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

10 December 2016

Contents

1 Introduction

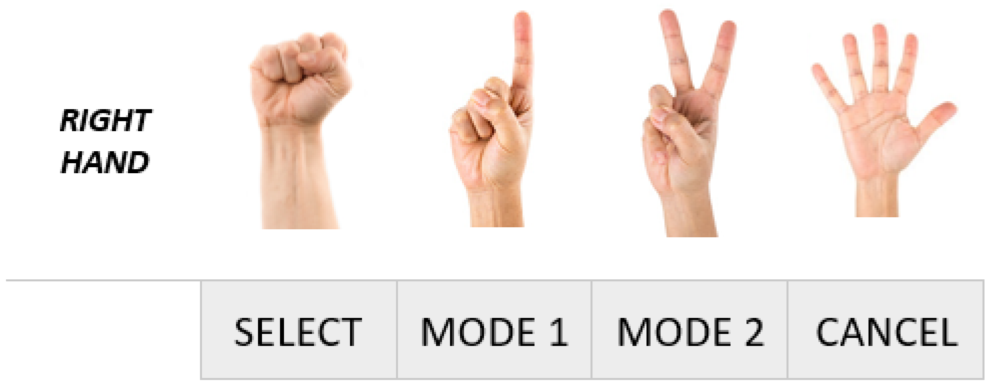


Figure 1: Mode selection - set of gestures

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.

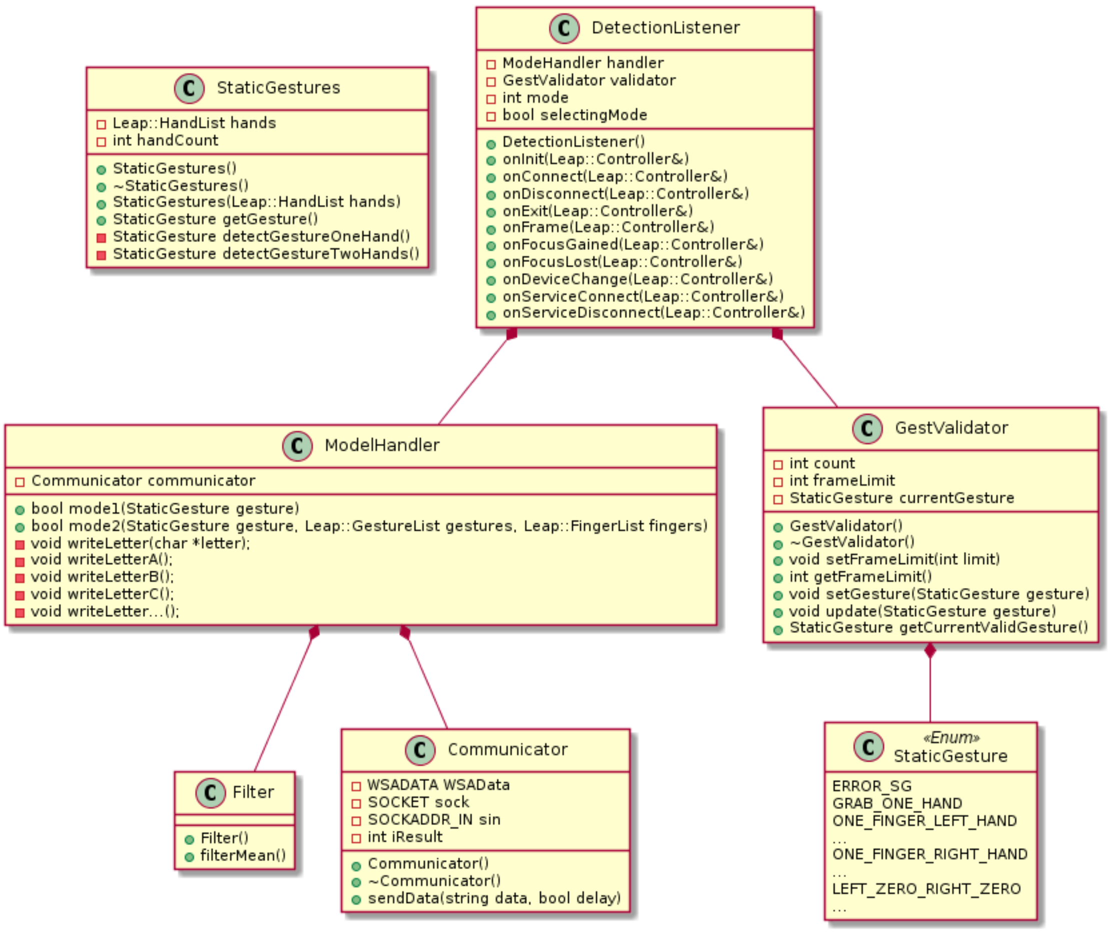

StaticGestures
Leap::HandList hands
int handCount
StaticGestures()
~StaticGestures()
StaticGestures(Leap::HandList hands)
StaticGesture getGesture()
StaticGesture detectGestureOneHand()
StaticGesture detectGestureTwoHands()
DetectionListener
ModeHandler handler
GestValidator validator
int mode
bool selectingMode
DetectionListener()
onInit(Leap::Controller&)
onConnect(Leap::Controller&)
onDisconnect(Leap::Controller&)
onExit(Leap::Controller&)
onFrame(Leap::Controller&)
onFocusGained(Leap::Controller&)
onFocusLost(Leap::Controller&)
onDeviceChange(Leap::Controller&)
onServiceConnect(Leap::Controller&)
onServiceDisconnect(Leap::Controller&)
ModelHandler
Communicator communicator
bool mode1(StaticGesture gesture)
bool mode2(StaticGesture gesture, Leap::GestureList gestures, Leap::FingerList fingers)
void writeLetter(char *letter);
void writeLetterA();
void writeLetterB();
void writeLetterC();
void writeLetter...();
GestValidator
int count
int frameLimit
StaticGesture currentGesture
GestValidator()
~GestValidator()
void setFrameLimit(int limit)
int getFrameLimit()
void setGesture(StaticGesture gesture)
void update(StaticGesture gesture)
StaticGesture getCurrentValidGesture()
Filter
Filter()
filterMean()
Communicator
WSADATA WSAData
SOCKET sock
SOCKADDR_IN sin
int iResult
Communicator()
~Communicator()
sendData(string data, bool delay)
«Enum»
StaticGesture
ERROR_SG
GRAB_ONE_HAND
ONE_FINGER_LEFT_HAND
...
ONE_FINGER_RIGHT_HAND
...
LEFT_ZERO_RIGHT_ZERO
...

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 not to have 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.

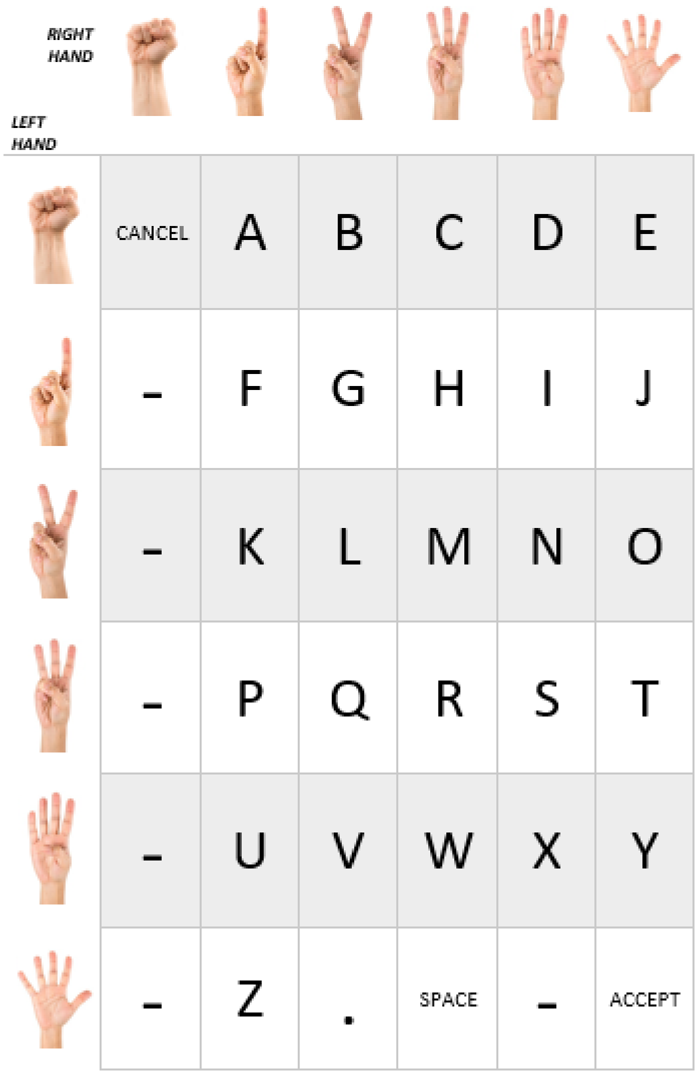
It is within this class that the filter is used (mode 2) in order to smoothen the output when communicating with the robot.

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 - Writting letters

The first mode allows user to select and write proper letter. In 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 as simple as possible, because too complicated set of movements and signs could have not been user-friendly and straightforward. The scheme was shown in the Figure 3.


RIGHT HAND
LEFT HAND

CANCEL	A	B	C	D	E
-	F	G	H	I	J
-	K	L	M	N	O
-	P	Q	R	S	T
-	U	V	W	X	Y
-	Z	.	SPACE	-	ACCEPT

Figure 3: Letter selection - set of gestures

Every letter and command had its own and unique command. **By counting number of extended fingers in either hands differentation was done**. It was very clear and easy to learn set of gestures. For instance, when in the left hand only one finger was extended and in the same moment the right hand was fully open then it meant that letter "J" should have been drawn. The system worked well only when leap was detecting both hands.

Sometimes wrong letter was selected and to avoid wrong comunication with the robot two additional gestures were added - command "Accept" to send proper letter to the robot and command "Cancel" to reset current letter and repeat selection. We wanted also to write full sentences. Thus we decided to implement gesture to draw dot and one more to make space between two words. It is easy to see in the Figure 3 that there are 6 more possible gestures that were not used. That set of gestures can be easily extended.

4 Mode 2 - Hand tracking

adasdasdadad

5 Conclusion