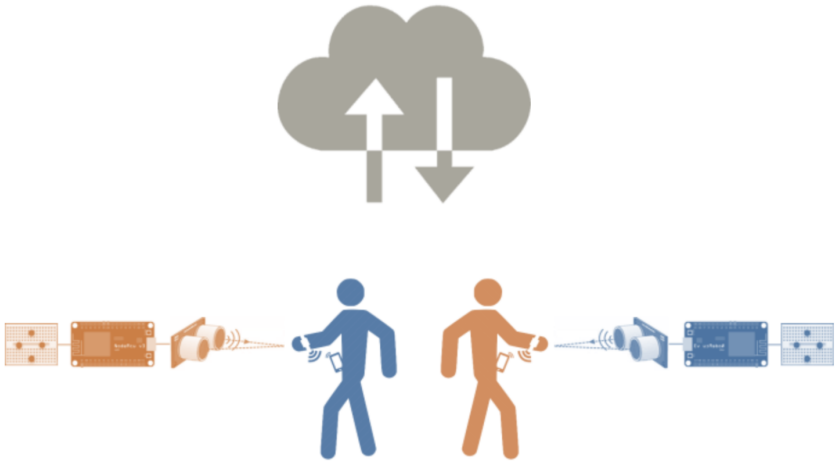
MALMÖ UNIVERSITET

Gesture-based Interaction with Smart Objects



Louay Khalil - AG0903
Benjamin Sejdic - AF5951
Sebastian Hoggard - AD4767
Yurdaer Dalkic - AF5589

Gesture-based Interaction with Smart Objects

1 Introduction

Hand gesture in technology is becoming more common as the new phenomenon Internet Of Things is growing bigger and bigger. Technology developers aim to make people more free whilst using smart devices. It means that simple hand gestures could be enough to control several devices at home or at work.

In this project, we are four students and we will implement a hand-gesture recognition system for interaction with smart objects. Using a smart wristband we should be able to send data to an Android device forward to a cloud and finally to control an Arduino-board(the smart object). The control of the Arduino-board consists of different hand gestures mirrored back as lights turned on or off.

2 System architecture

The system of this project, figure 1, contains a smart object in the shape of a NodeMCU micro-controller including 4 LEDs to show the detected gestures, and an ultrasound proximity sensor to detect the presence of a user. Whenever the user performs a gesture, an Android application detects the gesture and communicates with the smart object through the cloud service. The smart object should then light up the LED that is associated with the detected gesture. There are 4 LEDs for four directions, up, down, left and right.

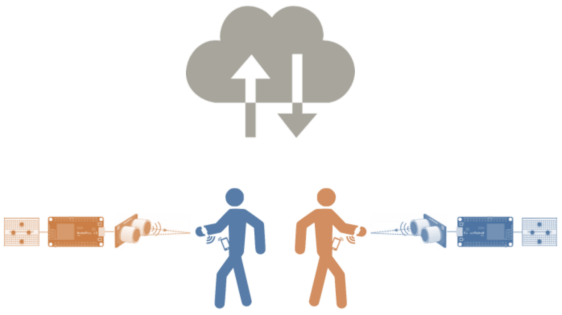


Figure 1. System architecture.

2.1 Arduino NodeMCU

The smart object is basically a NodeMCU micro-controller, figure 2, wired to an ultrasound proximity sensor and 6 different LEDs. NodeMCU is an open source IoT platform which includes firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware that is based on the ESP-12 module.[1] To make the NodeMCU controlled with a hand gesture, it needed to be connected to a cloud through WiFi connection.

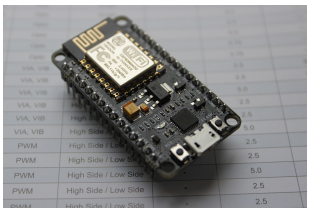


Figure 2. Single-board microcontroller NodeMCU.

4 LEDs are used to visualize the detection of different gestures. 2 other LEDs are used, an amber one for an established WiFi connection and another but blue one as a signal, detecting user presence in front of the ultrasound sensor(or the smart object itself). Figure 3 shows the smart object as a complete unit.

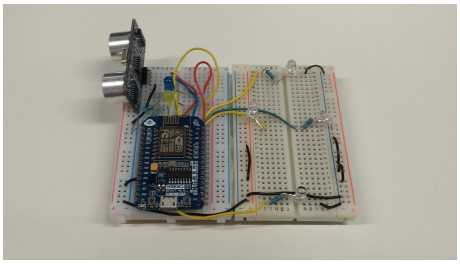


Figure 3. The smart object.

2.2 Android and Wristband

The Android application creates a Bluetooth communication channel with the wristband and tries to establish a connection with the server (MQTT). The phone and the wristband must be paired with each other beforehand to be able to create the channel because there is no option to do so in the application. Otherwise the application will only try to listen for server messages. If, however, a channel is established, the phone will read and process the data from the wristband to determine which gesture has been performed and to finally send it to the server.

The wristband consists of accelerometer and gyro sensors. When performing a gesture it will output 6 values, 3 values for the Z, X and Y axis from the accelerometer and 3 values from the gyro for the same axis. The wristband is running on a frequency of 30Hz, which gives 180 values per gesture (30 samples * 6 values).

The user interface is created in such a way that its intention is to give a higher output of user friendliness and to

[1] https://en.wikipedia.org/wiki/NodeMCU

let the user react to events immediately for example on a disconnection. It consist of 2 text labels which are showing the status of the Bluetooth and server connection.

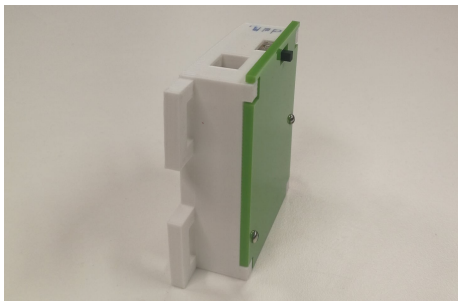


Figure 4. The wristband used in the project.

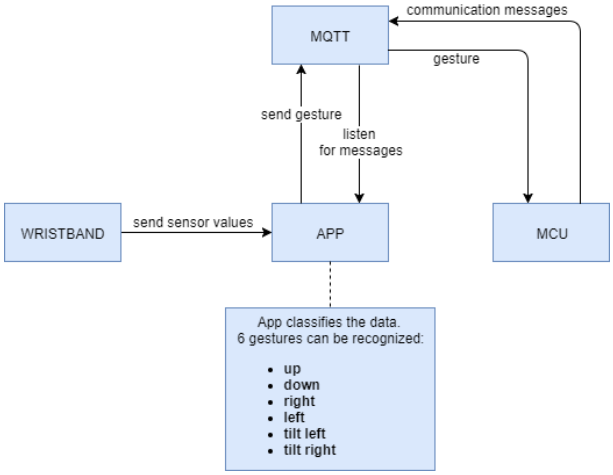


Figure 5. System overview

2.3 Server

The NodeMCU and android application communicate with each other via a server that implements the "Message Queueing Telemetry Transport" which is a machine-to-machine protocol. If offers simple communication between multiple devices as it is using a publish and subscribe pattern.[2] That means whenever a device publishes a message, for example when a sensor publishes data, a client or another device can subscribe to a particular topic where the data/message from the sending device is channelled. A so called 'broker' takes care of these things and keep tracks of all the received messages, filters and distributes them to subscribed clients.[3]

2.4 Protocol and Interoperability

The main purpose for the Android device, or Android application, is to detect a gesture, determine what gesture and send it forward to the smart device. In order to have interoperability between two different devices and different wristbands it is essential to follow one and same protocol. Basically both smart devices need to receive handshakes from the 2 wristbands trying to send data.

The MQTT cloud service uses one or more topic levels. It is then important that all devices use same topic base level[4]. In the case of this project it is IoT/. The wristbands use 2 level topic with the same base as mentioned earlier. The smart devices listen to all messages with the base IoT/ before the handshake. Once the handshake is confirmed the smart device will then start listening only to messages coming from the handshakes 2 level topic. When connection with that handshake is lost the smart device returns to start mode where it again listens to messages coming from any topic with the base IoT/.

[2]http://mqtt.org/faq
[3]brijeshthumar, 2017-04-21, http://androidkt.com/android-mqtt/
[4]https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices

3 Data handling

The wristband is connected to an Android device via Bluetooth. Once a movement or a gesture is performed the data of that gesture is sent to the Android device to be processed. In order to do that the wristband had to be trained for all different gestures, up, right, down, left, rotate right and rotate left.

3.1 Values and attributes

Since the wristband is set on window size 30 and frequency 30, it is known that the gesture movement time is 1 second. The trained data was obtained from 4 users times 30 tries times 6 gestures. For each try, data is gathered for AccX1, AccY1, AccZ1, GyrX1, GyrY1, and GyrZ1. The Acc stands for acceleration in the direction labeled with the the letter right behind the Acc. Gyr stands for the rotation in the direction labeled with the letter right behind the Gyr. With the size 30 and the frequency 30 the result is the total of 180 attributes, plus one extra for the label describing or labeling the gesture. But in live action the last attribute labeling the gesture will be empty until our classification method, with the help of the generated 180 attributes, can determine what gesture was performed.

3.2 Train data classification

The implementation of the system in this project used a classification called J48 Decision Tree because it gave highest accuracy (98%) comparing with the other classifications. A decision tree works its way through nodes (equal to attributes from the trained data), to compare with the attributes from the unidentified or unlabeled incoming data. The terminal nodes tell us the final value (classification) which determines the gesture type. Using J48 classification, as figure 6 shows, the first value to check, is the GyrY1, if greater than 84878

or less than equal to 84878, will give either rotate-right or will lead to other nodes with other results.

```
GyrY1 <= 84878
|   GyrY1 <= -207317: rotate_left (119.0)
|   GyrY1 > -207317
|   |   AccX6 <= -300: up (118.0)
|   |   AccX6 > -300
|   |   |   AccZ5 <= -60
|   |   |   |   GyrY1 <= 10914: right (108.0)
|   |   |   |   GyrY1 > 10914
|   |   |   |   |   AccX11 <= -2: right (3.0)
|   |   |   |   |   AccX11 > -2: left (3.0)
|   |   |   AccZ5 > -60
|   |   |   |   GyrX9 <= -20414: left (103.0)
|   |   |   |   GyrX9 > -20414
|   |   |   |   |   GyrX1 <= 32987
|   |   |   |   |   |   AccZ7 <= 160
|   |   |   |   |   |   |   AccY4 <= -3: down (123.0/1.0)
|   |   |   |   |   |   |   AccY4 > -3: up (3.0)
|   |   |   |   |   |   AccZ7 > 160: right (8.0)
|   |   |   |   |   GyrX1 > 32987: left (14.0)
GyrY1 > 84878: rotate_right (121.0/1.0)
```

Figure 6. 10-fold cross-validation.

As mentioned earlier the end nodes determine what gesture was performed. Figure 6 is better explained in the shape of a tree in Figure 7 as shown below.

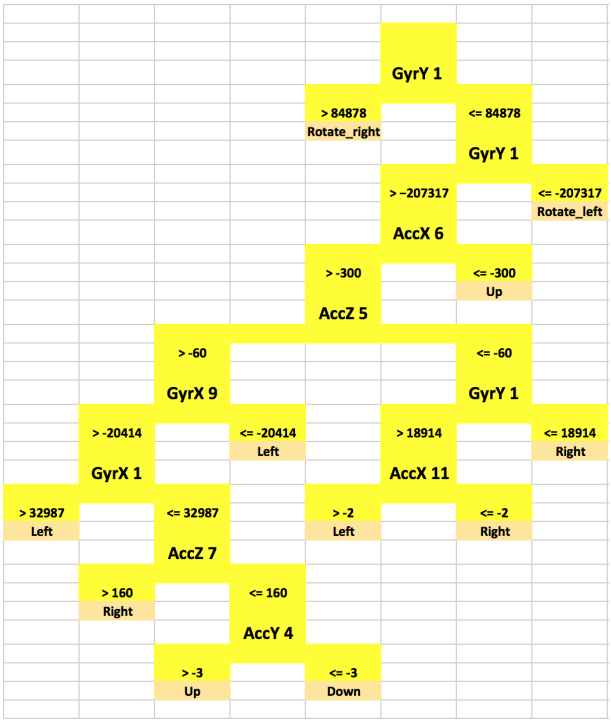


Figure 7. J48 tree.

3.3 Test data classification

With the J48 classifier, another classification was done as a test, on a set of test data. In that set a total of 60 gestures were performed. But 2 rows did not have all 180 attributes and was unlabeled and for that reason were deleted from the data set. At last there were just 58 instances and the performance was outstanding. This set of test data gave an accuracy of 100 %, as seen in figure 8 below.

```
=== Summary ===

Correctly Classified Instances          58               100
Incorrectly Classified Instances         0                 0
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances               58

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Mea
                 1.000    0.000    1.000      1.000    1.000
                 1.000    0.000    1.000      1.000    1.000
                 1.000    0.000    1.000      1.000    1.000
                 1.000    0.000    1.000      1.000    1.000
                 1.000    0.000    1.000      1.000    1.000
                 1.000    0.000    1.000      1.000    1.000
Weighted Avg.    1.000    0.000    1.000      1.000    1.000

=== Confusion Matrix ===

  a  b  c  d  e  f   <-- classified as
  9  0  0  0  0  0 |  a = up
  0 10  0  0  0  0 |  b = down
  0  0 10  0  0  0 |  c = left
  0  0  0  9  0  0 |  d = right
  0  0  0  0 10  0 |  e = rotate_right
  0  0  0  0  0 10 |  f = rotate_left
```

Figure 8. J48 tree.

3.4 Live gesture recognition

The final test was do a live gesture recognition and the result was better than ever expected. In total there were 60 gestures performed, 10 for each direction. Out of these 60 there was an accuracy of 93 %.

Table 1. Confusion Matrix of live recognition

	a	b	c	d	e	f
a = Up	9	1	0	0	0	0
b = Down	0	7	3	0	0	0
c = Left	0	0	10	0	0	0
d = Right	0	0	0	10	0	0
e = R.Right	0	0	0	0	10	0
f = R.Left	0	0	0	0	0	10

4 Evaluation

To evaluate the operation of our system, all one has to do is check that every step of the communication goes through to the correct device and elicits the correct response. A broadcast to "INITIATE" should encourage every connected Android device to reply with its own unique topic. The ESP should respond to the incoming message, and thus the pairing handshake is complete. Every gesture recognised by the Android device after the pairing handshake should arrive as a message to the ESP. When the ESP's range sensor detects that nothing is in range, it should send a "DISCONNECT" message to the paired Android device. All of these things happen most of the times we try the system.

The ESP device's WiFi-connection is not very solid. Quite often it seems to be disrupted by the user activating the device, so the user has to back away and try again.

5 Reflection

The idea behind this project was to tie together all knowledge and experience obtained during the course of Internet Of Things And People at Malmö University, and use it all in one single project. It was very useful because it gave a deeper understanding of the subject itself as such. The main challenge presented itself with the Android implementation of gesture recognition from the data sent from the wristband. All other aspects were already tried and known earlier during the course.

6 Conclusions

One could say that this project was successful in all its aspects. We were able as a group to produce a fully functional smart-device-product controlled with just a hand gesture. The project was all about performing some gestures where sensors sent the data to an Android device to be translated into useful commands that were sent forward to the smart object via a cloud service. These commands dictated which of the 4 LEDs should be lit.

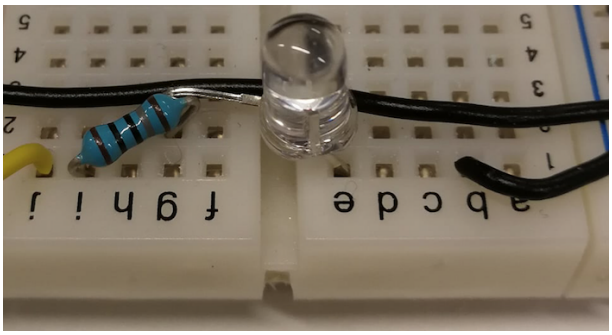


Figure 9. One of the LEDs used in the project.