Cranfield University

Computational and Software Techniques in Engineering

Gruop Project

Assignment

Authors:
Benjamin Deguerre
Wojciech Jonczyk
Alix Kamano
Anna Zaporowska

Supervisor:
Dr. Chi chun gilbert Tang

20 April 2017

Contents

1 Introduction

The aim of this project was to develop an application that would allow users to write using a robot arm. Thus, we decided to focus on making our application user-friendly and performant.
To do so, we implemented 2 modes. The first one allows us to use gestures to write letters by letters. The second mode is designed so the robot's arm follows the finger of the user.

The selection of mode is done as the start of the application but can also be done at every moment while using the application through *master gestures*.

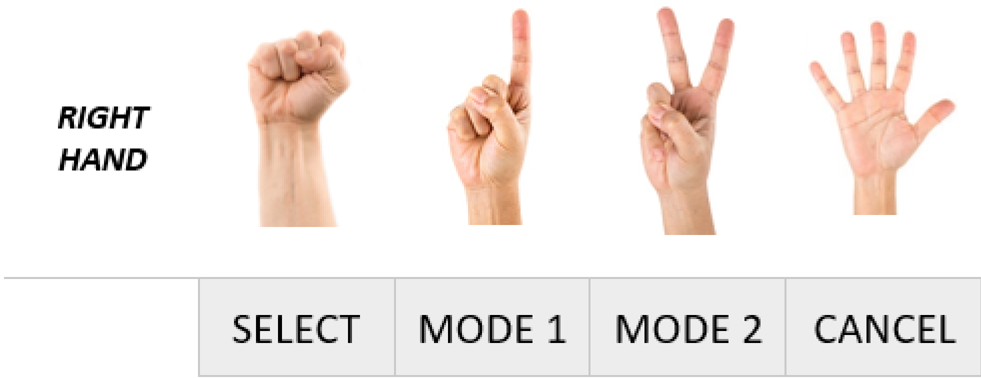

Figure 1: Mode selection - set of gestures

2 Conception

This section will detail the conception and the class diagram of our project. The following graph shows the different classes we used in the project.

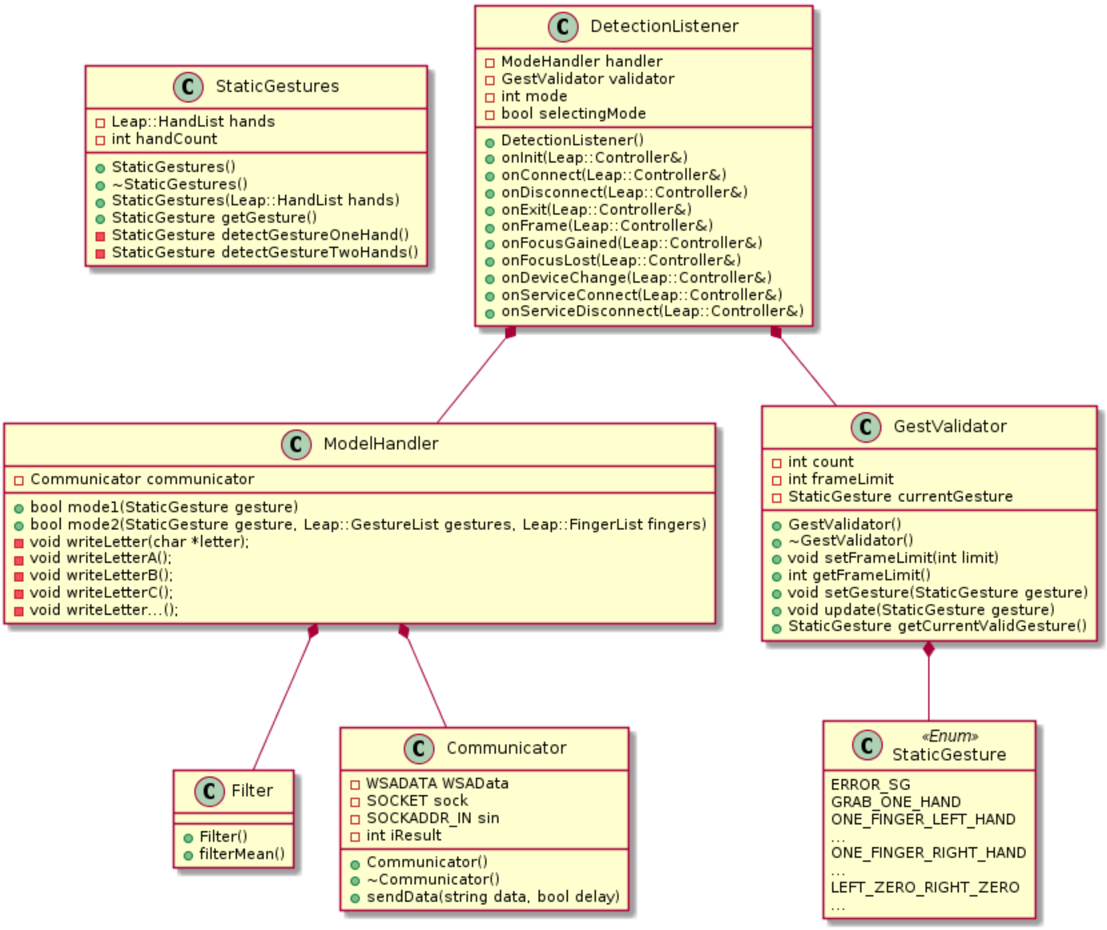

Figure 2: Class diagram

2.1 StaticGesture and StaticGestures

StaticGesture (without s) is not really a class, it is an enum that holds all of the static gesture that are possible to detect. The main role of using an enum is to avoid having to handle the value of each gesture and make it more user friendly. It also simplify the use within the functions for the developper.

StaticGestures (with an s) is the class which allow the user to detect the StaticGesture within a frame. To make it lighter a simple set of hand (Leap::HandList) is required when the object is created. A new object is required to be created each time a new HandList is used. This last part is one of the possible improvement on the code (even if the creation of the new object make sense and is viable).

2.2 GestValidator

As far as StaticGestures is concerned, a new gesture is possibly detected after each new frame. Since this is not usable for the user, the class GestValidator was added. This class allows to set a number of frame on which the the gesture needs to be in order to be validated. Within this class, two functions need to be detailed, update and SetGesture. Their aim can seem identical when they are not, update allows the user to update the gesture within the GestValidator object i.e 11th gesture is CLOSED_HAND whereas setGesture set the gesture count to 0 i.e 1st gesture is now CLOSE_HAND (and if update is used after that we have 2nd gesture is ...).

2.3 DetectionListener

This class is one of the simplest one, it is one of the class given within the LeapMotion api, it allows the user to define what will happen when a certain event is triggered. Here we use it to handle the mode selection (one or two). And to update the detection of the gesture.

2.4 ModeHandler

This is the class used to handle the different mode which are available within our program (1 and 2). The user only need to call to the function modeX to handle the required mode. The class will handle the communication with the robot through the Communicator (see below). Each previous state is stored and this class should not be re-instansciated after each new frame.

It is within this class that the filter is used (mode 2) in order to smoothen the output when communicating with the robot.

2.5 Communicator

This is thee class that handle the TCP communication. It is a simple class which will connect to the robot and then send the data through the function sendData(). This last function allows the user to set a delay or not in order to give the robot some times to carry out the action that is required.

3 Mode 1 - Static mode

3.1 Mode description

The first mode allows the user to select and write a proper letter. In the English alphabet there are 26 letters, so to be able to use all of them, a lot of gestures had to be specified and implemented. Our aim was to do this in the simplest way possible, because a too complicated set of movements and signs would have not been user-friendly and straightforward. The scheme is shown in figure 3.

Figure 3: Letter selection - set of gestures

Every letter and command has its own and unique command. **Differentation is done by counting the number of extended fingers in each hands**. It is a very clear and easy to learn set of gestures. For instance, when only one finger in the left hand is extended and at the same time, the right hand is fully open, it means that letter "J" should be drawn. The system works well when the Leap detects both hands of the user.

However, sometimes the wrong letter was selected. To avoid wrong comunication with the robot, two additional gestures were added - command "Accept" to send the proper letter to the robot and command "Cancel" to reset the current letter and restart the selection. We also wanted to be able to write full sentences. Thus, we decided to implement gesture to draw dot and the space character. It is easy to see in the Figure 3 that there are 6 more possible gestures that are not used. The set of gestures can then be easily extended.

3.2 Letters library

Additionally to the sign alphabet, a library of letter has been created. This library is, in fact, a set of movements which have to be done by the robot in order to write each of the letters. Each set of movements includes moving in all out of three dimensions (x, y, z). Letters library is built in such way that every letter is written in a rectangle of sides length of b and a, and left down corner coordinates of (c, d). After finishing movements for each letter, last move is always created to obtain new starting position for next letter, so the point (c, d) is updated with every new letter. It allows robot to write letters next to each other, which makes writing words or sentences possible. Starting point (c,d) as well as z1 and z2 (z dimension for robot being up, as a position when marker is not touching the surface, and down in writing position) and sides lengths of a and b might be specified by user at the beginning of this program, which makes it more universal and multifunctional.

```
void ModeHandler::writeLetterA() {
    std::ostringstream ss;

    ss << "movel(p[" << c << ", " << d << ", " << z2 << ", 0, 3.14, 0])\r\n";
    communicator.sendData(ss.str(), true);

    ss.str(std::string());//clearing the output
    ss << "movel(p[" << c+(b/2) << ", " << d+a << ", " << z2 << ", 0, 3.14, 0])\r\n";
    communicator.sendData(ss.str(), true);

    ss.str(std::string());//clearing the output
    ss << "movel(p[" << c+b << ", " << d << ", " << z2 << ", 0, 3.14, 0])\r\n";
    communicator.sendData(ss.str(), true);

    ss.str(std::string());//clearing the output
    ss << "movel(p[" << c+b << ", " << d << ", " << z1 << ", 0, 3.14, 0])\r\n";
    communicator.sendData(ss.str(), true);

    ss.str(std::string());//clearing the output
    ss << "movel(p[" << c << ", " << d+(a/3) << ", " << z1 << ", 0, 3.14, 0])\r\n";
    communicator.sendData(ss.str(), true);

    ss.str(std::string());//clearing the output
    ss << "movel(p[" << c << ", " << d+(a/3) << ", " << z2 << ", 0, 3.14, 0])\r\n";
    communicator.sendData(ss.str(), true);

    ss.str(std::string());//clearing the output
    ss << "movel(p[" << c+b << ", " << d+(a/3) << ", " << z2 << ", 0, 3.14, 0])\r\n";
```

Figure 4: Example code from letter library

Functions *writeLetter* are constructed in such way that the whole command is send to the robot controller, in this case it is the command *movel(p[x, y, z, 0, 3.14, 0]*. This kind of communication was chosen because of easier access and possibilities of changing coordinates of starting point (c, d), which is a crucial part for project main idea and was not possible or much more complicated to finish using other methods of working with robot. What is more, function *movel* was chosen because of its functionality this function makes robot move in straight line, without changing position of robots head (which is the main difference between other command *movej* which reaches the target position in the shortest possible way, which is not necessarily the straight line). After sending the command delay between sending next command is given in order to allow robot to finish the first task, then the output is cleared, thus next command may be send. Without delay, robots controller after receiving pack of commands would perform only the last one, which means that the idea of using delay is crucial for program operation. Example function for writing letter is given in the Figure 4.

Result of this set of movement is given in Figure5, showing the letter A written by the robots arm after the Leap Motion recognised, first the gesture responsible for enabling mode 1, then the gesture responsible for letter A and, finally, the gesture of acceptance. Inaccuracies visible in the Figure5 result from inappropriate marker placement and not because of the program. Those could be improved in case of other program tests.

Figure 5: Example code from letter library

Prepared letters library is only one example of possible libraries which could be created for this kind of program. Since designed gestures are meant to be simple and intuitive for any user, they can be used with other libraries, such as handwritten letters, small letters, numbers and any non-Latin alphabet, etc., which makes this program more universal and user-friendly. Set of gestures are not too difficult to obtain, but need time to be prepared correctly. The main advantage is that once they are written, there is no need to worry about robots movements after launching the program. To make it easier for user, in case of future work, additional application could be proposed, which would automatically prepare set of gestures and save them in programs memory.

4 Mode 2 - Hand tracking

adasdasdadad

5 Conclusion

This application goal was to implement writing and drawing with a robot arm, controlled by user gestures on a Leap Motion. To be able to control the writing, a set of gestures has been implemented. In order to have an application as much user-friendly as possible, those gestures used are static and very simple.
As of now, the application implements both the writing and the drawing functions successfully.

However, some improvements are possible on this application. For example, in our alphabet, we have gestures that do not correspond to any letter. Those gestures could be used to switch between upper-case and lower-case letters.
Since the application uses a library containing all movements for writing the letters, it could be possible to develop other libraries that would allow the user to use non-Latin character (such as Chinese characters, the Korean or the Arabic alphabet, etc..).

As the application handles dynamic movements from the user with the drawing function, one possible improvement could be implementing the English Sign Language. The user would then be able to write without having to learn another sign alphabet.
Moreover, in the drawing function, implementing gestures allowing the user to raise and lower the robot arm (to stop or start drawing), would greatly improve the user-friendliness of the application.

Finally, as described earlier, there is some delay applied when communicating with the robot. That delay could be optimized in order to attain a faster writing of the robot and avoid frustration from the user.

To conclude, the development of this application was focused on its use. The final result is functional and user-friendly. However, some improvements could be made in order to make the application more diverse and attractive.