Experiments

Tim Visée & Nathan Bakhuijzen

October 2018

1 Preface

In this document, we will go through some experiments we've conducted for the platform we have built during the research minor KB-80. For each experiment we've written a new section describing what it is about, how we conduct the experiment and what our findings are. The general idea of conducing these experiments is to assert whether the current platform is *usable* for the purpose we had originally in mind, namely controlling a generic computer.

Although we've tested the platform quite thoroughly while building it, these experiments are conducted on the final version of the platform we've worked on with properly detected experiments and findings. With this we'll attempt to also answer some seemingly obvious thoughts.

The last part of the document goes over some ideas we came up with when working on on the project, and when conducting these experiments. These ideas may be implemented by others that continue the work on this project.

You may of course conduct similar experiments as we've described along with reading this document. Make sure you read the *Manual* on how to install and use the product.

2 Accuracy & Reliability

The accuracy and reliability of the gesture recognition is crucial to the succes of the platform. If it proves to be insufficient, then the platform is obsolete. No one will use it and time spent developing it will be wasted. If it is accurate and reliable however, then it may prove useful. It may be used to conduct research, and can be developed further.

2.1 Hard-coded gestures

To test our platform, we came up with a list of predefined gestures. These gestures are programmed/*hard-coded* into the platform. This means that the predefined gesture is not jittery to begin with, as could be the case with recorded gestures. This list of gestures can be loaded into the program with the *Load default templates* button in the bottom of the web interface. These gestures are quite simple and show well what is possible with the platform. For this first test, we attempted each gesture 40 times to measure it's detection rate. This, in a randomized order to prevent ourselves from using muscle memory.

Here is the list of hard-coded gestures:

- Straight line
- Circle clockwise
- Circle counter-clockwise
- Big circle clockwise
- Big circle counter-clockwise
- Triangle clockwise
- Triangle counter-clockwise
- Mini square clockwise
- Square clockwise
- Square counter-clockwise

When we tested these gestures ourselves, the results weren't so surprising after working with the system for many weeks. First of, detecting clockwise versus counter-clockwise gestures is not a problem at all. The platform works well for both, which is logical considering how the internals work.

The first few gestures work very well. In fact, all attempts for performing these worked and all got recognized correctly. The *Straight line* gesture even got detected with a slight bent in the drawn line. Both circle variants are included too. These are easy and above all very quick to perform. We figured these could be cool for scrolling through a page. A light disadvantage of the line gestures is that because it is so simple, it did get matched against some false positives. When attempting to draw a large square the line gesture was detected twice.

The triangle gestures were great as well, although we didn't got them right the first time. Out of the total 80 attempts for each of the two variants, 71 got detected properly. Loosening the algorithm might even give us better results.

The square gestures were slightly harder. The normal squares had 33 and 34 correct hits. The small square had 25 hits. These squares have perfectly straight edges and corners. That might make it harder to perform correctly. What's interesting is that performing the small square was harder than the regular sized squares. Jitter in your finger and the sensor does have a larger effect on small figures, that might be what caused it.

Gesture	Attempts	Successes	Success %
Straight line	40	40	100%
Circle clockwise	40	40	100%
Circle counter-clockwise	40	40	100%
Big circle clockwise	40	40	100%
Big circle counter-clockwise	40	40	100%
Triangle clockwise	40	35	87,5%
Triangle counter-clockwise	40	36	90%
Mini square clockwise	40	25	60,25%
Square clockwise	40	33	82,5%
Square counter-clockwise	40	34	85%

In general the detection for these shapes works surprisingly well to our opinion. The data shows that gestures with sharp corners are slightly harder to draw, probably because it's hard to actually perform a gesture with such sharpness above the sensor. Just to note, we also noticed that performing gestures with your non-dominant hand is slightly harder, although we don't have proper data to substantiate this.

2.2 Custom gestures

Because the system allows you to record your own gestures, and the hard-coded gestures might be a little too perfect, we decided to also manually record each gesture on the list to perform the same test afterwards.

Gesture	Attempts	Successes	Success %	Previous %
Straight line	40	40	100%	100%
Circle clockwise	40	40	100%	100%
Circle counter-clockwise	40	40	100%	100%
Big circle clockwise	40	40	100%	100%
Big circle counter-clockwise	40	40	100%	100%
Triangle clockwise	40	38	95%	87,5%
Triangle counter-clockwise	40	36	90%	90%
Mini square clockwise	40	31	77,5%	60,25%
Square clockwise	40	35	87,5%	82,5%
Square counter-clockwise	40	33	82,5%	85%

Yet again, the line and circle gestures got recognized perfectly for each iteration even though the recordings might not have been perfect.

There was one exception though, during the performance of one circle gesture the sensor (or our platform) seemed to get stuck for a little. This was shown in the visualizer. Only half of the circle was shown, and it continued with a straight line down. It was as if the circle was cut in half. Because the visualizer showed that the data was incorrect we decided to drop this erroneous iteration.

The triangle gesture worked a little better. Maybe this is because it was recorded manually it had rounded and less perfect corners making it easier to perform properly. Out of the 80 total attempts 74 got recognized correctly.

The custom square gestures proved a similar result, although the result for the small squares was improved slightly from 25 correct attempts to 31.

2.3 Complex gestures

We also decided to attempt more complex gestures. With more complex we mean longer gestures and gestures with complex and sharp angles. We choose the following figures:

- A 4×1 rectangle
- The number `6`
- The letter `W`
- A clockwise circle with a counter-clockwise circle after it
- The word `hi` in cursive font

Gesture	Attempts	Successes	Success %
Rectangle	40	26	65%
Number 6	40	40	100%
Letter W	40	12	30%
(Counter)clockwise circle	40	38	95%
Word hi	40	1	2,5%

The results are a little less great for these figures. First of all, the rectangle. For the 40 attempts we tried it got 26 hits. Apparently it is harder to recognize a big rectangle as compared to a square as tested in a previous test.

The number `6` was perfect as well. It has a similar shape as the circle from previous tests of course, so this isn't too surprising. We did thought about the following thought during this test: with the current algorithm it is not possible to also define a gesture for the number `9` because it has the exact same shape but in a different orientation.

The `W` shape showed the first signs of trouble. To given an idea, it got detected 12 times out of 40 tries. The algorithm isn't optimized for this shape, apparently.

We think this has to do with the sharp corners this shape has. For each new data point our algorithm compares the current shape to all known (defined) gestures. This is done by walking through both the current and the defined gestures. Once they start to differentiate too much the option for that gesture is dropped. We believe the sharp angles in this character might cause the algorithm to think the difference is too big. You aren't able to draw it perfectly every time, of course. The logic is a bit more complex than this, but this should give a general idea of how it works and what we think is causing the dropped detection rates. The algorithm also has some play when comparing rotations, maybe the algorithm starts comparing the wrong rotation points further into the trace when the scan points start to deviate too much. This could result into inconsistent results as well.

The decision whether a gesture, or rather a trace of rotations, differentiates too much is defined by some constants defined in the `config.rs` file in source-code. We did increase the threshold and some other relevant parameters a little which did seem to improve the results a bit. We didn't investigate further though due to time constraints and reverted the parameters to ensure following tests are consistent. It might be interesting for other groups to take a look at this.

The gesture with two circles worked very well too. No further comment on this.

The word `hi`, of course a very optimistic gesture, didn't work at all. Out of 40 attempts only 1 got recognized correctly. The algorithm is at fault for sure, because the figure did show up correctly in the 2D web visualizer.

One might argue that you shouldn't expect too complex gestures to be recognized correctly, especially not for a first iteration of the project. But we thought we had to try it anyway.

2.4 Conclusion

These results means that our gesture recognition can definitely be improved. We need to improve the gesture recognition in order to implement even more complex gestures. This would all be software improvements, so no new hardware is required for now.

Some properties of the current gesture recognition can be altered in the *config.rs* file as mentioned. By including a larger amount of points in a match group/scope, failure to recognize a gesture will be lower. This is because there will be more points, and more room for deviation from the predefined gesture. Following this approach would allow more false positives though.

3 Gesture variety

For building a proper gesture based system, we think it is important to support as many gestures as possible. The current predefined set is of course lacking. If you're binding gestures to actions on a computer, you probably want to do as much different things as possible with a little difficult gestures as possible. We'll now attempt to define what variety of gestures is possible with our current implementation.

The platform currently has support for tracking gestures based on movements with the tip of the index finger. This already limits the number of gesture possibilities quite severely. You basically need to use your finger as a pen or wand to literally draw a shape.

In addition, recorded sensor data is currently being processed into a 2D representation. This means that it is currently not possible to define 3D aware gestures.

As shown in the previous experiment, large gestures or gestures with sharp corners don't work to well yet with the current algorithm without tweaks.

3.1 Supported

When only defining basic gestures, we could think of the following gesture types:

- Circle
- Line
- Square
- Rectangle, in various shapes
- Triangle
- Simple/distinct numbers: 2, 3, 6, 7
- Simple/distinct letters: C, D, J...
- ...

Each figure can have some variations:

- Various figure scales, larger and smaller variants
- Use both clockwise and counter-clockwise variants
- Scale in one dimension, use an oval instead of a circle

The list is limited, and some variations might collide resulting in false positives. Implementing 10 different gestures is however easily doable, as the built-in list of gestures does confirm. The question of whether the variety is good enough highly depends on the use case for the product.

3.2 Computer control

Now, the initial goal of the platform was to develop a platform that would be able to fully control a computer using only gestures. We currently don't think that that is possible, let alone viable. Yes, scrolling up and down by drawing circles left or right would be awesome. But these gestures don't cover simple tasks like moving the mouse around. It shouldn't be too hard to implement mouse movement based on the location of a hand in 3D space, but this falls out of the scope of this project for now.

There are some problems that would have to be resolved in order for the platform to become viable as a computer controlling solution. We've formulated a list:

- Only simple gestures get recognized very well, limiting the number of usable and reliable gestures

- With a system that is purely gesture based, you'd need to have a lot of different gestures to make it usable so you don't have to go through menus of actions
- Gestures do not support multiple fingers, yet
- New recorded gestures could be jittery, resulting in a low recognition rate
- Execution of computer actions would probably slower than using a keyboard and mouse
- The current implementation is hard to set-up

We know that resolving these issues is a lot to ask for, but if we would be able to solve these problems we could have built a revolutionary new platform.

3.3 Improve

To improve the results of this test, we believe that additional features need to be developed in the platform which does take some time. Think of supporting gestures consisting of multiple fingers or even hands, or supporting 3D gestures. If multiple fingers are combined in gestures, then it could mean a plethora of new possibilities. Improvements would mostly consist of software improvements rather than needing to have improved hardware. Think of changes in the algorithm and processing logic to allow more advanced figures. We'll go over ideas for improvement in the *Ideas* section, the last section in this document.

3.4 Simple applications

So, we don't think it is viable for computer control. There might be applications though with just 4 different simple one-shot actions. Yes, for such applications this could be used perfectly well. Drawing circles, lines and squares works good enough for sure. Think of scrolling through pictures in a sterile environment where touchless interaction is desired.

4 Visual feedback

An interesting aspect for using our platform is visual feedback, and what the effect of using it is on users using our platform. With visual feedback, a user would be able to see on a computer screen how the computer (or rather; sensor) recognizes his hand(s) and/or gestures. The general idea for this test is that showing visual feedback might improve accuracy, as users could adapt the gesture they're performing on what they see the computer recognizes. The sensor isn't super accurate all of the time, or users might go outside the view of the sensor. The visual feedback would immediately show this.

4.1 Visualizers

Two visualizers are available. The first one is part of the LeapMotion software package, and is a 3D visualizer that can be started from the LeapMotion control panel. This visualizer shows a 3D representation of recognized hands above the LeapMotion sensor inside the bounding box of the sensor. Joints are visible in the hand, although our software only uses the position of the index finger tip at this moment.

The second visualizer is the one we've implemented. It is show on the web interface that can be opened after starting the software. This visualizer is quite different, not only in aesthetics, but also in what it shows. Our visualizer is 2D and shows the data that has been processed by our software. The *Manual* explains how we are processing the data, and that we're mapping 3D points from the sensor in traces of rotational data. This is what is rendered in our visualizer, and is essentially what is used for gesture detection. This is therefore *closest to reality*, where *reality* consists of user defined gestures in our software and the recognition system. If gestures show up correctly in this visualizer, the software should recognize them correctly. This same visualizer is used for recording and trimming new gestures.

4.2 Findings

For this test we've tried a lot of different gestures, in a lot of different positions. Here are some of our statements on our findings:

- Drawing circles isn't a problem. The only thing a visualizer might help for is to see whether you're performing the gestures within the view of the sensor, and to see whether your hand is properly recognized.
- Without visual feedback attempting to draw squares, can easily result in drawing rectangles, especially when focussing on the screen instead of looking at your hand. Having the 2D web visualizer helps greatly as you can see whether you're drawing a square properly.
- The 3D visualizer is easier to understand, although the data shown in the 2D visualizer is more consistent. This, in the sense that using the 2D visualizer to guide you to perfectly draw a gesture works better and produces more accurate results than using the 3D visualizer.
- Either visualizer is useful to see whether the sensor is jittering, causing detection failures.
- Properly drawing a gesture with the other hand a gesture was recorded with is virtually impossible without a visualizer, except for shapes that are really simple.
- The 2D web visualizer gives a better sense of how big a user is drawing a gesture. Allowing the user to be more accurate.
- The 2D web visualizer is useful to figure out why a false positive may have been detected as it clearly shows the shape that is used for recognition.

4.3 Conclusion

Based on these statements we can conclude that using a visualizer is a good idea, as it does seem to improve results for most gestures. There are gestures however that don't really benefit from using a visualizer. Such gestures can be used in a situation where a visualizer isn't preferred as it would use up screen real estate.

The accuracy of a user drawing a gesture correctly might improve the more the user uses the system. We haven't tested this.

None of the visualizers notify the user if a drawn gesture is similar to a user defined gesture (but is not a match). Imagine that you're drawing a circle, but too large for it to be detected. The visualizers don't show that your circle is too large. This might be something to improve on.

Note that we, the developers of the platform, have conducted this test. Other users that don't understand the internals of the system might experience this different. The second visualizer in the web interface shows processed data, this might look weird and unintuitive for other users.

5 Detection speed

Another important aspect of the gesture recognition system is the speed, or rather, the delay for detecting a performed gesture. To avoid confusion, we've decided to define the term delay as follows: *The time it takes for the platform to show the name of the correct gesture in the console, after a gesture has been completely performed above the sensor*.

5.1 Counting frames

Since it can be tricky to accurately measure the time difference between an invocation and an action, we've decided to record 20 invocations. With the resulting footage, we'll be able to count delay in frames, with precision accurately enough for our use case. The gesture that we choose was the square, as is has quite distinctive corners.

5.2 Results

After performing the our own tests it was quite clear that delay varies greatly. The *delay* varied from about 5 to 11 frames. With a speed of 60 frames per second, each frame is about 17 milliseconds. This defines a delay varying between 85 to 187 milliseconds. There were two outliers with a delay of 16 to 22 frames which translate to about 270 and 370 milliseconds.

We believe the varying difference has to do with the accuracy of detecting the gesture itself. Imagine you are drawing a perfect square gesture (perfect relative to the square gesture you might have previously recorded) above the sensor, the gesture will be recognized quicker before even fully completing the gesture, as compared to a square motion that differs slightly. Because it's virtually impossible to perform the gesture in exactly the same way every iteration, a difference in detection is indeed expected.

Note that our method of testing latency is still quite simplistic. It doesn't take computer and screen delay into account. Nor does it test what the latency of the used sensor itself is. Our test strictly focusses on the delay between performing the gesture and seeing visual feedback. We performed the test with the visualizer enabled, which was visible on the material we recorded. What's interesting is that it looks like the gesture is instantly (meaning; within the same frame) recognized as the last sampled point of a gesture shows up on the visualizer. This would suggest that the actual latency for detection on the processed data is faster than 17 milliseconds. This is of course not very scientific, but we feel it's an awesome result non the less.

What is important though, is how responsive the detection feels. During these tests, the detection (strictly speaking about detection speed) felt snappy even during the case of those outliers. The cool thing is that after you've performed a gesture you don't have to wait for a detection notification before the next gesture can be performed, that helps quite a lot for it to feel responsive to our opinion when when repeatedly experimenting with the sensor.

The platform doesn't support binding generic actions to gestures to control a computer yet. And until that is tested, it's hard to say whether we'd feel the same when controlling a computer with these gestures. We mention this idea of binding actions later in this report in the *Ideas* section.

6 Ideas

During the project, and during the experiments we've conducted we came up with quite a few ideas. Some of them are things we'd wanted to implement but didn't have the time for it due to time constraints. Other ideas originate from test results we've collected. Here follow a few of them:

6.1 Multiple fingers

The current system only has support for gestures consisting of a single trace, being the trace drawn by the index finger of a hand. The first logical expansion would of course be to support gestures having motions with 5 fingers. Sadly, due to time constraints, we couldn't implement this. This could be a great opportunity to improve the complexity and depth of the platform.

Our guess is that implementing this isn't too hard when using the same detection system as our current platform. The implementation might be as easy as extending our `Hand` structure (in code) to contain 5 traces, instead of just 1. This might be cool to experiment with for other groups taking a look at this project.

6.1.1 Not just finger tips

With the extension described above, the system would still only be tracking finger tips. The sensor we're currently using is able to report the position of all joints in a hand, including the center of each bone and the center of the hand palm. This is easily visible in the 3D visualizer that is provided with the LeapMotion software package. The `leap-rs` wrapper that is currently used doesn't have an interface to obtain this information yet, but this could be implemented without too much effort.

The same might be true for this implementation, just tracking the trace for every joint might be enough. We're of course not sure, that's something cool to test out.

6.2 3D detection

Although we intended to support it at first, 3D gesture detection would be an awesome improvement. The current system uses 2D rotational data to represent user motions. Simply said; because of processing data to this format, the third dimension is lost. We found it difficult to reliably transform a 3D curve into 2D rotation data, and wanted to get the 2D system working first.

Of course, motions in 3D space are transformed into data our platform uses. During this process the depth (Z-axis) is lost though. Drawing a circle sideways would be detected as moving up and down on a straight line. In this system, you might imagine drawing on a virtual plane on the X and Y axis. Supporting the Z-axis as well for a 3rd dimension would open up a lot of new possibilities for drawing gestures in different orientations and would support much more complex gestures.

6.3 Bindable actions

Our idea was to allow users to bind custom actions to recorded gesture templates quite early in the project. This would allow a user to configure the platform to control their computer in a generic way. The implementation can be as simple as binding mouse (and scroll weel) actions, keys and system commands.

Such an implementation would immediately make the project much more viable for use by others (given that it would work properly). This also opens new research questions such as; can gestures be complex enough to support enough gestures for controlling a generic computer?

This idea was also the segway to research on how usable a gesture detection system is for use in medical & sterile environments. Imagine doctors not having to touch a röntgen machine to swipe between pictures, by using touchless hand gestures instead. The implementation of bindable actions should make quite clear whether such a system is usable and reliable.

6.4 Multiple sensors

An awesome extension would be to use and combine multiple sensors in the platform, instead of using a single LeapMotion sensor. The obvious choice would be to use a second LeapMotion sensor to combine data of the two. This could be used to filter jitter, and to fallback if the view for one sensor os occluded.

This is one of the first things we looked into during the project. Sadly, it became apparent that the LeapMotion SDK doesn't support connecting multiple sensors, even though the provided library would suggest otherwise as there are functions available to obtain a list of connected sensors. This forum thread is about this topic, in which users are asking for support to use multiple sensors. In this thread, LeapMotion developers have said this is not possible.

It would be fun to use a different computer, or possible a virtual machine with a virtual USB controller to support connecting another sensor. The data from this sensor would then need to be transmitted to the platform to collect the sensor data. This is however outside the scope of our project, and is a challenge to properly implement on it's own.

Another option is to implement support for other kinds of sensors, such as the XBox Kinect, or something different. Our platform would than need to provide the proper abstractions to aggregate the data from different kinds of sensors into something the platform could use.

7 Final thoughts

With building this platform, and doing these tests we can confirm for sure that the software we've developed is able to detect and distinguish user defined gestures. The system is dynamic and can be used for recognizing figures we would never have thought of.

Although we didn't reach the goal controlling a computer in a generic manner, we do believe we have achieved enough during this project to prove that gesture recognition with dynamic gestures is in fact usable. The question remains of course, whether it's viable to use for computer control. We answered this question with no for now, looking at our implementation. Improvements to the system and the algorithm we've implemented to greatly improve the possibilities though.

The platform provides a good basis for further research and is extendable so it might open doors for other research purposes. There are more than enough things that can be tweaked or improved improved for gesture recognition, more complex input and bindable actions. We've mentioned some ideas above. This allows other groups interested in this project to experiment further, with what is possible and what works. Maybe an implementation can be developed for a simple and concrete application where touchless interaction is a useful solution.

We find it unfortunate that we were not able to reach our initial goal, because the actual implementation of the gesture recognition system took much longer than expected. We had some trouble understanding the true mechanics during the project at first, and did walk into some roadblocks on the way. We like what we've got so far though, and had fun for sure while implementing it. We hope others that might continue this project feel the same and are able to built something awesome.

— Tim Visée & Nathan Bakhuijzen