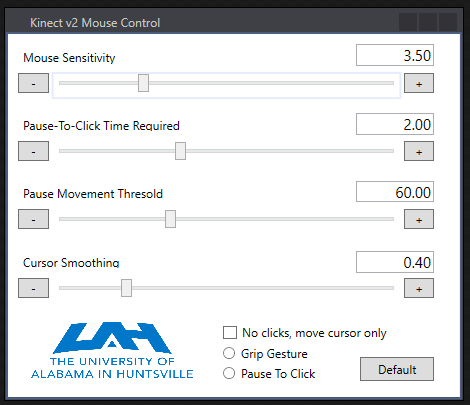
In the picture [G], the main GUI is displayed. Labeled are what the GUI consists of on the main page. There are four values that are variable through the "plus" and "minus" buttons or through the slider. The textbox is only there to serve a purpose to tell the end-user of what the value is. The Mouse Sensitivity value how far the cursor will move relative to the hand position recognized by the Kinect. “Pause-to-click” value is the time in seconds to wait for the cursor to wait on a specific position to click if the option is enabled. The pause movement threshold value is the radius range for a hand to move inside a “circle”. The circle is noted by the Kinect and how the Kinect recognizes the hand state. The cursor smoothing value determines how smooth the cursor will appear on the screen. The closer the cursor smoothing value is to 1, the smoother it is.

We did try to implement dynamically changing the four values through the text box, but was not able to. Every time the text box was implemented to change the values along with the sliders and "plus and minus" buttons, the application would not start as it would crash upon starting. So the text boxes were then left to be updated with the debug menu using a timer.



[H]

In the picture [H], the debug menu is shown. Initially, when first implementing the debug menu, the menu would not update automatically. Through a bit of online research, a conclusion was denoted to adding a timer to the window form to constantly update the text boxes in both the settings and debug tabs. The debug tab displays the values of said variable in the program. The Program Progress text box is used to tell the end-user of what position the application is currently at. There is an enable check box to enable the real time updates to the screen. This is used to make the program run better on the Intel Compute Stick rather than wasting CPU Usage on updating when the program is only needed to run and not debug the program and the Kinect sensor values.

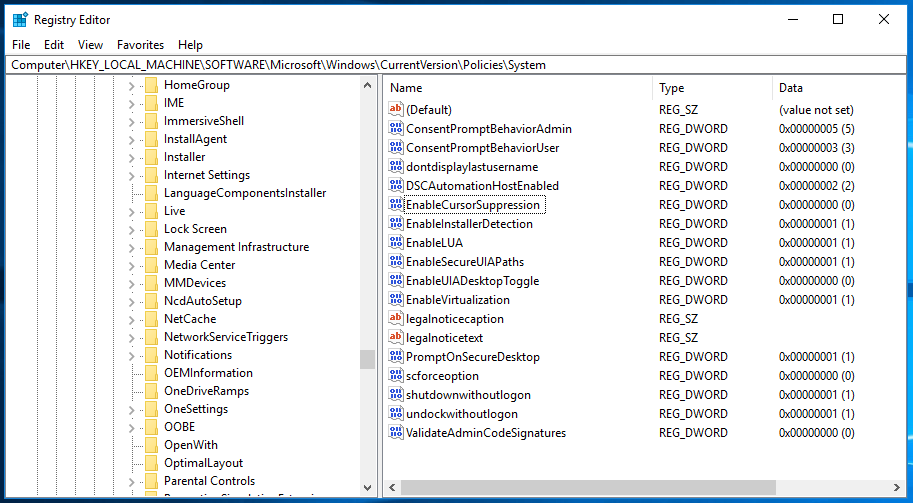
****[I]

When we were finished implementing the debug menu, we tested the program with the intel compute stick to see how the debug menu affected the performance. The performance of the application on the intel compute stick was horrendous compared to the performance of my Laptop. To remedy this problem, we decided to slim down the program to exclude the debug menu, picture [I]. At this point under the GitHub of the "uahsmartmirror" [2], we created another folder to house the "release", "beta", and "original" program source codes. The original source code is taken from and existing GitHub [1], and was included for reference. The release source code is the version that excludes the debug menu. The beta source code is the developmental source code of the release to give more information to the developers to see what the Kinect is reading in from its sensors.

An “Enable” check box was later added to the beta program to reduce the CPU Usage as noted above to free up the CPU with usage with the Kinect on the Intel Compute Stick as shown in picture [I]. The reasoning to this was that the timer\_refresh timer was consuming more of the CPU time than just the stripped version. So this then this checkbox was later implemented.

A few other minor problems we addressed was as follows, the TV sensor being in an awkward position, microphone levels for the voice command recognition, and also how the Intel Compute stick required a physical mouse.

For the TV IR sensor, we moved it from facing towards the back to behind the Kinect facing forward for ease of turning on the TV. For the microphone levels, we checked both the Blu and Kinects microphone and compared both mic levels and was surprised that the Kinect’s microphone had better mic levels. Thus, we set the default microphone to the Kinect’s microphone. Last but not least, we also did a registry hack in the Windows’ Registry to un-suppress the mouse pointer when a mouse is not connected, which alleviated the requirement of the mouse having to be connected to the Intel Compute Stick. The registry key that was changed is as follows: [HKLM\Software\Micfosoft\Windows\CurrentVersion\Policies\System\EnableCursorSuppression=1] and the value was changed to 0. This change can be viewed in the photo [K] below.



[K]

In conclusion, the GUI of the program has been refined to allow anyone who would like to work on this project to be able to see how the Kinect sensors read in a person when navigating through Windows. In addition, a few hardware changes have been made, including, the movement of the TV IR sensor and not requiring the mouse anymore.

References:

[1] Kinect V2 Mouse Control:

Website: <http://tangochen.com/blog/?p=2137>

GitHub: <https://github.com/TangoChen/KinectV2MouseControl>

[2] UAH Smart Mirror

Initial GitHub: <https://github.com/AnoraYunGen/kinectMouse>

Final GitHub: <https://github.com/AnoraYunGen/uahsmartmirror>

[3] Google

Researching from Various Links