User Manual

Tim Visée & Nathan Bakhuijzen

October 2018

1 Introduction

The purpose of this manual is twofold:

- To inform research how to use the platform to execute experiments and conduct research
- To inform future developers how to continue developing the platform

1.1 Problem Definition

What physical motions are natural, effortless and easy, in order to control a computer or other digital device?

Questions of this nature can be answered using the *Can't Touch This* platform. *Can't Touch This* aims to be a platform for researchers that would like to conduct research in the field of touchless computer systems. We believe that our platform allows researchers to build a strong foundation for the future of touchless control. Giving researchers the opportunity to conduct research improves the chance for touchless control of computers only seen in futuristic movies and tv shows.

1.2 Motivation

At the start of the KB-80 minor, students were given a choice in the subject of the research. Mister Al-Ers introduced us to a series of subjects, of which the LeapMotion project was the most interesting. The idea behind the LeapMotion project was to create or extend existing software to enable people to control a computer without touching any peripherals, like keyboards and mice.

1.3 Background information

Research in the field of touchless computer systems is motivated by the desire for comparable systems in sterile environments. For example, surgeons often make use of computer systems to aid them during their surgeries by providing crucial information such as CT, MRI and X-ray scans. This is where touchless computer systems may offer a solution. Touchless computer systems allows surgeons to control a system without the need for physical peripherals.

2 Installation Guide

The *Can't Touch This* platform requires the following:

- A computer with the Windows (7+), OSX (10.7+, Lion+) or Linux (kernel 2.6.18+) operating system
- An installation of the LeapMotion SDK
- An installation of the Rust programming language
- The physical LeapMotion device itself
- The *Can't Touch This* platform

2.1 Software Dependencies

The *Can't Touch This* platform is written using the Rust programming language. This means that the operating system that the platform will run on must also support the Rust programming language. Fortunately, Rust runs on all popular operating systems today, shown above in the list of requirements. An up-to-date list of all supported versions can be found on the Rust website. Additonally, the *Can't Touch This* platform requires the LeapMotion SDK to provide all necessary sensor data. Just like the Rust programming language, the LeapMotion SDK can be installed on all platforms.

2.2 External resources

No additonal resources are required to run the *Can't Touch This* platform.

2.3 External development tools

Continuing development of the *Can't Touch This* platform requires basic tools like a text editor or IDE, and a terminal. It is highly recommended to use git, as this was used during development of the platform. Additionally, setting up an CI server may prove useful. Setting up an CI server is beyond the scope of this manual.

3 User Instructions

This chapter gives users instructions on how to use the *Can't Touch This* platform. It assumes that the user has followed the instructions found in the *Installation* chapter. The following instructions will detail how to setup the platform so that you can conduct the *experiments* found later in this manual.

3.1 Usage

- Attach the LeapMotion device using it's provided USB cable
- Start the LeapMotion tracking program provided by the LeapMotion SDK
- Start the *Can't Touch This* platform by running the provided executable program, or run it manually in a terminal (`cargo run`)
- Start up an web browser (Chrome, Firefox, Safari, etc) and navigate to `http://localhost:8000`
- Click on '*Start Recording*'
- Move your physical hand above the LeapMotion device to make a desired gesture
- Once you are done making the gesture, click on '*Stop Recording*'
- The recorded gesture you've just made should be visibly represented on the canvas
- Save the gesture by clicking on '*Save Recording*' and bind the gesture to a predefined action
- Click on '*Recognition Mode*' and make one of the gestures you've made beforehand
- The computer will give positive feedback if the gesture is recognized

4 Requirements

This chapter details the list of open and finished requirements of the *Can't Touch This* platform. The prioritization of this list is not according to the MoSCoW method, because we had to write the entire platform from scratch. This took a long time, with lots of challenges and unexpected issues. We therefore chose to figure out what to do on the fly, rather than planning things out beforehand.

The following requirements have been satisfied:

- Operating System independant
- Web interface for user interaction
- Predefined gestures
- Record new gestures
- Gestures management
- Multi-finger recognition

By using the Rust programming language we inherently met the operating system requirement. This saved us a lot of work and allowed us to focus on the platform's functionality. Similarly, we chose to use a web interface for the user interaction. These decisions saved us a substantial workload.

The following requirements are left unsatisfied:

- Gesture bound actions
- Combining multiple sensors

Gesture bound actions are not yet implemented, because we did not believe this is an important feature. We instead worked on gesture recognition and template management. Instead of binding actions to gestures, we give users visual feedback when the gestures is recognized.

When we started out with the project, we wanted to see if it was possible to combine multiple LeapMotion sensors in order to achieve increased data accuracy. We looked into this, but we found that this was impossible, due to the proprietary SDK. Several websites pointed out that only the developers of the LeapMotion we able to do this.

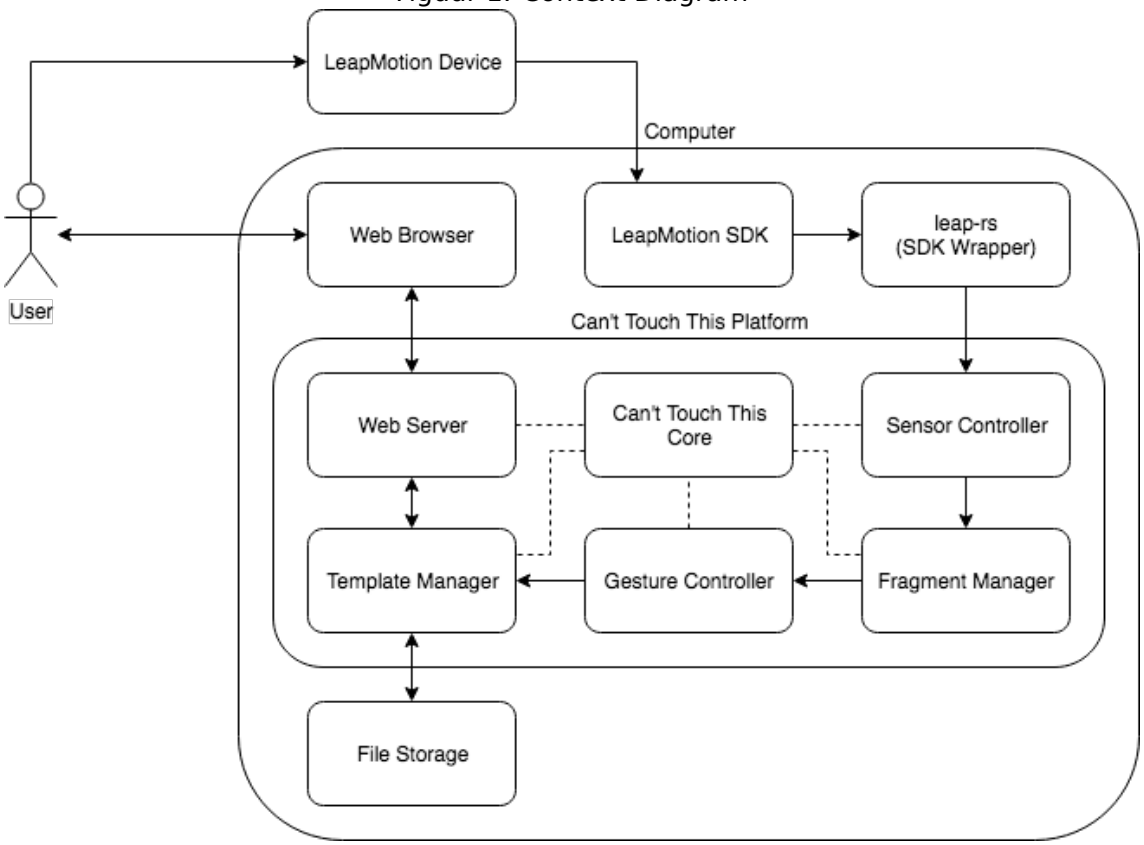
We contacted the LeapMotion team ourselves, but unfortunately they will not continue development on the LeapMotion. New and more interesting projects keep the team occupied, and the leapMotion itself is approaching obsolescence.

5 Architecture Diagrams

In this diagram, the *Can't Touch This* platform displayed globally. It Shows three main objects:

- The user
- The LeapMotion device
- The Can't Touch This platform

Figuur 1: Context Diagram



The user attaches the LeapMotion device to the computer, installs the platform and moves it's hand above the sensor to conduct an experiment. The LeapMotion device captures the data of the hand, and passes it on through the LeapMotion SDK, to the leap-rs wrapper. This leap-rs wrapper maps all functions made available through the LeapMotion SDK to the Rust programming language. Normally, a platform like *Can't Touch This* would have to be programmed using the C programming language, as this is the language the SDK is written in.

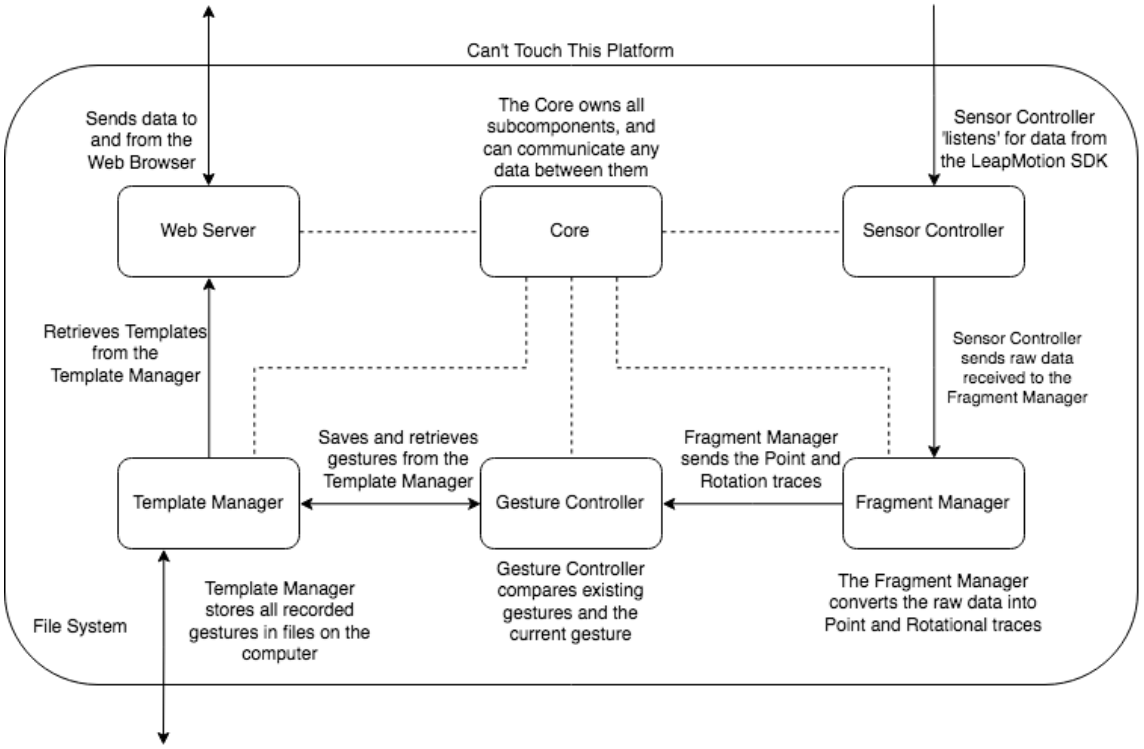
The leap-rs wrapper enables the SDK functionality in our platform, which we use in the Sensor Controller. The Sensor controller is our gateway of information, of our bits and bytes. The Sensor controller passes this data on to the Fragment Manager, which records all data and converts it into Points, Rotational Points (*RotPoints*) and Traces of both kind. It even improves the recorded points in the trace by sampling them. Sampling is . . .

After the conversion and sampling, the Gesture Controller receives the data and compares this to existing gestures, stored in the Template Manager. The Template Manager compares the received gesture to the existing ones stores on the system. The gestures are saved in text files, as the data is not that complex. If the system matches the current gestures with an existing one, it will give positive feedback through the

web interface. The user will see the positive feedback and know that that gesture is working well.

The web interface also retrieves all the stored gestures for the user to review. It can add and delete gestures as the user wants, except the pre-defined gestures.

Figuur 2: Context Diagram



6 Test Report

6.1 Code Quality

6.2 Existing Tests

The platform currently has a few unit tests that cover basic operations, such as the conversion of a `Point` to a `PointTrace`, and more. We also have set up a few tests to cover the comparison of traces, such as straight lines and curves. Below you can see the a unit test for a straight line code:

```
1  #[test]
2  fn straight() {
3      let points = PointTrace::new(vec![
4          Point3::new(0.0, 0.0, 0.0),
5          Point3::new(1.0, 1.0, 0.0),
6          Point3::new(5.0, 5.0, 0.0),
7      ]);
8
9      let expected = RotTrace::new(vec![RotPoint::new(0f64, 2f64.sqrt())]);
10
11     assert_eq!(points.to_rot_trace(false), expected);
12 }
```

Listing 1: Straight line unit test

This unit test creates `points`, a trace of `Point3`'s, and compares it to `expected`, a `RotTrace`.

The `PointTrace` on line 3 contains three points that travel the same distance at every step. The platform recognizes this as a straight line, as there is no change in the trajectory. The variable `expected` then gets assigned a `RotTrace` that contains only one `RotPoint` of 0 degrees. This is correct, as the line drawn on line 3 is straight.

```
1  #[test]
2  fn corner() {
3      let points = PointTrace::new(vec![
4          Point3::new(0.0, 0.0, 0.0),
5          Point3::new(0.0, 5.0, 0.0),
6          Point3::new(5.0, 5.0, 0.0),
7          Point3::new(5.0, 0.0, 0.0),
8          Point3::new(0.0, 0.0, 0.0),
9      ]);
10
11     let expected = RotTrace::new(vec![RotPoint::from_degrees(-90.0, 5.0); 3]);
12
13     assert_eq!(points.to_rot_trace(false), expected);
14     assert_eq!(
15         points.to_last_rot_point(),
16         Some(RotPoint::from_degrees(-90.0, 5.0))
17     );
18 }
```

Listing 2: Corner unit test

The `corner` test is a little more complicated than the `straight` unit test. It creates a `PointTrace` with points that represent a 2D square. This motion first moves to a point, 5 units onto the `y` axis. After moving on this axis, it moves 5 units on the `x` axis. It then concludes the motion by moving back 5 units on the `y` and `x` axis, respectively.

The `RotTrace` on line 11 contains three `RotPoint`'s of `-90` degrees. This is because a `RotPoint` calculates itself based on three `Point`'s. In the case of this unit test, it takes three collections of three `Points`: `1-3`, `2-4` and `3-5`. For each collection, it calculates 2 angles: `1-2` and `2-3`. It then compares the difference between these angles, and that will be the `RotPoint` for those three points. This will produce three `RotPoint`'s of `-90` degrees.

6.3 Known Bugs

- *Can't Touch This* may crash upon running the release version of the exectable
- On macOS, the LeapMotion device may never give data to begin with
- On macOS, the LeapMotion device may stop recording data randomly
- On macOS, the application may not run well when minimalizing the backend application