

AALBORG UNIVERSITY

Context-Aware Home Automation Using Gestures on a Wearable

Authors:

Kasper Lind Sørensen
Simon Binderup Støvring

Advisor:

Brian Nielsen

February 2016 – June 2016



Title:

Context-Aware Home Automation Using
Gestures on a Wearable

Theme of project:

Master's Thesis

Project period:

1st February, 2016
to
8th June, 2016

Participants:

Kasper Lind Sørensen
(klsal1@student.aau.dk)
Simon Binderup Støvring
(sstavr11@student.aau.dk)

Advisor:

Brian Nielsen
(bnielsen@cs.aau.dk)

Abstract:

This report presents a context-aware system that enables users to control a smart home using gestures. Bluetooth Low Energy beacons are used to position a user wearing a smartwatch indoor. Gesture recognition is performed on the smartwatch. When a gesture is recognized, the system attempts to recognize the context the user is situated in and based on the context, the smart devices in the smart home change to an appropriate state. We use the gesture performed by the user and the position of the user as *contextual information*.

The design and implementation of the prototype is presented and the results of a user test focusing on the prototype is discussed.

Kasper Lind Sørensen

Kasper Sørensen

klsa11@student.aau.dk

Simon B. Støvring

Simon B. Støvring

sstavr11@student.aau.dk

Preface

The report is the product of a master's thesis project on the 10th semester at Aalborg University, Denmark and describes the design and implementation of the problem presented in Chapter 1.

Special Thanks To

We want to thank our supervisor Brian Nielsen for his help and guidance throughout the project. We also want to thank Aalborg University for providing us with the necessary hardware to develop the prototype.

Summary

This thesis focuses on the design and implementation of a context-aware system for controlling a smart home system using gesture recognition performed on a smartwatch.

The system is designed to utilize contextual information about the user and the environment he resides in to determine which smart devices he wishes to interact with and what change of state he wish to apply to the smart devices. To limit the scope of the project the system only utilizes the following two sources of contextual information.

- The gesture the user has performed.
- The room the user is situated in.

The hardware requirements of the system is a smartwatch that supports Bluetooth Low Energy and contains an accelerometer, a computer to act as a hub for communication between the wearable and smart devices and one Estimote beacon per room in the smart home. The prototype was implemented using a second generation Moto 360 smartwatch and a Raspberry Pi 3 running the hub software.

The hub is intended to facilitate communication between the smartwatch and all the controllable devices and is responsible for triggering actions on these devices, such as turning on a lamp, when a user performs a gesture and an action has been determined.

In order to recognize gestures performed by the user, we developed an algorithm that combines the 1c [32] and the \$3 [40] gesture recognizer algorithms. The algorithm takes accelerometer measurements recorded on the Moto 360 smartwatch as input.

The second source of contextual information in this project is the position of the user which is obtained using Estimote Bluetooth Low Energy beacons. One or more beacons are placed in each room in the smart home. The beacons send out Bluetooth Low Energy signals that are picked up by the smartwatch and based on the RSSI value, the smartwatch determines which beacon is the closest and thus which room the user is in.

Users of the system can create configurations where combinations of a gesture, a room, a device and a supported action are selected. The selected action is triggered when the user performs the specified gesture in the selected room.

In order to utilize the gesture and room information to determine an appropriate action to trigger, we developed a context engine based on a Bayesian network. Given probabilities of the configured gestures and rooms, the Bayesian network computes beliefs of the configured actions. When the belief of an action is sufficiently strong, the action is triggered, thus changing the state of one or more smart devices in the home. If a single action cannot be determined, the user is presented with a list of probable actions.

To evaluate how well the system performed when used by users, a user test was carried out where seven people participated. The participants were asked to train four unique gestures ten times each, resulting in a total of 40 gesture templates. The gesture shapes were predetermined by us and the participants were instructed how to perform them but were given no opportunity to practice them. They were then asked to perform these gestures in different rooms simulated by turning the different beacons on and off.

Based on the user test we found that the correct action was triggered 44% of the time. A flaw was discovered in our Bayesian network after the test which led us to consider ways of improving it. The suggested improvements were tested using data from a single participant collected during the user test and the most promising suggestions were influence diagrams and introducing the state of the smart devices in the system as a contextual information to better eliminate false actions such as turning off a device that is already turned off.

Table of Contents

1	Introduction	1
1.1	Initial Problem	1
1.2	Scenario	3
1.3	Controlling a Smart Home	7
1.4	Problem Statement	8
1.5	Related Work	8
1.6	Requirements Specification	9
2	Analysis	11
2.1	Hardware Components	11
2.2	Context	15
2.3	Choice of Wearable	17
2.4	Choice of Gesture Recognizer	19
2.5	Choice of Hub	22
2.6	Indoor Positioning	24
2.7	Recommender System Methods	25
2.8	Bayesian Network	27
2.9	Context Engine	31
3	Design	33
3.1	Positioning Using BLE	33
3.2	Gesture Recognition	37
3.3	Bayesian Network	39
3.4	Context Engine	46
4	Implementation	47
4.1	Status	47
4.2	Programming Languages	51
4.3	Component Diagram	52
4.4	Bayesian Network	55
4.5	Context Engine	55
4.6	Communication with openHAB	58
4.7	Prototype	59
4.8	Version Control	62
5	Evaluation	65
5.1	User Test	65
5.2	Alternative Models	69
6	Conclusion	73
6.1	Project Results	73
6.2	Future Work	74
	Bibliography	77

A	Housing Types	83
B	Communication With openHAB	85
C	User Test Results	89

Introduction

In this report we wish to investigate the option of controlling a smart home using motion gestures recognized using a smartwatch. In this chapter we will define the problem investigated in the report. We also define the target group and present an example of a scenario in which the target group can use the system envisioned in this report.

1.1 Initial Problem

In a previous report we documented the work on a system that integrated wearables into a home automation environment in order to provide an interface for controlling devices that are connected to the Internet [30].

In the previous report [30, pp. 69-73] we found that motion gestures can be used to control a home automation environment. We found the system to have an accuracy of 4.29%, *i.e.* the correct action was performed 4.29% of the time. The poor accuracy was due to the use of fine grained position information used when determining which device in the smart home to control.

The position of the user was utilized when determining which device the user points at and thus which device should be controlled. The future work of the report [30, pp. 71-73] suggests using contextual information to determine which device should be controlled rather than the granular positioning. By doing this it is no longer possible for the user to point at a device in order to control it but we may be able to improve the accuracy of the solution and given the correct contextual information we can narrow the set of devices the user desires to control sufficiently to provide an attractive solution for controlling a smart home.

This report is based on the work done in our previous report.

In the previous report [30, pp. 1-4] we presented Figures 1.1 and 1.2 that show an increasing trend in wearables and smart homes. Common for wearables and smart homes are that they are both involved in the concept of Internet of Things. As both wearables and smart homes are predicted to be an increasing trend [58, 35], it is interesting to combine the two in order to make a system that provides an interface for controlling a smart home using a wearable device.

The definition of a *smart home system* is that the system uses a smartphone application or a web portal as a user interface [35]. Therefore the numbers presented in Figure 1.2 does not include homes controlled solely by switches, timers, sensors and remote controls.

In order for a web portal or a smartphone application to provide meaningful functionality in a smart home system, the software should provide some mechanism for controlling or monitoring

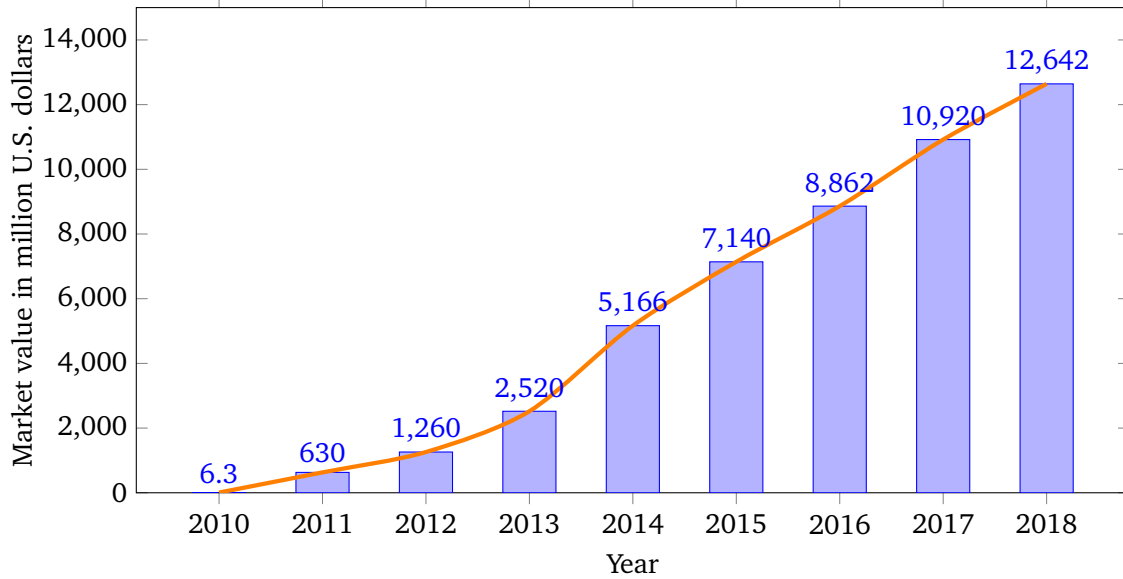


Figure 1.1: Wearables trend based on sales and statistics. Data from [58].

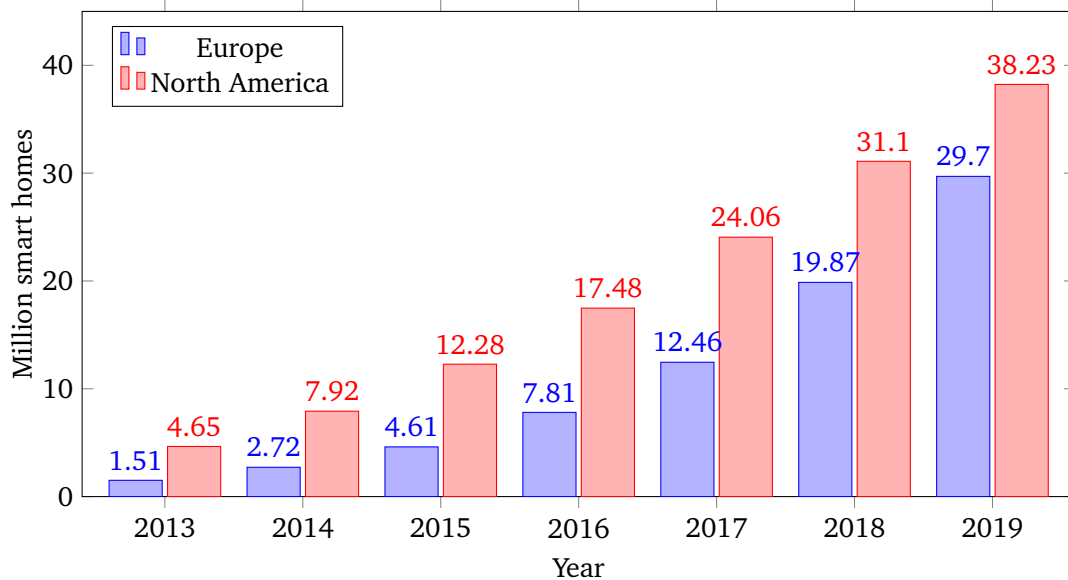


Figure 1.2: Smart homes trend based on sales and statistics. Data from [35].

devices in the home. In order to do this, the devices are accessible using some technology for exchanging data, e.g. WiFi or Bluetooth. These devices are involved in the concept of Internet of Things.

While not directly related to the concept of a smart home, a wearable device can play a role in a smart home. The wearable device can provide the application used for interacting with devices in the smart home.

Note that the definition of a smart home system as presented in [35] does not include to which degree the smartphone application or web portal is involved in the system. A simple system including a smartphone application with a single button for turning a light on and off is a *smart*

home system.

We accept the definition of a smart home presented in [35] and formulate it as shown in Definition 1.1 as it is sufficient for our use in that we are interested in replacing the smartphone application with an application running on wearable device.

Definition 1.1 *A smart home, is a home that can be controlled using a smartphone application or a web portal as a user interface.*

1.2 Scenario

This section describes a scenario in which the envisioned system may be used and outlines the target group of the system.

In [30, p. 15] we defined the target group as people who live in a smart home and are interested in controlling the state of their home, e.g. lights, music centres, doors and windows in a convenient way. The target group currently consist of early adopters of smart home technologies but based on the trend in Internet of Things, abbreviated IoT, our assumption is that the technology will be widespread within 5-10 years.

In this report we extend the definition of the target group presented in [30, p. 15] to include a definition of the surroundings. As shown in Table A.1 the most common size of housings in Denmark is 75-99 square meters. Therefore we assume that the solution presented in this report is installed in a typical Danish apartment with a size of 90 square meters and with 3-4 rooms.

We make the following assumptions about the system and the environment in which the scenario takes place:

- All controllable devices are connected through a central hub.
- The system is installed in a private home where the different rooms can be defined.
- The state of the controllable devices is always available.

We assume each room has two or three controllable lamps. The living room may have a television and a music centre while the kitchen may have a controllable coffee machine. We estimate that there are 2-5 controllable devices per room leaving us with a maximum of 20 controllable devices in an apartment.

We consider the system to be installed in the apartment shown in Figure 1.3. The rooms in the apartment are listed in Table 1.1 along with the controllable devices that seem realistic to be located in each room.

Table 1.1: Rooms in the apartment with possible devices in each room.

Region	Controllable devices
Hallway	Two lamps and a door lock.
Living room	Three lamps, a television, a music centre, a thermostat and a weather station.
Kitchen	Two lamps, a coffee maker and a music centre.
Bedroom	Two lamps and a television.
Home office	Two lamps and a music centre.
Bathroom	A lamp.

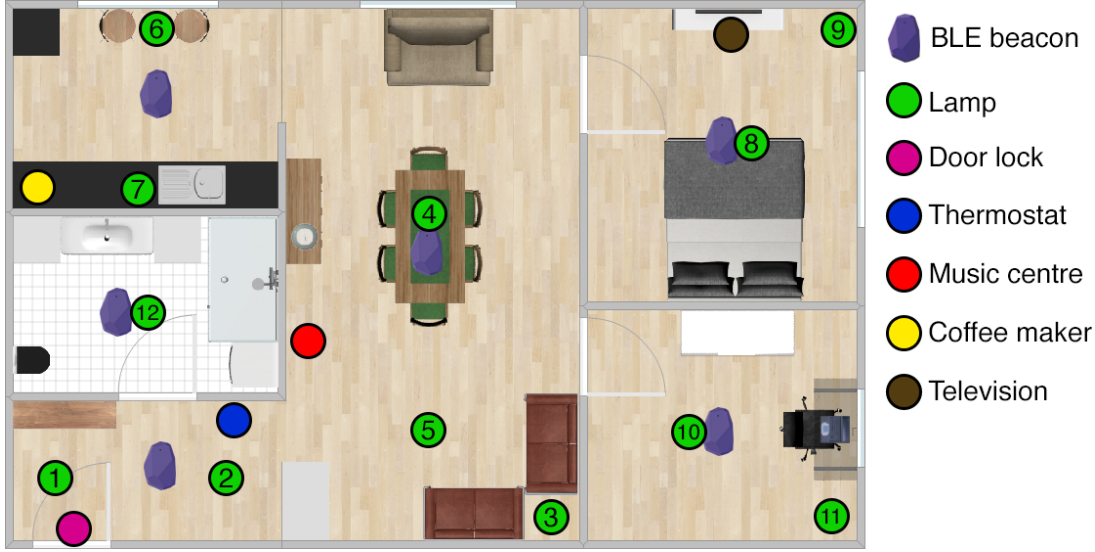


Figure 1.3: Example apartment we envision the system to be installed in.

For each room there are a number of gestures that are valid in that room and when performing a gesture, an action is sent to the hub and the action affects a device. Table 1.3 shows the devices and actions associated with the gestures in each room. For example, when the user performs a circle gesture in the hallway, he locks or unlocks the door lock.

We consider the following example of a day in a users life realistic to take place in the apartment illustrated in Figure 1.3 and table 1.1 with the gestures outlined in Table 1.2.

The following example briefly shows how the user can move around his apartment and use the same eight gestures to control different devices located in different rooms.

Example:

At the beginning of the day, the user wakes up in his bedroom. He makes a Z gesture to turn on Lamp 8 in the bedroom and gets out of bed. He goes to the bathroom to take a shower and turns on Lamp 12 by performing gesture Z as he enters. When he is done, he performs gesture Z to turn off the Lamp 12.

The user goes into the living room, and increases the temperature by performing gesture L, turns on Lamp 4 and Lamp 5 by performing gestures V and Triangle. Before he goes to the kitchen, he starts the music in the living room by performing the Circle gesture. In the kitchen he starts brewing coffee by performing the W gesture and gets some breakfast.









When the user is done eating breakfast, he goes to the living room and turns off the music by performing the Circle gesture. The user goes to the home office, starts the music in the office by performing the Circle gesture and turns on Lamp 10 and Lamp 11 by performing gestures Z and V.

When done working, he goes grocery shopping. As he leaves the home office, he performs gestures Z and V to turn off Lamp 10 and Lamp 11. He goes to the hallway and performs the Circle gesture to unlock the door. The user locks the door by performing a Circle gesture.

When the user comes home, he unlocks the door by performing a Circle gesture and unpacks his groceries in the kitchen and turns on Lamp 7 by performing a V gesture. When he is done unpacking the groceries, he turns off Lamp 7 again by performing the V gesture.

Earlier that day, the user had turned on Lamp 4 and Lamp 5 in the living room. He goes to the living room and turns off Lamp 4 and Lamp 5 by performing the V and Triangle gestures.

Table 1.2: Gestures a user may perform in the apartment shown in Figure 1.3

Gesture name	Illustration
Z	
V	
Circle	
Triangle	
L	
W	
Horizontal Line	
Half Circle	

He continues to the bedroom to watch television. The television is turned on by performing the Circle gesture. After watching television, the user turns off the TV by performing the Circle gesture and Lamp 8 by performing the Z gesture.

1.2.1 Handling Uncertainties

Not all scenarios will work out as well as the one previously presented. Sometimes a meaningful action cannot be determined from the context of the user. Assume that a person is in his living room and he performs a gesture that is recognized as “Turn on TV”, but the only TV in the house is already turned on. In this case the action could be considered void, but we propose a different solution. Instead of ignoring the persons request, a list of alternate actions could be presented to him on his smartwatch. Which actions would be presented however, could be the inverse of the one performed (if applicable) which in this case would be to turn the TV off. It could also be a list of the actions most frequently performed by the user.

If the gesture performed is not recognized, this could be considered void and the person would be asked to try again. Alternatively, the intended target could be assumed to be the device that the person most recently interacted with, and a list of the most frequently used actions could be presented on the persons smartwatch.

1.2.2 Controlling Devices in Other Rooms

Assume that a person is in his home office listening to music but then remembers that he forgot to turn off the music in the living room. He wishes to turn off the music in the living room but not

Table 1.3: Gestures that perform actions on controllable devices in each room of the apartment shown in Figure 1.3.

Region	Affected device	Gesture
Hallway	Lamp 1	Z turns on and off.
Hallway	Lamp 2	V turns on and off.
Hallway	Door lock	Circle locks and unlocks.
Living room	Lamp 3	Z turns on and off.
Living room	Lamp 4	V turns on and off.
Living room	Lamp 5	Triangle turns on and off.
Living room	Thermostat	L increases the temperature.
Living room	Thermostat	W decreases the temperature.
Living room	Music centre	Horizontal Line skips to next track.
Living room	Music centre	Half Circle skips to previous track.
Living room	Music centre	Circle plays and pauses.
Kitchen	Lamp 6	Z turns on and off.
Kitchen	Lamp 7	V turns on and off.
Kitchen	Coffee maker	W starts brewing.
Kitchen	Music centre	Horizontal Line skips to next track.
Kitchen	Music centre	Half Circle skips to previous track.
Kitchen	Music centre	Circle plays and pauses.
Bedroom	Lamp 8	Z turns on and off.
Bedroom	Lamp 9	V turns on and off.
Bedroom	Television	Circle turns on and off the television.
Bedroom	Television	L increases the volume.
Bedroom	Television	W decreases the volume.
Bedroom	Television	Horizontal Line changes to next channel.
Bedroom	Television	Half Circle changes to previous channel.
Home office	Lamp 10	Z turns on and off.
Home office	Lamp 11	V turns on and off.
Home office	Music centre	Horizontal Line skips to next track.
Home office	Music centre	Half circle skips to previous track.
Home office	Music centre	Circle plays and pauses.
Bathroom	Lamp 12	Z turns on and off.

on the speakers located in the home office. If the user performs the Circle gesture to pause the music, he would pause it in the home office. To circumvent this, the smartwatch application could allow the user to select which room he would like to control devices in.

1.3 Controlling a Smart Home

Controlling a smart home is a matter of changing the state of smart devices in the home. For example, opening or closing a window, locking or unlocking the door or changing the temperature on the thermostat. There are a variety of ways to control a smart home including, but not limited to, the following.

- Using a smartphone application.
- Using a physical remote control such as the Logitech Harmony Remote¹.
- Using rules that are automatically triggered when specific events occur, *e.g.* the time of the day changes.

An alternative way of controlling a smart home is using motion gestures using a wearable worn by the user. A survey of 37 people found that 76% consider gestures a natural way of controlling devices, 8% found it unnatural and the remaining 16% left the question unanswered [37].

1.3.1 Required Amount of Gestures

The amount of gestures a person is able to recall is limited and the smaller a set of gestures is, the easier it is for a person to remember. This is supported by [37] wherein a user study reported that users would like to be able to use the same gesture for multiple devices and use a small set of gestures. The ideal size of a gesture set is unknown, in part because people have differing recollection capabilities. Miller [43] theorized that an adult can store 7 ± 2 objects in their working memory. Reuse of gestures across multiple devices results in fewer gestures to remember, but imposes the challenge of making sure that only the user's intended target device reacts to a gesture.

The scenario presented in Section 1.2 has a total of 30 actions that can be triggered. If we assume there is only one music centre in the apartment that can be controlled from multiple rooms, we can reduce the amount of actions to 24.

If we make a system in which one gesture maps to a single action, a minimum of 24 gestures would be required to control the smart home shown in Figure 1.3. Previous studies suggests that a user cannot remember 24 gestures and more so, cannot remember which action each of the gestures trigger [37, 43].

By taking the location of the user into account, we can reduce the number of gestures a user must remember in order to control the smart home.

From the scenario presented in Section 1.2 it is apparent, that in a gesture controlled system in which the performance of a gesture triggers different actions depending on the room a user is in, the maximum amount of gestures a user must remember equals the number of actions that can be triggered in the room with the most actions.

In the scenario in Section 1.2, the living room has a total of eight different actions, making it the room with most actions. Therefore a total of eight different gestures are required in the system.

¹More information about the Logitech Harmony Remote is available at <http://www.logitech.com/harmony-remotes>

1.4 Problem Statement

It is our hypothesis that we can utilize contextual information to determine which device a user intends to control in a smart home environment. Our problem statement is as follows.

How can we design and implement a system that utilizes contextual information for controlling a smart home using a wearable in a gesture driven solution?

1.5 Related Work

Various ways of interacting with the systems in a smart home are presented in [11, pp. 9-10] including speech, facial expressions and gestures. Gestures have been found to be an easy way of interacting with systems [51, p. 6] and is utilized in the solution presented in this report. In [57, pp. 2-3] motion gestures are described to be more convenient than regular remotes, since they do not require the user to be carrying a remote with them or in the case of wall mounted panels, walk up to the remote. Furthermore [57] describes how speech commands may drown in the noise if users are controlling a media center and that interaction using speech is inconvenient in multi-user configurations.

In [30, pp. 9-11] we described Reemo as related work. Reemo is a solution in which users point at devices and control them using pre-programmed motion gestures [52]. While the company has not released any details about their technology, we can see from their website that the solution is limited to devices within line of sight as a receiver must be placed next to each controllable device.

Figure 1.4 illustrates how a user of Reemo points at a smart device and then performs a gesture to send a command, *i.e.* trigger an action.

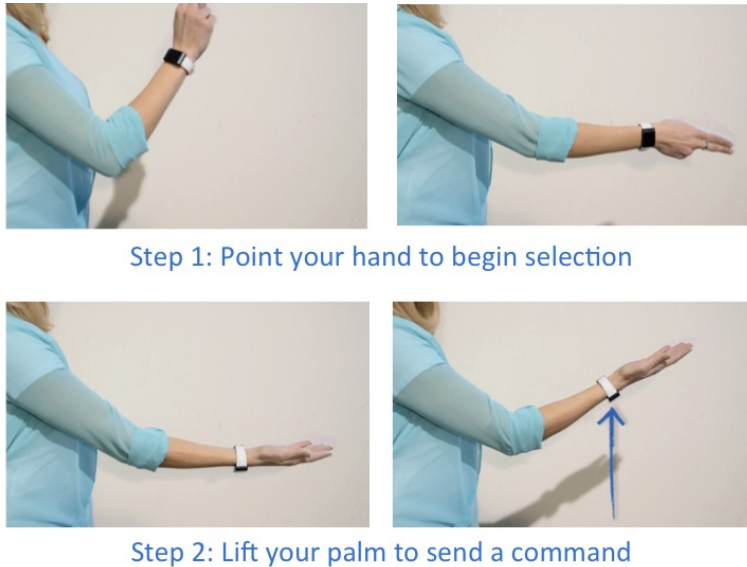


Figure 1.4: Using the Reemo wearable to control a smart device. Image from [30].

In [9] a solution for recognizing context-aware motion gestures using multiple Kinects is presented. Users control devices by pointing at them and depending on the current state of the device and the position of the user, different actions are triggered. The authors use two Kinects to

position the user and recognize motion gestures in a living room, requiring a total of 6-8 Kinects in an apartment consisting of 3-4 rooms. The high number of Kinects results in a high price for installing the system. Furthermore there may be privacy concerns when using Kinects, as the cameras can be utilized for malicious activity.

In contrast to the previous solutions, the solution presented in this report differs in the way, that controllable devices are not required to be within line of sight. Users are able to control all devices within the system from anywhere in their house.

The solution will utilize beacons for positioning users as opposed to Kinects as utilized in [9]. This lowers the entry barrier by decreasing the initial cost as well as the cost for scaling the system to more rooms.

By utilizing an accelerometer, a common component in wearables [30, pp. 3-4], the user may not need to buy hardware specifically for recognizing gestures.

1.6 Requirements Specification

This section presents the requirements for the solution. The requirements are divided into the following three groupings.

Functional requirements These represent the functionality the solution should implement.

Performance requirements These requirements describe how well the solution should perform in given conditions.

Overall requirements These are requirements that the system as a whole should fulfill and that does not fit within any of the two other groupings.

1.6.1 Functional Requirements

Train and recognize gestures Users should be able to train a motion gesture and the system should be able to recognize the gesture when the user performs this gesture.

Trigger actions on controllable devices using gestures Users should be able to trigger an action on a controllable device by performing a gesture using a wearable.

Context-aware The system should be context-aware such that different actions may be triggered from the same gesture depending on contextual information. For example, a circular gesture may turn on the TV when the user is in the living room but turn on the lights when the user is in the kitchen.

Associate a gesture with actions The user should be able to associate a gesture with one or more actions that a controllable device can perform. A gesture can only be associated with multiple actions, if the controllable devices to which the action belongs reside in different rooms. Actions change the state of one or more smart devices in a smart home when triggered.

Virtual positioning of users A user should be able to virtually position himself in his home using the wearable. This allows the user to perform gestures in one room, while being in another.

1.6.2 Performance Requirements

Limit the amount of gestures by letting devices share gestures As described in Section 1.3, users should be able to use the same gestures for multiple devices and thus reduce the overall amount of gestures they need to recall.

Handle 15-20 controllable devices As described in Section 1.2 we assume the system is deployed in an apartment with 3-4 rooms with 2-5 devices in each room. Therefore the system should be able to handle a minimum of 15-20 controllable devices.

Trigger Correct Action The correct action should be triggered at least 80% of the time. An action is considered correct if it is the one that the user intended.

1.6.3 Overall Requirements

Use inexpensive hardware and software As described in Section 1.2 the target group are people living in a smart home, typically in an apartment with a size of 90 square meters. Therefore we are not interested in expensive solutions when looking at hardware or software.

Not limited to smart devices in line of sight Reemo [52] limits the user to control devices that are in line of sight. We do not want to have this limitation in our solution, as it limits the smart devices that can be controlled.

Analysis

2.1 Hardware Components

The following section summarizes the hardware components needed in order to design and implement the scenarios presented in Section 1.2.

We envision two different configurations for locating the user in the system. Both configurations assume that Bluetooth technology is utilized to determine the position of the user as described in Section 2.6.

1. One configuration in which the wearable continuously scans for Bluetooth beacons. The wearable determines the position of the user based on data advertised by the beacons.
2. Another configuration in which Bluetooth enabled microcontrollers scan for wearables and upload the Received Signal Strength Indication (RSSI) to a central location in which the position of the user is determined based on the set of available RSSIs.

The idea of the second configuration is to design the system in such a way that it works on wearables that do not provide access to the Bluetooth APIs. A microcontroller should do one of the following.

- It should either continuously scan for wearables and read their RSSI.
- Given the MAC address of the wearable, it should obtain the RSSI.

The second configuration was abandoned due to restrictions posed by the Bluetooth Low Energy (BLE) specification. The specification poses the following limitations that prevent such a system from working properly.

- A BLE peripheral, *e.g.* a wearable, can only be paired with a single other device.
- A peripheral must advertise in order to be discovered by a central. Advertising requires a piece of software to be running on the peripheral and thus access to the Bluetooth API.
- A central can obtain an RSSI based on the MAC address of a peripheral. However, the Bluetooth specification allows a peripheral to change its MAC address in order to prevent tracking of the user [5, p. 91]. The change of MAC address could be disabled for this approach, if this is possible on the wearable. However, since this is a security feature introduced in the Bluetooth specification, this seems undesirable for our purposes.

The rest of the report will focus on the first configuration of the system.

2.1.1 Required Hardware

The following list presents the hardware needed for the system.

- A computer running the hub. The hub is responsible for forwarding commands from the user to the controllable devices, *e.g.* lamps. The benefit of running the hub on a central computer rather than the smartwatch, is that the hub could potentially be configured with rules for automation and should therefore always be running, compromising the battery life of a wearable. Furthermore placing the logic in a central place can prove beneficial in an environment with multiple users in which multiple hubs would have to be synchronized.
- A wearable which provides access to APIs for both Bluetooth and the accelerometer. Furthermore it should be possible to give some sort of feedback to the user when a gesture could not be recognized. The wearable should also be able to communicate with hub.
- Minimum one Bluetooth beacon per room. Two beacons are needed in order to test the system in more than one room. The Bluetooth beacons are used to determine which room the user is in.
- Minimum two controllable devices, one per room in the system. These devices receive requests from the hub that ask them to change their state.

The above lists the bare minimum of hardware required in order to implement the system. Sections 2.3, 2.5 and 2.6 elaborates on the choice of hardware components.

2.1.2 Accelerometer

One of the required components the wearable should contain according to Section 2.1.1 is an accelerometer.

Accelerometers are used for measuring the acceleration of an object. In our case we use it to measure the acceleration of the wearable installed on a users wrist and as a result of this, the users arm movements. When measuring the users movements, we can recognize the gestures he performs.

When an object is subjected to a force, including gravity, it accelerates. Acceleration can be expressed as change in velocity over time as in Equation (2.1) where \vec{a} is the acceleration, Δv is the change in velocity and Δt is the duration. Velocity is measured as meters per second and time is measured in seconds, hence the acceleration is measured as meters per second per second or $\frac{m}{s^2}$. Acceleration can also be expressed in terms of force applied to the object as in Equation (2.2) where a is the acceleration of the object, F is the forces applied to the object expressed as a vector with a force for each axis and m is the mass expressed as a scalar value. The forces are measured in Newton and mass is measured in kg.

$$\vec{a} = \frac{\Delta v}{\Delta t} \quad (2.1)$$

$$a = \frac{F}{m} \quad (2.2)$$

The accelerometer determines the force applied to the object [17, pp. 392-393] so in order to calculate the acceleration, the mass of the object must be known.

Accelerometers of varying design exist [17, pp. 392-411]. An example of these is the capacitive type semiconductor accelerometer which is illustrated in Figure 2.1. The accelerometer has an electrode in the middle (14) which is supported by a beam (13). The beam is flexible enough to move slightly up and down when the accelerometer is moved, thus moving the electrode up and down. The beam is mounted to the side of the accelerometer (2). The gap between the electrode (14) and the two stationary electrodes (25, 26) constitute two electric capacitors having capacitances of C_1 and C_2 . When the movable electrode in the middle (14) moves up and down, the capacitances C_1 and C_2 changes slightly [39].

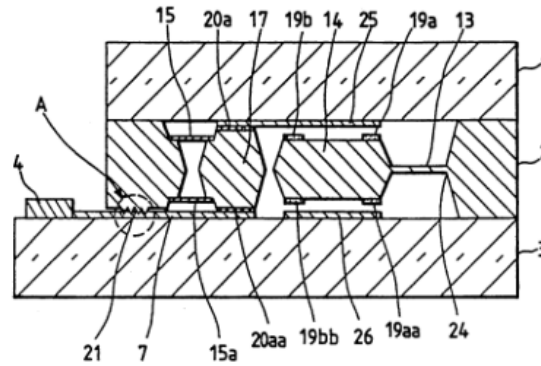


Figure 2.1: Capacitive type semiconductor accelerometer. Illustration from [39].

The changes are registered and constitutes the acceleration. This allows for measuring the acceleration on one axis. The same principle can be used to measure the acceleration on several axes.

In this project we utilize the accelerometer for recognizing gestures but other common applications of the accelerometer include pedometers, game controllers and fall detection.

2.1.3 Bluetooth

We use Bluetooth Low Energy (BLE) in order to position the user wearing the smartwatch. Bluetooth is a standard for short-range and low-power wireless communication between devices and is commonly found in a broad range of devices including desktop computers, phones and speakers [29].

Bluetooth has a maximum range of 100 meters but is typically used for much shorter distances [29, p. 20]. Accomplishing the maximum range is difficult as walls, furniture and people can dampen the strength of the signals [15].

Figure 2.2 shows the high-level layered architecture of Bluetooth. The layers are described below.

Lower layers Perform low level operations, including discovering devices, establishing connections and exchanging data packets. The functionality is implemented on the Bluetooth chip [29, pp. 21-22]. We will not go into details about this layer.

Upper layers Use functionality of the lower layers to perform complex functionality, including transferring large amounts of data by splitting it into multiple packets and streaming of data [29, p. 22].

Profiles The profiles define how the protocol layers within the upper and lower layers implement specific use cases, *e.g.* proximity detection [29, p. 22].

Applications These are applications utilizing the Bluetooth stack, *e.g.* mechanisms for discovering and connecting to Bluetooth devices, choosing music to stream and selecting files to transfer [29, p. 22].

As part of the upper layers, is the Logical Link Control and Adaption Protocol, abbreviated L2CAP. The protocol builds on top of protocols in the lower levels, to exchange data with a remote Bluetooth device. L2CAP provides functionality that includes segmentation and reassembly of packets, quality of service, streaming data and retransmission of packets. Devices communicating using L2CAP exchange Packet Data Unit (PDU) packets, containing information about the L2CAP protocol, *e.g.* the type of the PDU and a payload [29, pp. 80-83]. For example, the PDU type of a

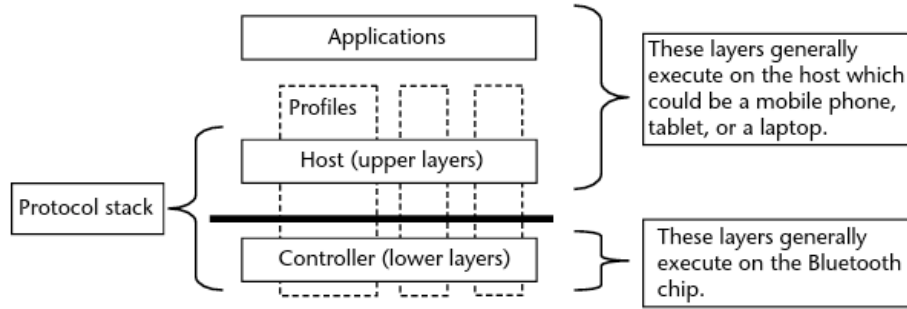


Figure 2.2: High-level architecture of Bluetooth. Illustration from [29, p. 22].

beacon can be `ADV_NONCONN_IND`, indicating a non-connectable undirected advertising packet. The payload of a PDU is referred to as the Service Data Unit (SDU) which originates from a level above the L2CAP, e.g. the Attribute Protocol [29, p. 201].

Attribute Protocol, abbreviated ATT, uses L2CAP to transfer data. Mechanisms provided by ATT include discovering the attributes provided by a remote device and reading and writing the attributes. An attribute represents data, for example the temperature from a thermostat, the unit in which the temperature is provided or the name of a device. Attributes can be pushed or pulled to and from a remote device. Attributes have a handle that identifies an attribute, a value and access permissions. The protocol works in a client-server manner in which a server exposes a set of attributes and a client can read and write the attributes [29].

The Bluetooth Core 4.0 Specification [6], i.e. the specification including BLE, introduces the General Attribute Profile (GATT) architecture illustrated in Figure 2.3. The GATT framework specifies how a device can discover, read, write and indicate its characteristics. Profiles consist of one or more services that are needed in order to provide a specific functionality, e.g. proximity monitoring [29, p. 259-261]. Services provide one or more characteristics that describe a feature, e.g. the temperature of a thermostat. Services may be shared by multiple profiles.

The Bluetooth Special Interest Group, the group that maintains the Bluetooth standards, provide a range of profiles, e.g. a proximity profile for monitoring the distance to devices and a profile for heart rate sensors. Apple and Google have developed the iBeacon and Eddystone profiles, respectively, for region monitoring. Eddystone GATT profile defines the Eddystone Service that advertises *frames* of information. These are described in detail in Section 3.1 as well as the Eddystone Configuration Services in which a device is connectable and the advertised values can be configured by a client BLE-enabled device.

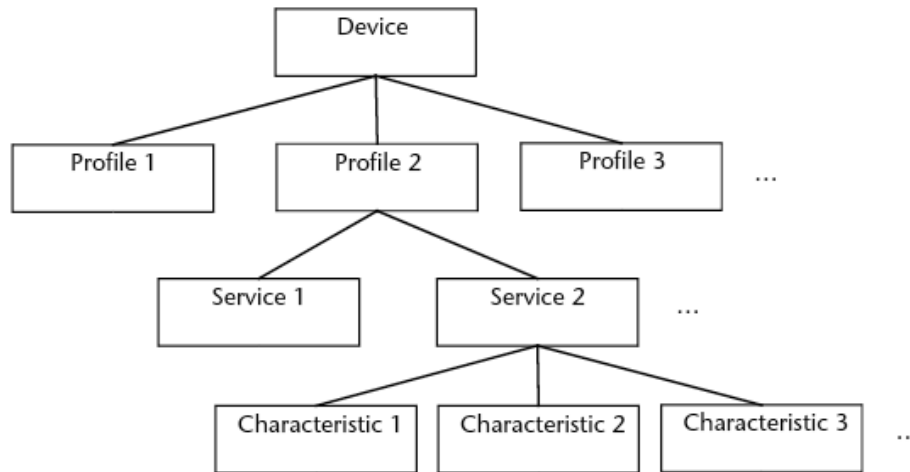


Figure 2.3: Relationship between profiles, services and characteristics. Illustration from [29, p. 261].

2.2 Context

In the envisioned system the context is used to determine which actions should be triggered when the user performs a gesture. Therefore the context plays an important part in the envisioned system.

The notion of context is researched in multiple fields, including philosophy and psychology [7]. For the purpose of this project we focus on the notion of context in context-aware software systems. In [2], the context for a context-aware system is defined as in Definition 2.1.

Definition 2.1 “We have defined context to be any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or object. In context-aware computing, these entities are anything relevant to the interaction between the user and application, including the user and the application.” [2]

According to this definition, information is context if the information characterizes the situation of a participant in an interaction [2].

Consider the following example.

Example:

A clothing store has a system installed that sends notifications to users mobile devices with offers when they are near the clothes that the offer apply to. If a t-shirt is 20% off, and the user is near the t-shirt, the user will receive a notification letting them know that the t-shirt is on sale.

In the above example, context includes:

- The position of the user.
- The position of the t-shirt on sale.
- The sex of the user.
- The age of the user.

- The percentage the price of the t-shirt is reduced with.

Information such as what other customers are in the store and the time of the day is not context because it is not relevant to the interaction between the user and the application.

2.2.1 Context Types

In [2] the context is divided into two categories, *primary* and *secondary*. The “Location, identity, time, and activity” [2, p. 5] are categorized as the *primary* context types that “answer the questions of who, what, when, and where” [2, p. 5]. Answering these questions help us understand *why* a given situation occurs. In this project an action is triggered on a controllable device because the user is interested in triggering it (that is the *why*).

Secondary context types can be derived from the *primary* context types. When the identity of a user is known, additional information can be derived. In the previous example the sex and age of the user are primary context types that can be derived from the identity of the user. The price reduction of the t-shirt is secondary information as well as it can be derived from the identity of the t-shirt. The position of the user and the t-shirt are both primary context types.

2.2.2 Context Features

Abowd *et al.* [2] defines a context-aware system as in Definition 2.2.

Definition 2.2 “A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.” [2, p. 6]

Based on this definition, [16] outlines the following three main features a context-aware system can provide to its users.

“Presentation of information and services” [16, p. 13] Systems with this feature use context to suggest services to the users or present them with relevant information. Yelp, a service that presents users with nearby businesses, is an example of a system implementing this feature.

“Automatic execution of a service” [16, p. 13] Systems with this feature automatically execute a service based on context. Philips Hue, which can automatically change the lighting based on the time of day, is an example of a system implementing this feature.

“Tagging of context to information for later retrieval” [16, p. 13] Systems with this feature associate information with context. [16] uses a service that tags locations with a virtual note for other users to see as an example of systems implementing this feature.

According to the categorization of context-aware systems by Ferreira *et al.* the system envisioned in this report belongs to the category of systems implementing automatic execution of a service. Based on context, the system automatically triggers an action on a controllable device. While the user must perform a gesture in order to trigger the action, the system is still automatic as we consider the gesture to be context.

2.2.3 Conclusion

We follow the definitions of context and context-aware systems proposed by Abowd *et al.* We also introduce the term *contextual information* as defined in Definition 2.3.

Definition 2.3 *Contextual information is information that is part of the context, e.g. the position of the user.*

Examples of contextual information, include the following.

- The day of the week.
- The time of the day.
- The position of the user.
- The mood of the user.
- The sex of the user.
- The age of the user.
- Users present, e.g. users that are home.
- Sentences or words pronounced by the user.
- Motion gestures made by the user.
- The state of the controllable devices in the system.

While other context can be included in the system, for the purpose of the prototype presented in this report, we chose to focus on the following two contextual informations.

- The position of the user.
- The gesture performed by the user.

The contextual information provides us with a way of initiating the process of determining the correct action to trigger. When the user has performed a motion, we begin the recognition of the context. Gestures can trigger different actions depending on the position of the user, thus reducing the number of gestures the user needs to remember as a single gesture can be configured to trigger different actions as illustrated in Section 1.2.

We refer to the combination of gesture, room and action as a *gesture configuration* as defined in Definition 2.4. Gesture configurations are meant to be configured by a user. For example, a user can configure a circle gesture to turn on the table lamp when he is in the living room.

Definition 2.4 *A gesture configuration is the combination of a gesture, a room and an action. When the user performs the gesture in the specified room, the selected action is triggered in the system.*

The system is context-aware as it automatically executes a service when a gesture is performed, provided that an appropriate action to trigger can be determined.

2.3 Choice of Wearable

The wearable is primarily used for gesture recognition and positioning of the user. Furthermore the wearable is utilised to give feedback to the user when a gesture could not be recognized and for creation of gesture configurations. The wearable should allow for interactions, such as virtually changing the position of the user as described in Section 1.2.

Hence we decided on the following requirements.

- The wearable must possess a screen on which visual feedback can be presented and allow for presenting and interacting with proposed actions, in case we are unable to determine one specific action to be triggered. Furthermore, displaying and interacting with information is needed when the user virtually changes his position.
- The wearable must possess an accelerometer in order to detect the users movements and derive gestures from those.
- As described in Section 2.6, Bluetooth is used for positioning the user. Therefore the wearable should possess Bluetooth in order to position the user.
- WiFi connection in order to send commands to the hub. The wearable should ideally be independent of the phone and should send commands directly to the hub. WiFi is suitable for this as it is assumed that a WiFi connection is available in the entire house of the user. Had we used Bluetooth for communication between the wearable and the hub, it is likely there would often be connection issues as Bluetooth signals are significantly weakened by walls and furniture.
- We must be able to install and run our own applications on the wearable.

As we want a wearable that allows for both performing gestures, providing visual feedback to the user and let the user interact with an application running on the wearable, we chose to focus on smartwatches.

Performing gestures using a hand feels more natural than the feet, head or other body parts. This makes smartwatches ideal for recognizing gestures as they are worn on the wrist. Furthermore smartwatches are typically equipped with a touchscreen or buttons to interact with a screen or both.

The following wearables were chosen based on their popularity, availability and a desire to include a watch by each of the major players in the mobile market which, as of writing is Google, Microsoft, Apple and Samsung. Furthermore we included a Pebble Classic in our analysis as we already had access to one.

- Pebble Classic running Pebble OS.
- Second generation Moto 360 running Android Wear.
- Samsung Gear S2 running Tizen.
- Apple Watch running watchOS.
- Microsoft Band running Microsofts wearable architecture.

Table 2.1 shows a comparison of the smartwatches based on the parameters previously listed. The table shows that the Pebble Classic and the Apple Watch do not provide access to the Bluetooth API and thus we cannot perform positioning directly on the smartwatch. An alternative is to accept this limitation and perform the positioning of the user on a smartphone and continuously transfer the positions to the hub. The smartwatch can then retrieve the positions when needed. This limitation requires the user to carry the phone wherever he goes.

2.3.1 Conclusion

Table 2.1 shows that only two wearables fulfill our requirements, namely the second generation Moto 360 by Motorola and the Gear S2 by Samsung. The other wearables provide access to the an accelerometer but do not provide access to the Bluetooth API.

Of the two proposed wearables, we find the Moto 360 to be best suited for our project as we already have experience with the Android platform and therefore should be able to make progress on a prototype of the system faster.

⁰We found no reference to this in the official documentation and therefore assume that the feature is not available.

Table 2.1: Comparison of wearables

	Pebble Classic	Second generation Moto 360
Feedback	Screen with hardware buttons for interaction.	Touchscreen.
Gesture recognition	Accelerometer. [50]	Accelerometer and gyroscope available from API level 3 and 9 respectively. [44, 22, 23, 24]
Positioning	No access to the Bluetooth API.	Access to the Bluetooth API available from API level 1. [44, 22, 20, 21]
WiFi	Unknown ⁰ .	Yes.
	Samsung Gear S2	Apple Watch
Feedback	Touchscreen with a rotating bezel.	Touchscreen with digital crown.
Gesture recognition	Accelerometer and gyroscope. [53, 62]	Accelerometer.
Positioning	Access to the Bluetooth API. [53, 61]	No access to the Bluetooth API.
WiFi	Yes.	Yes.
	Microsoft Band	
Feedback	Touchscreen.	
Gesture recognition	Accelerometer and gyroscope. [41, 42]	
Positioning	Unknown ⁰ .	
WiFi	No.	

2.4 Choice of Gesture Recognizer

As in our previous report [30, p. 19] we divide gesture recognition into the following two categories based on [37]:

- Camera based.
- Motion based.

Examples of camera based approaches for gesture recognition include the use of one or more Microsoft Kinects that record the users movements [9]. The benefit of Microsoft Kinects is, that they can potentially record gestures performed with any part of the body. Another approach is an infrared gesture pendant worn by the user which records their hand motions when gestures are performed in front of their chest [57].

As stated in Section 1.5 we opt to not use camera based methods as they require a line of sight between the user and any device he intends to interact with or optionally, a line of sight between the user and multiple cameras installed in his smart home.

Hence we elected to use a motion based approach. As of spring 2016 a significant amount (55%) of wrist-worn wearables contained an accelerometer [30, pp. 2-3] and as such it makes sense to focus on motion based gesture recognition utilizing an accelerometer.

While it is natural to use hands to perform gestures, we chose to look at wearables that are worn on the wrist rather than wearables that are worn on the hands or the fingers. As shown in

Figure 2.4 only very few wearables are worn on the hands and we found it more interesting to develop for more widely available devices. In regards to gestures, the primary difference between recognizing a gesture with a wearable worn on the hand and on the wrist, is that the motions must be larger, when the wearable is worn on the wrist, *i.e.* the user must move the entire arm in order to produce significant accelerations on the axes of the accelerometer.

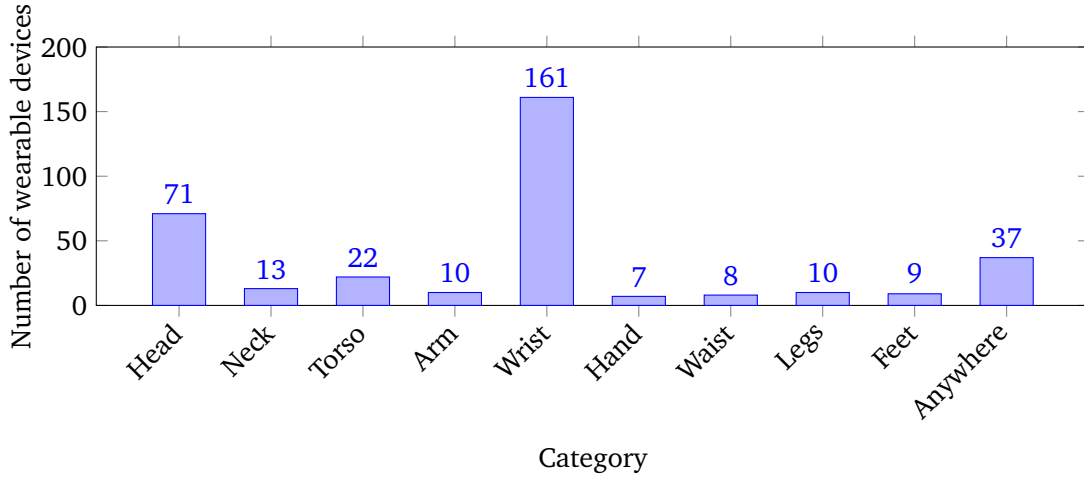


Figure 2.4: Placements of wearables. Graph from [30, p. 2], data from [34].

In [30] we used the \$3 recognizer [40] which is based on the \$1 recognizer [65]. Both are designed to be simple and easy to implement and the main difference between them is that \$1 is designed for two-dimensional gestures drawn on a screen whereas \$3 is designed for three-dimensional gestures captured using a tri-axis accelerometer.

While the \$3 gesture recognizer works adequately [30, p. 55], we decided to search for an alternative that would possibly work better on embedded and resource constrained devices.

We have the following requirements for the gesture recognizer.

- It must utilize an accelerometer to detect gesture motion data.
- It must run on Android Wear.
- It must recognize a gesture faster than 200 milliseconds.
- It must support user-defined gestures.
- Preferably it should run on the wearable independently from the smartphone.

Based on these criteria we will examine the \$3 [40] and 1¢ [32] gesture recognizers.

2.4.1 \$3

The \$3 gesture recognizer is based on the \$1 gesture recognizer but works with three-dimensional accelerometer measurements instead of two-dimensional coordinates [40].

The \$3 gesture recognizer translates measurements on a tri-axis accelerometer to coordinates by using the first measurement as origo and adding the difference between subsequent measurements to form a timeseries of acceleration deltas as shown in Figure 2.5.

This timeseries T is known as the gesture trace and will be rotated to find the smallest distance to the trained gesture templates.

The gesture recognizer was used in [30] and was found to have a precision between 58% - 98%, depending on the user utilizing the recognizer [40, p. 344]. The concrete implementation of the

$$T = \{p_1, \dots, p_X\} \text{ s.t. } p_i = p_{i-1} + a_i - a_{i-1}$$

Figure 2.5: Gesture trace representation in \$3. T is the gesture trace, a_i is an accelerometer measurement and X is the number of accelerations measured. This equation is not available in [40] but is created by us to illustrate how they create gesture traces.

recognizer was also found to have memory issues where allocated memory was not deallocated [30, p. 54].

2.4.2 1¢

The 1¢ gesture recognizer is designed as an improvement of the 1\$ recognizer [65]. Improvements are made by removing computationally expensive rotate-and-check methods and instead using a one-dimensional representation of gestures that is rotation-invariant. This means that rather than attempting to rotate a given gesture trace to best match the stored templates, an alternative representation is used such that rotating the trace is not required regardless of how it was performed.

A gesture is recorded as a timeseries of two-dimensional coordinates which is then resampled to contain a fixed amount of equidistant points. The 1¢ recognizer uses a resample rate of 64 to match that of \$1 but state that any value between 16 and 128 will suffice [32, p. 41]. \$3 resample the gesture traces with $N = 150$ and state that a higher resample rate N provides higher precision at the cost of increased computation time [40, p. 342]. Once a gesture has been resampled, the centroid is calculated and for each point in the gesture the euclidean distance between that point and the centroid is calculated and stored in an array as shown in Figure 2.6.

$$d = \{d_i, \dots, d_N\} \text{ s.t. } d_i = ||p_i - c||$$

Figure 2.6: Gesture trace representation in 1¢. d is the gesture trace represented as distances to the centroid, p_i is a point in the trace and c is the centroid of the trace. Source: [32]

This array of distances to the centroid is the one-dimensional representation of a gesture, called a *template* in [32]. The idea behind this one-dimensional representation is that it does not matter whether the user performs a rotated version of the gesture because the distance from each point in the trace to the centroid is the same regardless. This is illustrated in Figure 2.7 which shows the same gesture with two different rotations. The distances $D1$ and $D2$ from two points to the centroid C will remain the same in both cases.

When attempting to recognize an input from a user, the input is resampled and converted to a template in the same way to enable comparison between the input and the stored templates. Comparison between the input and a template is done by calculating the distance between the input and each stored template using Figure 2.8. The template with the lowest L^2 distance to the input is the best match.

2.4.3 Conclusion

Concrete implementations of \$3 and 1¢ are available and both support the Android Wear platform, although only \$3 utilizes the accelerometer data in order to train and recognize gestures. While we have previous experience with the \$3 algorithm and it is sufficiently accurate, we chose to

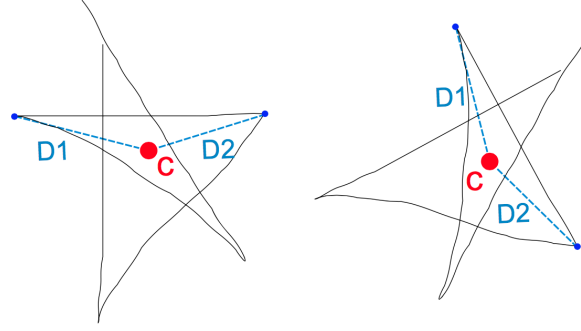


Figure 2.7: Star Example. Source: [32]

$$L^2(v_1, v_2) = ||v_1 - v_2||^2$$

Figure 2.8: Distance between two templates. Source: [32]

utilize the 1¢ as it requires fewer computations by not rotating the trace and is thus better suited for embedded devices.

The 1¢ gesture recognizer only supports recognition in two dimensions, *e.g.* recognition on a touch screen, therefore we chose to implement support for a third dimension and use it with accelerations on the X, Y and Z axes provided by the accelerometer of an Android Wear device by merging it with the technique used in \$3.

2.5 Choice of Hub

As mentioned in Section 1.2, we intend to use a central hub for connecting devices in the smart home setting. In [30] we presented a list of different home automation hubs and decided to use HomePort.

For this project we decided to revisit the following hubs to determine which one we deem to be the right one to use now.

- HomePort
- SmartThings
- OpenHAB

Common for all three solution is that they serve as adapters. Devices that fit within the concept of IoT, *e.g.* smart bulbs, media centres, windows, door locks *etc.* communicate using different protocols. These protocols include ZigBee, Z-Wave, HTTP and Bluetooth. The hubs expose devices communicating using those protocols under a common interface *e.g.* HTTP.

The hub should fulfill the following requirements.

- We intend to keep the price of the entire solution presented in this report at a minimum, therefore we are interested in an inexpensive hub.
- As the primary focus of the report is not on the development and research of a hub, we intend to choose a solution which can easily be used within our system with little to no modifications.
- The hub should be commonly accessible.

2.5.1 HomePort

HomePort is a free open and source software solution developed as a research project at the Institute of Computer Science at Aalborg University [8, 1]. HomePort is a software solution that the user needs to install on a computer that will then act as the hub.

HomePort supports communication with Phidget USB devices and expose them over the HTTP protocol. Other applications can control the Phidget devices and read their state by issuing requests to HomePort.

When using HomePort in [30] we experienced the following issues.

- We found that the Phidget adapter was not compatible with the most recent version of HomePort.
- We had difficulties getting an older version of HomePort with a compatible Phidget adapter running on OS X.
- We got the older version of HomePort running on a Windows machine but found that it would frequently crash requiring a reboot of the HomePort software.

2.5.2 SmartThings

SmartThings [33]¹ is commercial home automation hub offered by Samsung. SmartThings is a hardware device that comes preinstalled with the necessary software.

Protocols supported by SmartThings include ZigBee and Z-Wave as well as devices communicating over WiFi within the local network [54]. SmartThings expose the devices over HTTP using a REST API [55].

While exposing devices using a REST interface, Samsung also provides a concept of SmartApps, applications installed in the cloud, *i.e.* on Samsungs servers. These applications react to events within the users SmartThings environment, *i.e.* a new temperature reading and triggers other events based on the new information. SmartApps can also run events periodically.

At the time of writing the retail price for the hub is \$99.

2.5.3 OpenHAB

Unlike SmartThings, openHAB [47] does not come with the hardware needed to run the hub. openHAB is based on Eclipse SmartHome², an open source framework for building software for a smart home. Amongst others, the framework provides mechanisms for handling data (including a common type system) and a rule engine. openHAB and Eclipse SmartHome are both Java based software solutions that support any platform capable of running a JVM [46].

openHAB is free and open source maintained by a community of enthusiasts. It supports a wide range of devices, 121 as of writing [48]. Amongst these are devices communicating using ZigBee, Z-Wave, MQTT and HTTP. openHABs concept of adapters is called “bindings” and these can be specific to a protocol or a product. For example, the community has developed bindings for the HTTP, Bluetooth and MQTT protocols as well as for the Philips Hue, Netatmo, and Sonos products.

The devices connected to an openHAB environment is exposed over HTTP.

¹More information about SmartThings is available at <https://www.smarththings.com/products/hub>

²The Eclipse SmartHome project can be found at <https://www.eclipse.org/smarthome/>

2.5.4 Conclusion

While the SmartThings hub is a consumer oriented solution which is easy to install, we want to keep the price of our solution to a minimum and thus prefer a free software solution rather than purchase the SmartThings hub.

The choice between HomePort and openHAB came down to the following two things.

- We experienced various issues with HomePort. In a previous project [30] we found that the Phidget adapter is incompatible with the most recent version of HomePort, that we were unable to run previous a previous version on OS X and that the same version would frequently crash on Windows.
- Furthermore we found that the community developing openHAB has created bindings for hardware we already have access to and assuming these work, we will not need to create our own adapters.

The main focus of this report is not the research or development of a smart hub and therefore we do not intend to spend much time installing and configuring a hub. Hence we think that openHAB is the best fit for this project and will be our choice of hub.

2.5.5 Hardware

The hardware that, together with openHAB, constitutes the hub should fulfill the following requirements.

- Compatible with openHAB as this is our software of choice.
- Low cost in order to reduce the barrier of entry.
- Small in order for it to easily fit into any place of a home, *e.g.* a closet.

openHAB runs on any computer that can run a Java virtual machine (JVM) [46]. It can be installed on desktop computers running Windows, Linux or OS X.

Typical desktop computers are not a great fit for this project as they are typically large and take up more space than recently introduced tiny computers such as the Raspberry Pi. The tiny computers are typically assembled from hardware components that are less powerful than the components installed in desktop computers. Therefore tiny computers consume less power than traditional desktop computers thus reducing the long-term cost of running the hub.

Furthermore tiny computers such as the Raspberry Pi Zero are priced as low as \$5 dollars [63]. While the Pi Zero requires an adapter to be connected to the Internet, the total cost of the solution is still less than a setup using a desktop computer. Recently Adafruit, the company behind the Raspberry Pi, released the Raspberry Pi 3, which is the first Raspberry Pi with built-in WiFi and BLE.

Raspberry Pi is completely compatible with openHAB and is one of the recommended computers for running openHAB [45]. Therefore we choose to base the hub on a Raspberry Pi but the hub should be able to run on any machine that can run a JVM.

2.6 Indoor Positioning

Indoor positioning concerns determining the position of a device placed indoors. For outdoor positioning, the Global Positioning System, shortened GPS, can be used to determine the position of a device. GPS can be unreliable indoor because the waves from the satellites used to position the device are weakened and scattered by the roof and walls of a building as well as the objects inside and outside the building.

Instead, alternative technologies can be used to estimate the position of a device, or a user carrying a device, while the user is indoor. As per [30] we differentiate between the following two types of indoor positioning.

Ranging Granular positioning of the user in which we attempt to determine his precise location in a room. Trilateration can be used to estimate the users position given a minimum of three *anchors*, e.g. WiFi hotspots.

Region monitoring Coarse grained positioning of the user in which we determine which region of a larger area the user is located in. A region is specified by one or more anchors. We determine the user to be in the region which contains the anchor from which we receive the strongest signal.

[30] investigates solutions based on various technologies for positioning the user indoors. The technologies include WiFi, ultra-wideband, Bluetooth Low Energy, shortened BLE, as well as the accelerometer. Ranging using BLE beacons was proven to have an average accuracy of 2.92m [30, p. 63]. The accuracy is not high enough to perform ranging in a home or apartment of 90 square meters with 3-4 rooms, where each room would be an average of 23-30 square meters. Because we have previously seen bad results when performing granular positioning indoor, we choose to focus on region monitoring and determining which room in an apartment the user is situated in.

WiFi, ultra-wideband and BLE are all technologies that can be used to perform region monitoring but we choose to use BLE beacons to position the user in the apartment because we have experience with the technology from [30] and more specifically we have experience with beacons from Estimote, a company producing BLE beacons, as well as the SDK for reading RSSI values from the beacons. Refer to Section 3.1 for more information on positioning using BLE beacons.

2.7 Recommender System Methods

Given the gesture a user has performed and his location in a smart home, *i.e.* contextual information, we want to determine an appropriate action to trigger thus changing the state of one or more devices in the smart home. The following section focus on approaches for determining the appropriate action.

It is our hypothesis that we can use concepts from the field of machine intelligence to determine the action to trigger given contextual information. Recommender systems are designed to recommend some *item* to the user given information about the domain the system is operating in. For example, the video streaming service Netflix can utilize a recommender system to recommend movies to its users based on movies the user has previously expressed interest in, *e.g.* through ratings. In the case of the system presented in this report, *items* are actions that change the state of one or more smart devices in a smart home. The recommendation is based on the provided contextual information.

This section presents selected techniques for creating a recommender system. The strengths and weaknesses of the techniques are outlined and the technique used in this project is decided.

2.7.1 Collaborative Filtering

Collaborative filtering is a technique that uses information about multiple users to recommend items for a given user. In a setting in which it is used to recommend movies, a source of information could be user-submitted movie ratings. Based on information about the movies a user has rated highly, such as genres, actors and release year, the recommender system begins acquiring the preferences of that user. The system then find users with similar preferences and look at which movies they have rated highly. These movies can be suggested to other users.

Consider the example of John. Because John has given high ratings to western movies from the 50s, the system knows that John likes those movies. The system then looks for users with a similar taste in movies, *i.e.* users who have given high ratings to western movies from the 50s. If the set of movies retrieved from the users with a similar taste in movies contains a movie John have not watched, the movie can be suggested to John. Note that it is key that both John and the users have given high ratings to the movies. As a consequence, if users with similar taste in movies have given the western movies from the 50s low ratings, these are not recommended to John.

One of the issues with collaborative filtering according to [10] is that it depends heavily on users actually providing ratings, otherwise the recommender system will have nothing to base its recommendations on. This issue will occur every time a new item is added as users have yet to rate it. In addition, when a new recommender system is created, the system will suffer from lack of data and every user will be afflicted by this “Early rater problem” for all items.

2.7.2 Content-based Filtering

A different method to compensate for the “Early rater problem” is content-based filtering which relies on the correlation between a user’s preferences and the information about items [10]. *e.g.* John, the user who likes western movies from the 50s, is more likely to be recommended other western movies from the 50s regardless of how other users have rated the recommended movies. One of the challenges of this technique is to create proper classification of items and their attributes along with user profiles such that they can be matched. [49] classify web pages by analysing the word frequencies, excluding common English words. To determine the preferences of the user some form of relevancy feedback is required, either positive or negative. An example of this is to have the user categorize the web pages he visits as either hot, luke-warm or cold [49]. It is also possible to use implicit feedback such as the time spent on a web page [64]. However when using implicit feedback one has to be cautious with how strong belief is put into each observation. One could look at ignored links as being negative feedback on those pages, but perhaps the user did not dislike the link but rather did not see it [64].

2.7.3 Decision Trees

Decision trees are a model for decision making and prediction using a graph structure where leaf nodes represent outcome and non-leaf nodes test attributes or random variables. The branches that leave non-leaf nodes are labeled with the value of the corresponding attribute. Decision trees need not be balanced trees so some paths may end in a leaf node after very few nodes.

A decision tree is built on rules, *e.g.* $(A \wedge B) \Rightarrow C$. That is, if both A and B are observed, then C applies. Decision trees are commonly used for content-based filtering [3].

2.7.4 Bayesian Networks

Bayesian networks are a model for classification. The representation is a directed acyclic graph where each node represents a random variable and edges between represent conditional dependencies. This means that any child node is conditionally dependent on its parents. Each node or item in the network has a probability distribution that may change when evidence is observed. A node representing the probability for a given disease may be conditionally dependent in such a way that observing the symptoms of a patient will change the likelihood that the patient is afflicted. One of the advantages of Bayesian networks is that they can handle missing data well [31].

A Bayesian network is based on joint probabilities, *e.g.* $P(G, S)$ is the joint probability of events G and S. $P(A|G, S)$ is the probability of event A given information about events G and S.

In the system designed in this report, we are interested in finding the probabilities of events occurring based on the observations of other events. More specifically, we want to determine an action to trigger based on the performed gesture and the room the user is in.

As per the requirements presented in Section 1.6, we are interested in presenting the user with actions he is likely to trigger when we are unable to determine which action he desires to trigger. Working with probabilities is a good way to fulfill this requirement. When working with probabilities, an action can be triggered if it is the only action with a sufficiently high probability, P . When no action has a probability exceeding P or more actions have probabilities exceeding P , the actions can be suggested to the user.

2.7.5 Conclusion

A multitude of different solutions have been proposed for recommender systems [3] with just a few presented in this report. Collaborative filtering, presented in Section 2.7.1, can be a useful approach when different users are comparable and the interests of other users may have a positive impact on a given user.

However in this project we focus on single user needs and do not see the behaviour of other users as applicable to a given user. Hence we deem content-based filtering to be a better fit for this project.

Decision trees, described in Section 2.7.3, though prevalent for use in content-based recommender systems, do not handle missing data as well as Bayesian networks and thusly we deem Bayesian networks a better suited model for the purposes of this project. Furthermore Bayesian networks is highly based on probabilities, a desired feature in our system as we will see in Sections 3.3.2 and 3.3.3, and is a natural way to model the contextual information.

2.8 Bayesian Network

Determining the correct action to trigger based on the contextual information is done using a Bayesian network. In this section, we will describe key aspects of a Bayesian network.

A Bayesian network is modelled as a graph which is composed of nodes and edges between the nodes. Each node in the network represents a variable and edges connect two nodes. If there is a causal relationship between two nodes, the edge is directed. If there is only some correlation between two nodes, the edge is undirected [60].

Two nodes, A and C are said to be conditionally independent if there is no edge between them. If we introduce a variable B and directed edges from B to A and B to C, both A and C are said to be dependent on B. B is totally independent because it does not depend on other nodes whereas A and C are said to be conditionally independent given B [60]. The network is illustrated in figure Figure 2.9.

For each variable in the Bayesian network there is a probability distribution function which depends on the edges leading into the variable. Probability distribution functions can be illustrated using *probability tables*. A probability table is said to be *conditional*, i.e. it is a *conditional probability table* if the variable to which the table belongs depends on one or more variables.

Probabilities are denoted $P(A|B)$, meaning the probability of A given B. An alternative notation is $P(A = a|B = b)$, meaning the probability of variable A having value a , also referred to as being *in state a*, when B is in state b . We have that $P(A|B, C) = P(A|B)$ when A and C are conditionally independent as is the case in Figure 2.9. The joint distribution of nodes, $X = X_1, \dots, X_n$, in a network is computed as shown in Equation (2.3) [60].

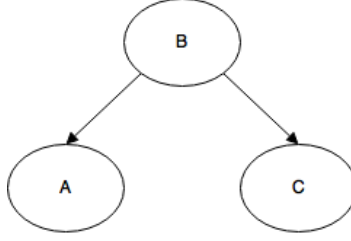


Figure 2.9: Simple Bayesian network. The network consist of three nodes, i.e. three variables: A, B and C. A and C depend on B and B is totally independent [60].

$$P(X) = \prod_{i=1}^n P(X_i | \text{parents}(X_i)) \quad (2.3)$$

In case of the network illustrated in Figure 2.9, the joint distribution of all variables is as shown in Equation (2.4) [60].

$$P(A, B, C) = P(A|B) \cdot P(B) \cdot P(C|B) \quad (2.4)$$

We also have that if the node x_0 does not depend on x_n , then we can remove x_n when calculating the probability of x_0 as shown in Equation (2.5) [60].

$$P(x_0 | x_1, \dots, x_n, \dots, x_k) = P(x_0 | x_1, \dots, x_{n-1}, x_{n+1}, \dots, x_k) \quad (2.5)$$

A graphical model of a Bayesian network consists of a set of nodes V , a set of edges E between the nodes and a set of probability distributions P for each variable [60]. In case of Figure 2.9 we have that $V = \{A, B, C\}$ and $E = \{\{B, A\}, \{B, C\}\}$. An example definition of P can be the probability table and conditional probability tables shown in Table 2.2. For example, the tables show that $P(B = \text{Yes}) = 0.3$ and $P(A = \text{No} | B = \text{Yes}) = 0.8$. This is the *prior distribution*, i.e. our beliefs before any evidence is observed.

Table 2.2: Sample probability table and conditional probability tables for nodes A, B and C in the network illustrated in Figure 2.9.

B		A		C	
Yes	No	Yes	No	Yes	No
0.3	0.7	Yes	0.2 0.8	Yes	0.9 0.1
		No	0.7 0.3	No	0.3 0.7

2.8.1 Evidence

Bayesian networks are used in order to reason under uncertainty. When reasoning under uncertainty we update the beliefs of some events based on observations of other events, i.e. *evidence* on those events [38, p. 90].

When using a Bayesian network, we assign evidence to the states of a node. If a node, X , in a Bayesian network has states x_1 , x_2 and x_3 then an evidence function $\epsilon_X = (1, 0, 0)$ indicates that $X = x_1$, i.e. node X is in state x_1 with certainty. If $\epsilon_X = (1, 2, 0)$ then we are sure that X is not in state x_3 and $X = x_2$ is twice as likely as $X = x_1$ [38, pp. 23-24]. The probabilities

for a column in a conditional probability table should have a sum of 1. That is for a node X , $P(x_1) + P(x_2) + \dots + P(x_n) = 1$ where n is the number of states in X .

We distinguish between *hard evidence* and *soft evidence* [38]. If an evidence function assigns a probability of zero to all but one state in a node, then there is *hard evidence* on that one state. If an evidence function assigns an evidence greater than zero to multiple states, there is said to be *soft evidence* on the states.

Bayesian inference or *belief propagation* is the action of updating beliefs of variables when evidence is observed in the network. This means that the *posterior distribution*, i.e. the probability after evidence is observed, is computed.

Let x_1, \dots, x_n be the prior distributions for node X with n states and ϵ the soft evidence observed for X where ϵ_i is the soft evidence for state x_i . Then the posterior probability of x_i , $P(x_{b,i})$ is computed as shown in Equation (2.6).

$$P(x_{b,i}) = \frac{x_i \cdot \epsilon_i}{\sum_{i=1}^n x_i \cdot \epsilon_i} \quad (2.6)$$

2.8.2 Hugin

The Hugin Tools³, also referred to as just Hugin, amongst others consist of the Hugin Decision Engine and the Hugin Graphical User Interface [36]. Hugin can be useful for modelling Bayesian networks and is used in the design phase of our system.

The Hugin Decision Engine provide functionalities for constructing, learning and analysing Bayesian networks and influence diagrams. The engine supports manual creation of networks and learning the structure of networks based on provided data [36]. Manual creation of networks can be done by experts in a field who knows the relationships between nodes. If the relationship is not known but the data is available, the structural learning can be used to build the network with no knowledge of relationships.

The graphical user interface builds on top of the decision engine to provide an easy-to-use interface for creating and running Bayesian networks and influence diagrams [36].

Figures 2.10 and 2.11 show screenshots of the Hugin graphical user interface. In Figure 2.10 the screen is split vertically, showing the network on the right-hand side and the beliefs and evidence of all nodes on the left-hand side. In Figure 2.11 the screen is split horizontally showing the network at the bottom and the probability tables and conditional probability tables at the top.

The features of the software used in this project are highlighted on the screenshots with black circles and annotated with a letter referencing the below description of the features.

- A. Enter editing mode shown in Figure 2.11. The mode is used when editing edges, nodes and probability tables in the network.
- B. Enter running mode shown in Figure 2.10. The mode is used when computing beliefs in the network given some evidence on the nodes.
- C. Indicates hard evidence on one of the states in the node.
- D. Indicates soft evidence on one of the states in the node.
- E. Green bars show beliefs of states in a node. For example, in Figure 2.11 the “Television: on/off” state of the Action node has a belief of 28.61%.
- F. Blue bars show soft evidence on states of a node. For example, in Figure 2.11 the “Bedroom” state of the Room node has soft evidence of 80% where as the “Living room” state has soft evidence of 20%.

³The Hugin Tools are available at <http://www.hugin.com>.

2. ANALYSIS

- G. Red bars show hard evidence on one of the states in the node. For example, in Figure 2.11 hard evidence is applied to the “Yes” state of the TV_IsOn node.
- H. Example of a conditional probability table in which probabilities given some states are entered. The first column in the screenshot configures the probabilities for both the System_State, Room_Action and Gesture_Action nodes being in state “Music centre: play/pause”. Note that Hugin supports normalizing the probabilities.

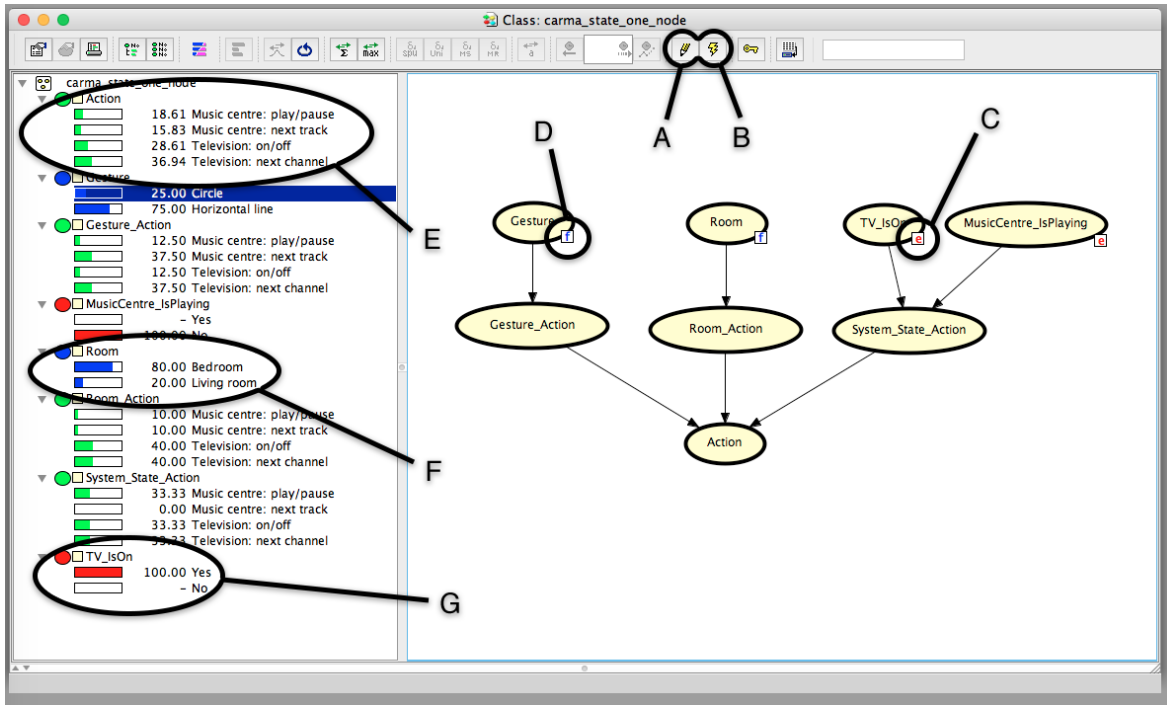


Figure 2.10: Screenshot of Hugin while running the Bayesian network. The screenshot highlights important features of the software. See above for a description of the features.

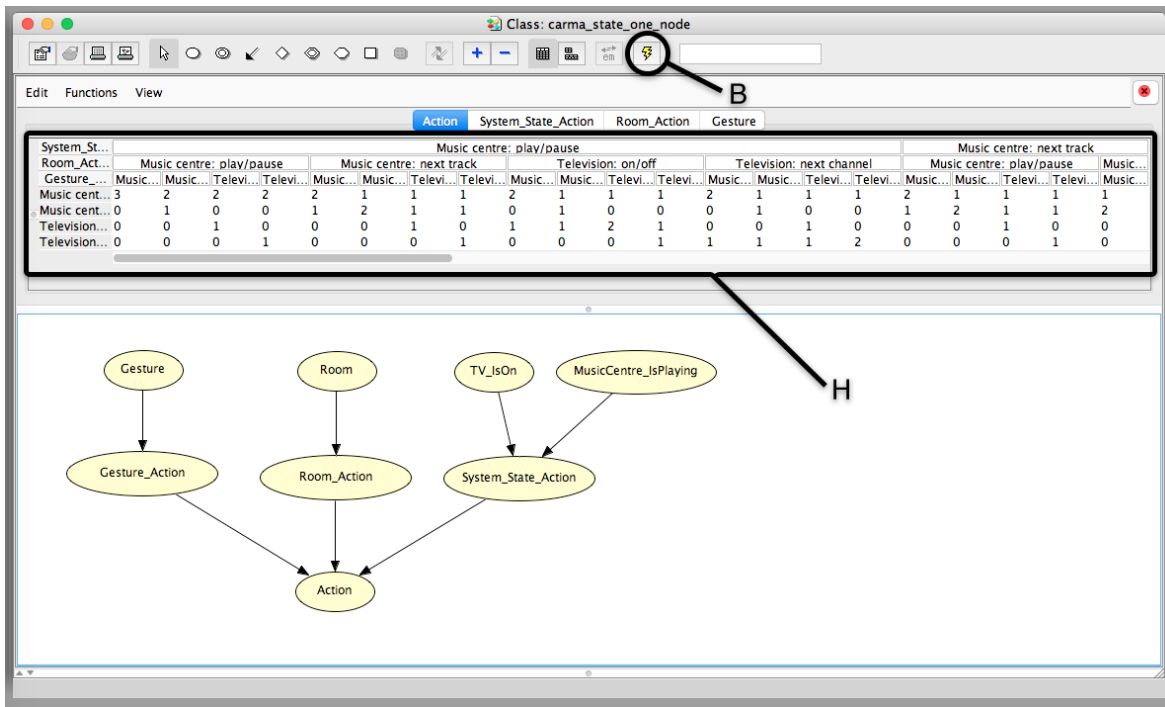


Figure 2.11: Screenshot of Hugin while editing the Bayesian network. The screenshot highlights important features of the software. See above for a description of the features.

2.9 Context Engine

As presented in Section 1.6, the system should be context-aware in order to ensure that the correct actions are triggered on the correct devices in accordance with the user's intention. To accomplish this we propose what we call a *context engine*. This context engine should ideally encompass all of the contextual information regarding the system, such as which gesture the user performed and where he is located. Using this information the context engine will determine which device the user intended to control as well as which action he intended to perform on it. Since gestures are part of the context, the user can use the same gesture for multiple devices assuming the entire context is not identical. This allows for a reduced set of gestures that the user has to recall. We suggest a design of the context engine based on the following requirements.

- While this project is focused on combining gesture recognition with the position of a user, the design should make no assumptions about the type of supplied contextual information.
- The design should support suggesting multiple actions if it is unsure which action the user desired to trigger.
- The design should support an arbitrary number of contextual information with zero being an exception as it does not make sense to determine the context with no information about it.

The context engine can be regarded as a recommender system which given some input, in our case the gesture performed by the user and his position in the smart home, recommends one or more actions the user may desire to trigger. Section 2.7.4 described the technique chosen for the context engine. Section 3.3 describes the design of the context engine.

CHAPTER 3

Design

This chapter describes the design of core components of the system.

3.1 Positioning Using BLE

In Section 2.6 various technologies for positioning a user was presented. We chose to use Bluetooth. The following section describes how we utilize Bluetooth for indoor positioning.

In [30] we used Estimote beacons for indoor positioning. The beacons are BLE-enabled devices that can be installed throughout a house. The beacons continuously emit a signal that other BLE-enabled devices can pick up. Estimote beacons can use either the iBeacon protocol or the Eddystone protocol. The two protocols and their uses are described below.

3.1.1 iBeacon

The iBeacon protocol is designed by Apple. When using the iBeacon protocol, a beacon will broadcast the following three identifiers [18, ch. 1].

UUID A Universally Unique Identifier is 16-byte string specific to a deployment of a set of beacons [14]. For example, if McDonalds deployed iBeacons in all of their restaurants, the beacons would all have the same UUID letting their application know which beacons to look for [18, ch. 1].

Major The major value is a 2-byte string that distinguishes a smaller group of installed beacons [18, ch. 1][14]. For McDonalds, the major value could be specific to an installation of iBeacons in a certain restaurant.

Minor The minor value is a 2-byte string that further distinguishes a group of beacons [18, ch. 1][14]. McDonalds could use this to distinguish between the floors of a restaurant.

Measured Power A value, measured in dBm, that helps BLE-enabled devices with ranging accuracy [4].

The Measured Power is broadcast by the beacon in order for other BLE-enabled devices to more accurately determine a granular proximity to the beacon. The value is measured using the following method and is performed using an iPhone 5s [4].

1. The iPhone 5s must be held in portrait orientation with nothing covering the top half.
2. RSSI values from the broadcasting beacon is recorded with a distance of 1 meter to the beacon. This should be done for a minimum of 10 seconds.

3. The highest 10% and the lowest 20% of the values are discarded.
4. The remaining values are averaged. This average constitutes the Measured Power value.

Tables 3.1 to 3.3 show examples of packets McDonalds could broadcast in order to retrieve information about a users positioning. Amongst others, McDonalds could use this for tracking users and sending them advertisements or offers based on their position.

In the examples in Tables 3.1 to 3.3, 664E170A–21BB–457C–A86B–F9265595452A is the UUID of all McDonalds’ beacons, a major value of 1 indicates a restaurant in Copenhagen and a major value of 2 indicates a restaurant in Aalborg. Beacons installed on first floor has a minor value of 1 and beacons installed on second floor has a minor value of 2. Note that all examples are fictitious.

Table 3.1: Example package emitted by one of McDonalds’ beacons installed on first floor in Copenhagen.

UUID	664E170A-21BB-457C-A86B-F9265595452A
Major	1
Minor	1
Measured Power	-44

Table 3.2: Example package emitted by one of McDonalds’ beacons installed on second floor in Copenhagen.

UUID	664E170A-21BB-457C-A86B-F9265595452A
Major	1
Minor	2
Measured Power	-44

Table 3.3: Example package emitted by one of McDonalds’ beacons installed on first floor in Aalborg.

UUID	664E170A-21BB-457C-A86B-F9265595452A
Major	2
Minor	1
Measured Power	-44

3.1.2 Eddystone

Eddystone is an open protocol designed by Google [13]. The Eddystone protocol supports the following *frame types*, *i.e.* different types of packets that can be broadcast [25].

Eddystone-UID

Eddystone-UID frames are broadcast when interested in identifying groups of beacons or particular beacons within a group. The beacon emits the following information [27]:

Namespace 10-byte identifier. Groups a particular set of beacons, similar to the UUID in the iBeacon protocol

Instance 6-byte identifier. Specific for each beacon within the namespace.

Tx Power The received power at 0 meters from the beacon, measured in dBm. Similar to the Measured Power in the iBeacon protocol. Resolution of 1 dBm.

The Eddystone-UID frame type is suitable for granular indoor positioning as the packet includes the Tx Power. Furthermore it is suitable for non-granular indoor positioning as each room can be identified using the instance.

Table 3.4 shows an example of a frame with the UID frame type.

Table 3.4: Example of an Eddystone-UID frame [27].

Byte offset	Field	Description	Value
0	Frame type	Type of the frame. UID, in this case.	0x00
1	Ranging data	Tx Power. Signed 8-bit integer.	0xD4
2	NID[0]	10-byte namespace.	0x3C
3	NID[1]		0x7B
4	NID[2]		0xB8
5	NID[3]		0x1A
6	NID[4]		0xEE
7	NID[5]		0x6E
8	NID[6]		0x97
9	NID[7]		0x1D
10	NID[8]		0x74
11	NID[9]		0x3B
12	BID[0]	6-byte instance.	0x0F
13	BID[1]		0x27
14	BID[2]		0x15
15	BID[3]		0x23
16	BID[5]		0x11
17	BID[6]		0xA6
18	RFU	Reserved for future use.	
19	RFU	Reserved for future use.	

Eddystone-URL

Eddystone-URL frames are broadcast when interested in redirecting users to a website. The frame is used in the Physical Web, in which beacons broadcast URLs that can be retrieved by users by tapping the beacon with their phone¹ [28]. The beacon emits the following information [28]:

Tx Power The received power at 0 meters from the beacon, measured in dBm.

URL Scheme Scheme prefix of the broadcast URL.

URL The broadcast URL.

Table 3.5 shows an example of an Eddystone-URL frame broadcasting the <http://goo.gl/POKXkv>.

¹More information about the Physical Web is available at <https://google.github.io/physical-web/>

Table 3.5: Example of an Eddystone-URL frame [28].

Byte offset	Field	Description	Value
0	Frame type	Type of the frame. URL, in this case.	0x10
1	Ranging data	Tx Power. Signed 8-bit integer.	0xD4
2	URL scheme	Value defined by the specification for the http:// prefix.	0x02
3	URL[0]	Encoded URL. Length 0 - 17.	g
4	URL[1]		o
5	URL[2]		o
6	URL[3]		.
7	URL[4]		g
8	URL[5]		l
9	URL[6]		/
10	URL[7]		P
11	URL[8]		O
12	URL[9]		K
13	URL[10]		X
14	URL[11]		k
15	URL[12]		v

Eddystone-TLM

Eddystone-TLM frames are broadcast when interested in monitoring the health of beacons. The Eddystone-TLM is meant to be transmitted along with either the Eddystone-UID or the Eddystone-URL. The beacon emits the following information [26]:

Version TLM version. Always 0x00 [26].

Battery voltage Voltage of the battery, *i.e.* remaining power, measured in 1 mv/bit.

Beacon temperature Temperature of the beacon, measured in degrees Celsius. Signed 8.8 fixed-point notation.

Advertisement count Number of frames advertised since the beacon was powered on or rebooted.

Seconds count Number of seconds since the beacon was powered on or rebooted. Resolution of 0.1 seconds.

Table 3.6 shows an example of an Eddystone-TLM frame that broadcasts 50% remaining battery power, a temperature of 24.25°C, 2674 frames advertised and an uptime of seven days, that is, 604800 seconds.

3.1.3 Conclusion

Both the iBeacon and the Eddystone protocols are suitable for the system described in this report. The iBeacon protocol solely supports positioning of the user, either granularly using the Measured Power or non-granularly using regions defined by the major and/or minor value of the beacon.

The Eddystone protocol supports more than positioning of the user. It also supports the monitoring the health of the beacon. While technically, Eddystone-URL frames can be used for positioning the user in regions based on the broadcast URL, *i.e.* different URLs for each regions,

Table 3.6: Example of an Eddystone-TLM frame [26].

Byte offset	Field	Description	Value
0	Frame type	Type of the frame. TLM, in this case.	0x20
1	Version.	TLM version.	0x00
2	VBATT[0]	Battery voltage. 1 mV/bit.	0x00
3	VBATT[1]		0xFF
4	TEMP[0]	Beacon temperature. Signed 8.8 fixed-point notation.	0x18
5	TEMP[1]		0x40
6	ADV_CNT[0]	Number of frames advertised since power-on or reboot.	0x00
7	ADV_CNT[1]		0x00
8	ADV_CNT[2]		0x05
9	ADV_CNT[4]		0x39
10	SEC_CNT[0]	Time since power-on or reboot.	0x00
11	SEC_CNT[1]		0x09
12	SEC_CNT[2]		0x3A
13	SEC_CNT[3]		0x80

the Eddystone-UID frame seems better suited for this purpose. Using Eddystone-UID regions can be identified based in the instance of a beacon.

While the iBeacon protocol is only officially supported by iOS devices, the Eddystone protocol is officially supported by both iOS and Android [13]. Since we are using a smartwatch running Android Wear for our prototype, we will use the Eddystone protocol.

In [30] we worked with granular positioning using the Measured Power in the iBeacon protocol. In this report we focus on non-granular positioning of the user. That is, we only determine the region which the user is currently in.

The smartwatch must pick up the frames advertised by the beacons and determine the region the user is in based in the namespace and instance of the Eddystone-UID frame. The smartwatch may also need to investigate the Tx power and RSSI received by the beacons, to determine which beacon the user is closest to if more beacons are available.

By analyzing the namespace, instance and Tx power in the frames advertised by the beacons, we can determine which region the user is in.

3.2 Gesture Recognition

As stated in Section 2.4 we decided to use the 1¢ gesture recognizer. However, as the 1¢ gesture recognizer is designed to be used for recognition of two-dimensional gestures we have created a modified version of it that is able to recognize three-dimensional gestures recorded using a three-axis accelerometer by utilizing one of the techniques used by \$3 [40].

3.2.1 Converting 1¢ to Three Dimensions

For use in this project we created a modified version of 1¢ that works with three-dimensional accelerometer data. A naive approach would be to use raw accelerometer data as coordinates

but the problem with this approach, besides lack of filtering, is that these coordinates would not properly represent a gesture.

Imagine a minimalistic example where the the accelerometer registers two consecutive accelerations, the first one is on the x -axis and the other is on the y -axis. The z -axis is ignored for the sake of the argument. This is illustrated by the two blue arrows in Figure 3.1. If these two measurements are used as coordinates for the gesture, it would end up looking like the gesture shown in Figure 3.2(a).

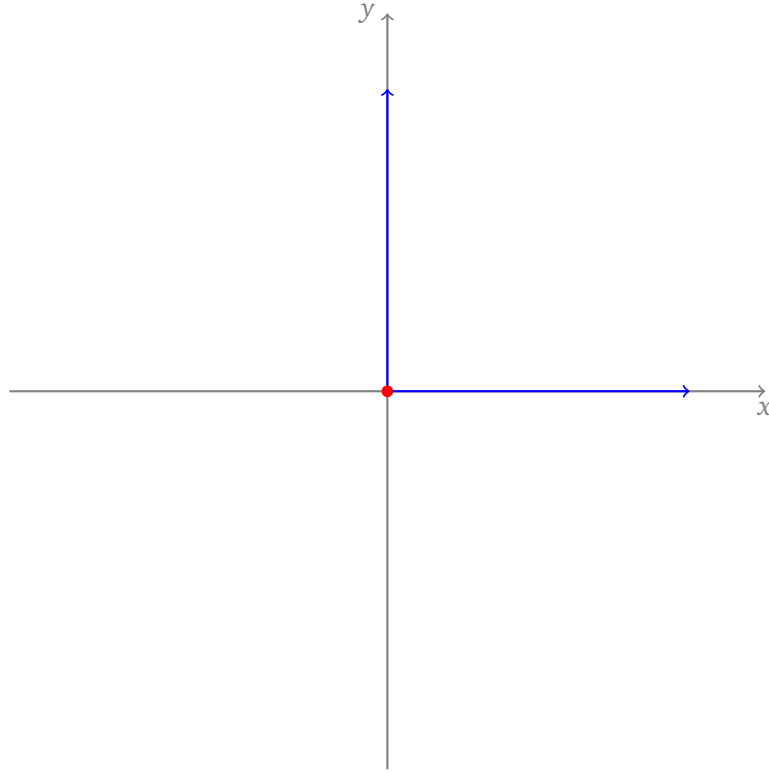
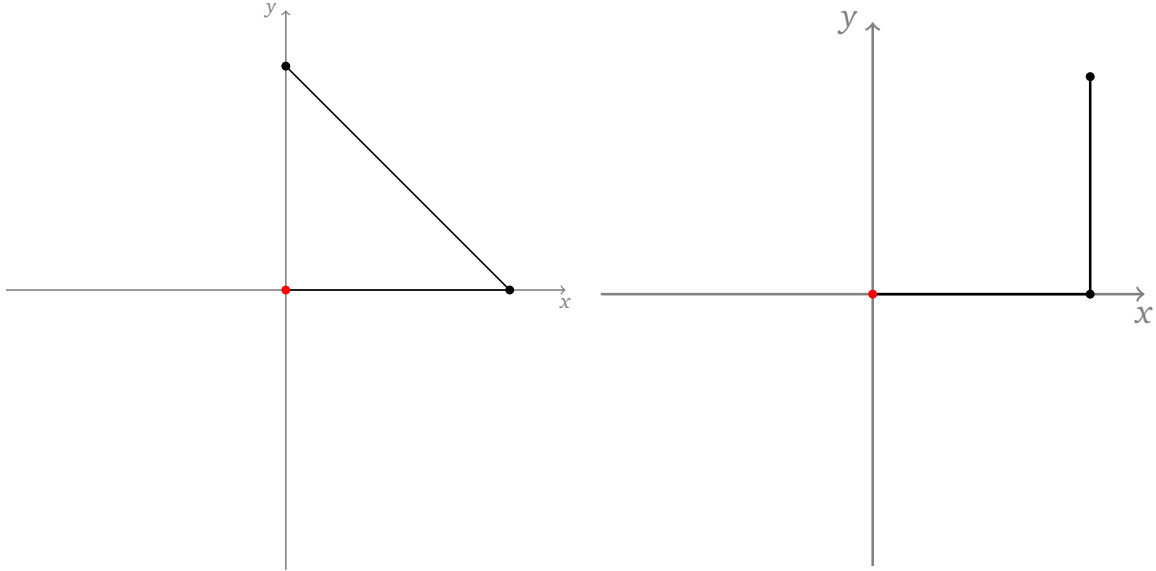


Figure 3.1: Two consecutive accelerometer measurements $\{4, 0\}$ and $\{0, 4\}$. Accelerations are measured in $\frac{m}{s^2}$

The proper path is illustrated in Figure 3.2(b). The way we achieve the proper path is by borrowing a technique from [40] where points are acquired by subtracting the current accelerometer measurement from the previous one to obtain an acceleration delta and adding it to the previous point to create the new point as shown in Figure 2.5. However, in our specific implementation we do not add the acceleration deltas to the previous point but instead add the current acceleration measurement.



((a)) Path drawn between the measured points by using ((b)) The proper path of the accelerometer-recorded gesture by adding accelerations together.

Figure 3.2: Minimalistic example of the difference between using accelerometer measurements directly as coordinates in a gesture trace compared to adding accelerations together as in §3.

3.3 Bayesian Network

The network is illustrated in Figure 3.3. We consider there to be the following three levels in the network.

- The uppermost level containing the “Gesture”, “Room”, “TV_IsOn” and “MusicCentre_IsPlaying” nodes provides contextual information.
- The middle level depends on, *i.e.* is children of, the uppermost level to provide probabilities for each action in the system based on a subset of the information observed in the uppermost level.
- The bottom level depends on all nodes in the middle level to provide probabilities for the actions in the system. Based on these probabilities the correct action to perform is determined.

The benefit of this structure is, that contextual information such as the performed gesture and the users position is always translated to probabilities for actions and these probabilities are independent of other contextual information that we may observe. When the contextual information is modelled as probabilities for actions, it becomes trivial to combine all the contextual information to choose the right action to perform.

The probability tables and conditional probability tables shown in Tables 3.7 to 3.12 only show selected states. In reality, the tables include states for all gestures, rooms and actions in the system but some have been left out for readability purposes.

Table 3.7 shows an excerpt of the probability table for the gesture node. The node contains prior beliefs for all gestures in the system and they should all have the same belief. This is because the user is equally likely to have performed the gestures when we have not observed any evidence. Based on the likelihoods of each gesture supplied by the gesture recognizer, we can add soft evidence to the Gesture node. The evidence is assigned as described in Section 3.3.2.

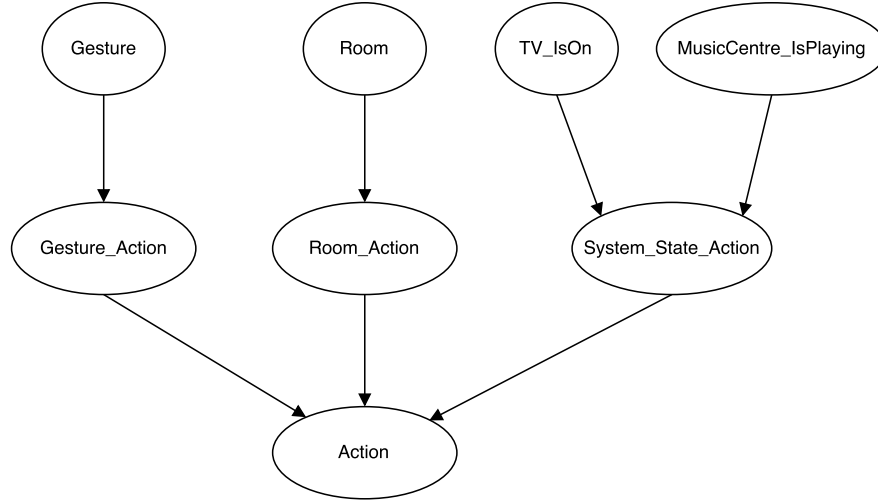


Figure 3.3: The Bayesian network used for determining the appropriate action to trigger based on the gesture performed by the user, the position of the user and the state of the system.

Table 3.7: Excerpt of the probability tables for the Gesture and Room nodes.

Gesture			Room	
Z	Half circle	Horizontal line	Bedroom	Living room
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0.5	0.5

Table 3.8: Probability tables for the TV_IsOn and MusicCentre_IsPlaying nodes.

TV_IsOn		MusicCentre_IsPlaying	
Yes	No	Yes	No
0.5	0.5	0.5	0.5

The Gesture, Room, TV_IsOn and MusicCentre_IsPlaying nodes constitute the uppermost level in the network. The middle level is constituted by the Gesture_Action, Room_Action and System_State_Action nodes.

The nodes on the middle level observe nodes in the uppermost level and based on the observed information, compute probabilities for all actions in the system. Without any evidence on a state, the probabilities of the states in nodes in the middle level are determined by gesture configurations.

Consider the prior beliefs of the Gesture_Action node as shown in Table 3.9. In the scenario presented in Section 1.2 we presented a configuration of the system. Table 3.9 shows an excerpt of the configuration in which the user has configured the Z gesture to turn Lamp 3 and Lamp 8 on and off. The Half Circle gesture skips to the next track on the music centre and changes to the next channel on the television. The Horizontal Line gesture skips to the previous track on the music centre and changes to the previous channel on the television. The valid actions for each gesture have the same probability.

As the Gesture_Action node depends on the Gesture node, the probabilities of each action are changed when evidence is added to the states of the Gesture node.

The principle of the Room_Action and System_State_Action nodes are the same as the Ges-

Table 3.9: Excerpt of the conditional probability table for the *Gesture_Action* node.

	Gesture_Action		
	Z	Half circle	Horizontal line
Lamp 3: on/off	0.5	0	0
Lamp 8: on/off	0.5	0	0
Music centre: next track	0	0.5	0
Music centre: previous track	0	0	0.5
Television: next channel	0	0.5	0
Television: previous channel	0	0	0.5

ture_Action node. Each state of the nodes correspond to an action in the system. The probabilities of the nodes change when evidence is added to the Room, MusicCentre_IsPlaying and TV_IsOn nodes.

The evidence added to the states of the Room node, is likely to be soft evidence as we adjust the probability of the user being in a room over time as he walks around as described in Section 3.3.3. The evidence of the MusicCentre_IsPlaying and TV_IsOn nodes are hard evidence as we can observe this with certainty from the state of items in openHAB.

Table 3.10: Excerpt of the conditional probability table for the *Room_Action* node.

	Room_Action	
	Bedroom	Living room
Lamp 3: on/off	0	$\frac{1}{3}$
Lamp 8: on/off	$\frac{1}{3}$	0
Music centre: next track	0	$\frac{1}{3}$
Music centre: previous track	0	$\frac{1}{3}$
Television: next channel	$\frac{1}{3}$	0
Television: previous channel	$\frac{1}{3}$	0

Table 3.11: Excerpt of the conditional probability table for the *System_State_Action* node.

	System_State_Action			
	Yes		No	
MusicCentre_IsPlaying				
TV_IsOn	Yes	No	Yes	No
Lamp 3: on/off	$\frac{1}{6}$	0.25	0.25	0.5
Lamp 8: on/off	$\frac{1}{6}$	0.25	0.25	0.5
Music centre: next track	$\frac{1}{6}$	0.25	0	0
Music centre: previous track	$\frac{1}{6}$	0.25	0	0
Television: next channel	$\frac{1}{6}$	0	0.25	0
Television: previous channel	$\frac{1}{6}$	0	0.25	0

Based on the posterior beliefs of the *Gesture_Action*, *Room_Action* and *System_State_Action*

nodes, the posterior beliefs of the states in the Action node are computed. The beliefs of the node are used to determine which action to trigger.

Table 3.12 shows a very small excerpt of the conditional probability table of the Action node. For readability purposes the table only contains two states, A and B. We assume A and B are actions in the system. For example, A = “Lamp 3: on/off” and B = “Music centre: next track”.

When $\text{System_State_Action} = A_1$, $\text{Room_Action} = A_1$ and $\text{Gesture_Action} = A_1$ there is a strong belief that action A should be triggered. The same applies for action A_2 when all three nodes are in state A_2 . If only two nodes are in state A_1 , then the probability of $P(\text{Action} = A_1) = \frac{2}{3}$ and as a result $P(\text{Action} = A_2) = \frac{1}{3}$ and vice versa.

Table 3.12: Excerpt of the conditional probability table for the Action node in the Bayesian network presented in Figure 3.3.

System_State_Action	Action							
	A ₁				A ₂			
	Room_Action							
	A ₁		A ₂		A ₁		A ₂	
Gesture_Action	A ₁	A ₂	A ₁	A ₂	A ₁	A ₂	A ₁	A ₂
A	1	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0
B	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	1

3.3.1 Alternative Model of System State

In Figure 3.3 both the TV_IsOn and the MusicCentre_IsPlaying nodes are parents of the System_State_Action node. One may propose an alternative model for the system state as shown in Figure 3.4.

The System_State action node have been replaced by the TV_IsOn_Action and MusicCentre_IsPlaying_Action nodes. Each observes a specific part of the system state. The conditional probability tables for the nodes are shown in Table 3.13.

Consider the table for TV_IsOn_Action with a subset of the actions presented in the scenario. As the node knows nothing about the state of the television, it must have equal probabilities for the two states when the television is on. However, if the television is off the node has zero probability of changing the channel on the television. The same principle applies for the MusicCentre_IsPlaying_Action node.

The consequences of the alternative model is illustrated using screenshots from Hugin shown in Figures 3.5 and 3.6. Note that the difference between the beliefs of the states in the Action node is smaller in Figure 3.6. This is the reason we choose to model the system state as presented in Figure 3.3 as it does not reduce the difference between states as significantly.

3.3.2 Evidence in Gesture Node

The following section describes how the evidence assigned to the states in the Gesture node of the Bayesian network is determined. In order to determine the evidence, we introduce the *gesture context provider*.

The gesture context provider determines the actions that make sense to trigger based on the motion performed by the user. When the user performs a motion, the gesture recognizer scores the trained gestures as explained in Section 2.4.2. When the gestures are scored, they are passed to the context provider which calculates the probabilities of each action associated with the gesture.

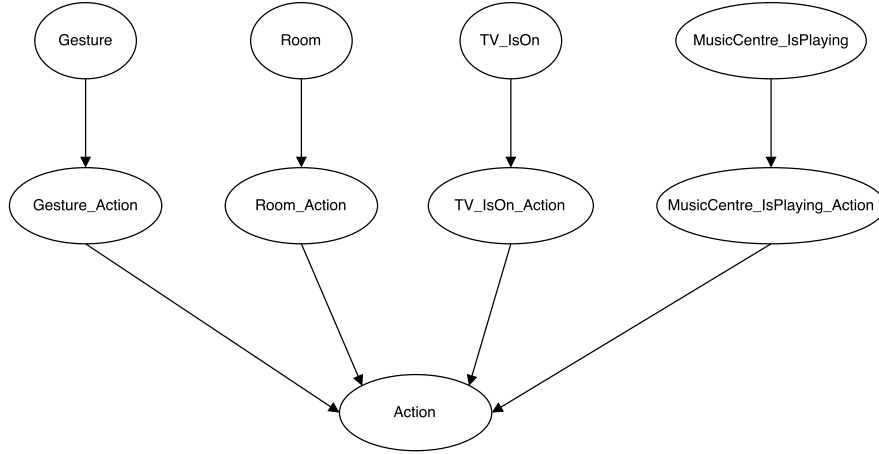


Figure 3.4: Alternative approach for modeling the system state in the Bayesian network.

Table 3.13: Probability tables for the *TV_IsOn_Action* and *MusicCentre_IsPlaying_Action* nodes in the alternative Bayesian network illustrated in Figure 3.4.

TV_IsOn_Action			MusicCentre_IsPlaying_Action		
	Yes	No		Yes	No
Music centre: play/pause	$\frac{1}{4}$	$\frac{1}{3}$	Music centre: play/pause	$\frac{1}{4}$	$\frac{1}{3}$
Music centre: next track	$\frac{1}{4}$	$\frac{1}{3}$	Music centre: next track	$\frac{1}{4}$	0
Television: on/off	$\frac{1}{4}$	$\frac{1}{3}$	Television: on/off	$\frac{1}{4}$	$\frac{1}{3}$
Television: next channel	$\frac{1}{4}$	0	Television: next channel	$\frac{1}{4}$	$\frac{1}{3}$

Because the gesture recognizer scores all trained templates, gestures that are obviously not correct are returned by the gesture recognizer. Therefore we introduce a threshold which the gesture scores must be below to be considered. We only consider gestures with a score of 70 or below. The threshold of 70 was determined by considering the scores of multiple gesture templates and we found that generally, gestures with a score of 70 or below can be considered as a properly recognized gesture.

The set of matches returned by the gesture recognizer are not unique per gesture name. A gesture named “Circle” may appear several times if the gesture trace matches multiple templates of the trained gesture templates. We take the average of each gesture, grouped by the name of the gesture.

The gesture recognizer uses the distance of the recorded template to each trained template as the score for the gesture. Thereby the lower the score, the better the result. When calculating the belief of gesture nodes, we create a *translated score* where a high score indicates a better match. Let $G = \{G_1, G_2, \dots, G_n\}$ be the set scores, then the translated score, G'_i is computed as $G'_i = \max(G) \cdot \frac{\max(G)}{G_i}$. This ensures the proportions between scores are the same.

For example, if we have recognized gestures with scores $G = \{40, 62\}$, then the translated scores are computed as $G' = \{62 \cdot \frac{62}{40}, 62 \cdot \frac{62}{62}\} = \{96.1, 62\}$.

Given the translated scores, we can calculate the beliefs as shown in Equation (3.1).

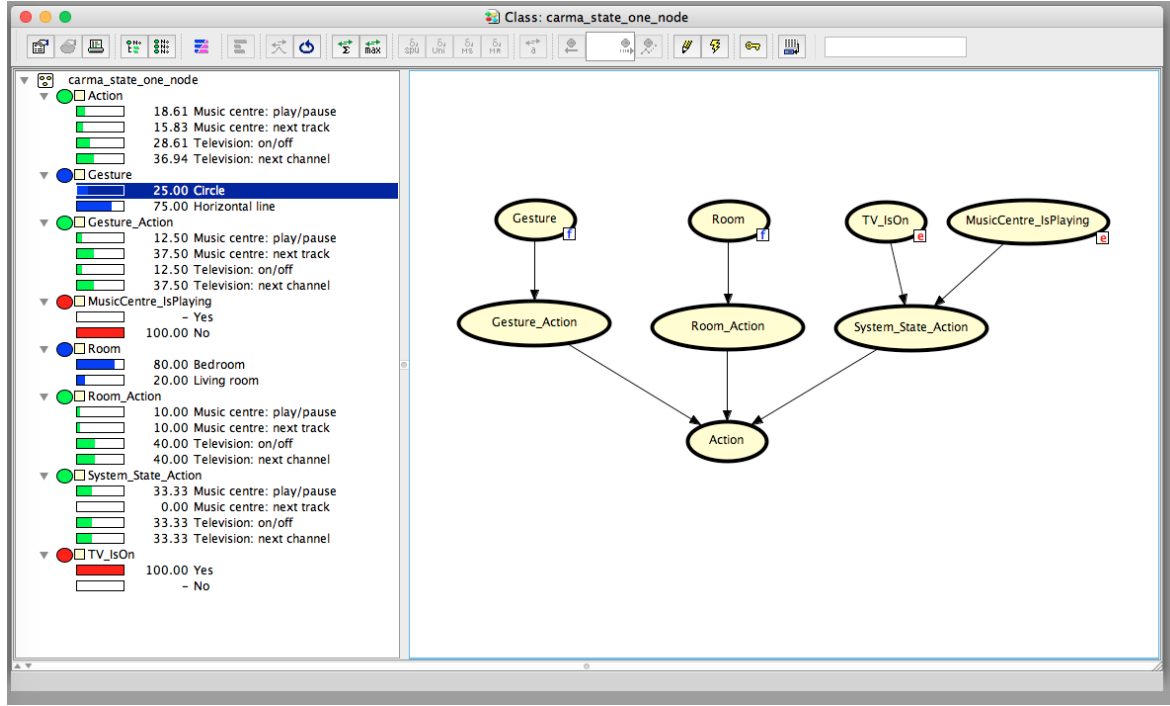


Figure 3.5: Screenshot from Hugin. Refer to Section 2.8.2 for a detailed description of the elements shown. The network shows beliefs of actions when using one node at the middle level for system state. Nodes are configured according to the scenario presented in Section 1.2. Conditional tables for the *TV_IsOn* and *MusicCentre_IsPlaying* nodes are configured as shown in Table 3.13.

$$B_i = \frac{G'_i}{\sum_{x \in G'} x} \quad (3.1)$$

3.3.3 Evidence in Room Node

The following section describes how the evidence assigned to states in the Room node of the Bayesian network is determined. In order to determine the evidence, we introduce the *position context provider*.

The position context provider is responsible for determining which actions make sense to trigger given the users current position. For each action it provides the likelihood of those actions being the intended ones. The likelihoods are the evidence assigned to the states in the Room node.

The context provider determines the position of the user using BLE. One or more Estimote beacons are installed in rooms that should be tracked. The beacons broadcast using the Eddystone protocol as described in Section 3.1. The smartwatch continuously scans for nearby beacons using the Estimote SDK.

The Estimote beacons delivers a set of discovered beacons several times each second. Each time beacons are discovered, the provider chooses the beacon with the highest RSSI. We assume that the higher the RSSI is, the closer the user is to the beacon as the RSSI is an indicator of the signal strength between the two Bluetooth devices.

In order to account for sporadic and false measurements indicating that the user is in the wrong room, we store a reference to the room in which the beacon with the highest RSSI is placed.

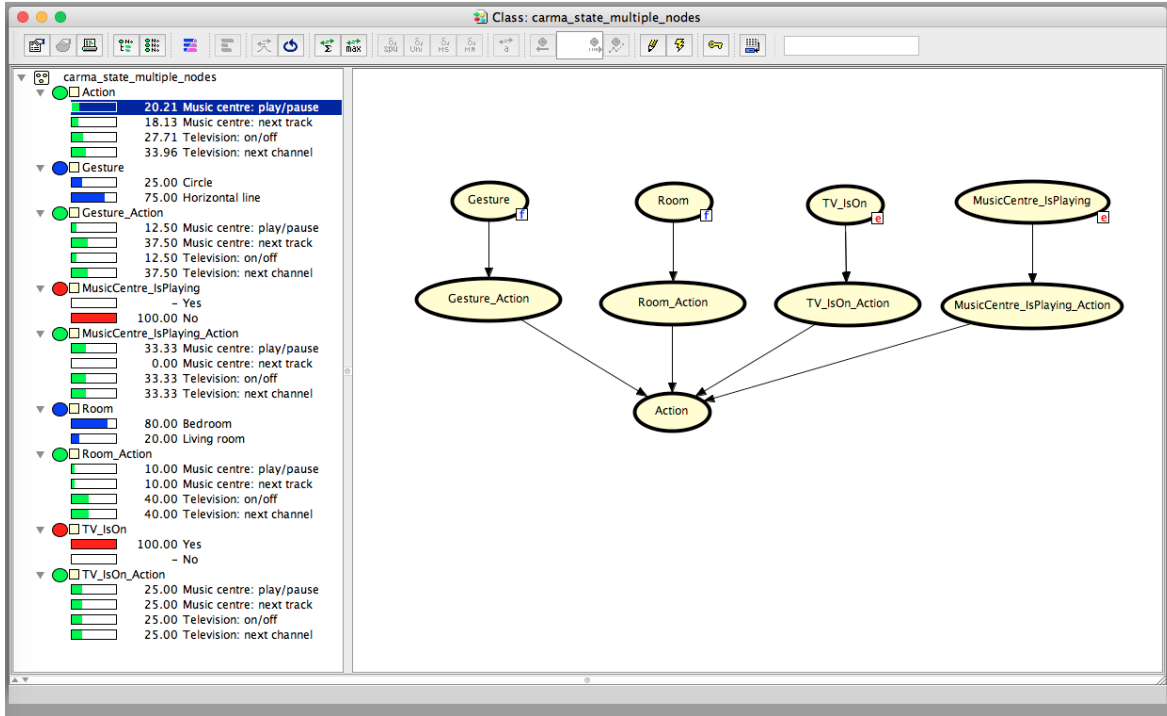


Figure 3.6: Screenshot from Hugin. Refer to Section 2.8.2 for a detailed description of the elements shown. The network shows beliefs of actions when using multiple nodes at the middle level for system state. Nodes are configured according to the scenario presented in Section 1.2. Conditional tables for the *TV_IsOn* and *MusicCentre_IsPlaying* nodes are configured as shown in Table 3.13.

Each reference lives in the queue for 30 seconds. We keep each reference for 30 seconds, because Estimote estimates that a region exit in their SDK usually takes up to 30 seconds and therefore we make the same assumption [12]. Whenever the context provider is asked to provide its context, we calculate the occurrence of each beacon in percentage. For example, consider the following set where R_1 , R_2 and R_3 are different rooms.

$$\{R_1, R_1, R_2, R_2, R_1, R_1, R_1, R_2, R_3, R_1, R_1, R_1\}$$

There are a total of twelve items in the queue. R_1 occurs eight times, R_2 occurs three times and R_3 occurs one time. Therefore we assign a 66.67% probability that the user is in R_1 , a 25% probability that the user is in R_2 and a roughly 8.33% probability that the user is in R_3 . The normalized probabilities are used as soft evidence of the states in the Room node.

3.3.4 Evidence in System State Nodes

In Figure 3.3 we suggested two nodes that supply information about the system state. The *TV_IsOn* supplies information about whether not the television is on and the *MusicCentre_IsPlaying* node supplies information on whether or not the music centre is playing music. Naturally these are examples of nodes that can describe the state of the system. Other examples can include if a lamp is on or not, a door is locked or unlocked or who is home.

Evidence on the system state nodes in the uppermost level is likely to be hard evidence, assuming that the state of the system retrieved from openHAB can be trusted. There could be

cases where the system state returned by openHAB is not the correct state. The following are examples of cases where we may not be able to trust the state stored in openHAB.

- Assume a television which can be controlled through openHAB. When a request to turn on the television is sent to openHAB, the request is forwarded to the television. openHAB can store the new state of the television. If some user decides to control the television using a regular TV remote, the request is not sent through openHAB and unless openHAB continuously polls for the state, the state of the television is never updated in openHAB. Fortunately, openHAB supports polling of states.
- When a request to change the state of an item is sent through openHAB, the new state can be stored by openHAB if the request was successful. For some protocols, *e.g.* UDP, it is not known if a message is delivered correctly and therefore openHAB cannot depend on storing the state of a device only if a request is successful, except if it only uses protocols that support this. Another issue with UDP is that the order of messages is unknown, hence it is unknown which state the device ends in.

When trusting the state stored in openHAB, the evidence can be hard. If the state is not trusted, one can resort to using soft evidence. *e.g.* based on historical data for each device.

3.4 Context Engine

In Section 2.9, we outlined the requirements for the context engine. According to the requirements the design of the context engine should support suggesting multiple actions, if a single action cannot be determined due to multiple actions having an approximately equal probability. The requirement is fulfilled by the nature of a Bayesian network as the result of running the network is beliefs on all states of all nodes. In the case of the Bayesian network shown in Figure 3.3 the beliefs of the Action node indicates which action the user may desire to trigger. If multiple actions have beliefs that are approximately equal within a threshold, the actions are presented to the user in a list and he can choose the appropriate action to trigger. We found that a threshold of four percentage points results in an acceptable behaviour. Actions which have a probability that is within four percentage points of the action with the highest probability, is considered *accepted*, *i.e.* they are presented on the list of possible actions.

The requirements also outline that the design should make no assumptions about the type of contextual information supplied to the context engine. We have decided to fulfill this requirement by enforcing a “translation” of probabilities in the uppermost level, *i.e.* the level containing the Gesture and Room node in our specific implementation, to probabilities of actions in the middle level. The consequence of this, is that the design makes no assumptions about the type of information but it enforces a uniformity of the parents to the Action node. Because the nodes at the uppermost level are independent of each other, we can implement it as two components with no knowledge to each other. This implementation allows us to easily extend the engine with new nodes providing contextual information as described in Section 4.4.

Furthermore the requirements mention that the design should support an arbitrary number of contextual information. While the design in principle fulfills the requirement, the conditional probability table of the Action node will grow exponentially when parent nodes are added. A solution for this can be to replace the conditional probability table with a deterministic function that, given the parent nodes as input, counts the number of nodes pointing at each state. In principle, this means that each parent node “votes” on which action should be triggered and the action with most votes is triggered. If multiple actions have the same amount of votes, the actions are presented to the user who can choose the correct action to trigger.

Implementation

The following chapter describes the implementation of the system. Jayes, the third party implementation used for the Bayesian network is briefly presented. We give a detailed description of the *context engine*, which provides developers with a flexible way of supporting new contexts in the system. The implementation used for communicating with OpenHAB is briefly described and lastly the prototype is presented.

4.1 Status

As part of the project we have designed and implemented a context-aware home automation solution on an Android Wear and a Raspberry Pi with Philips Hue light bulbs and a desktop machine running a Spotify client acting as the controllable devices.

Figure 4.1 shows a deployment diagram of the prototype developed during the project. Section 4.1.1 describes the devices involved in the deployment as well as the features implemented on the devices. In our specific prototype, we have a computer acting as the media centre and we have two Philips Hue light bulbs connected to openHAB. The number of devices will vary depending on the specific deployment, *e.g.* there can be more Philips Hue light bulbs or other devices in the system, *e.g.* door locks, a television or a thermostat.

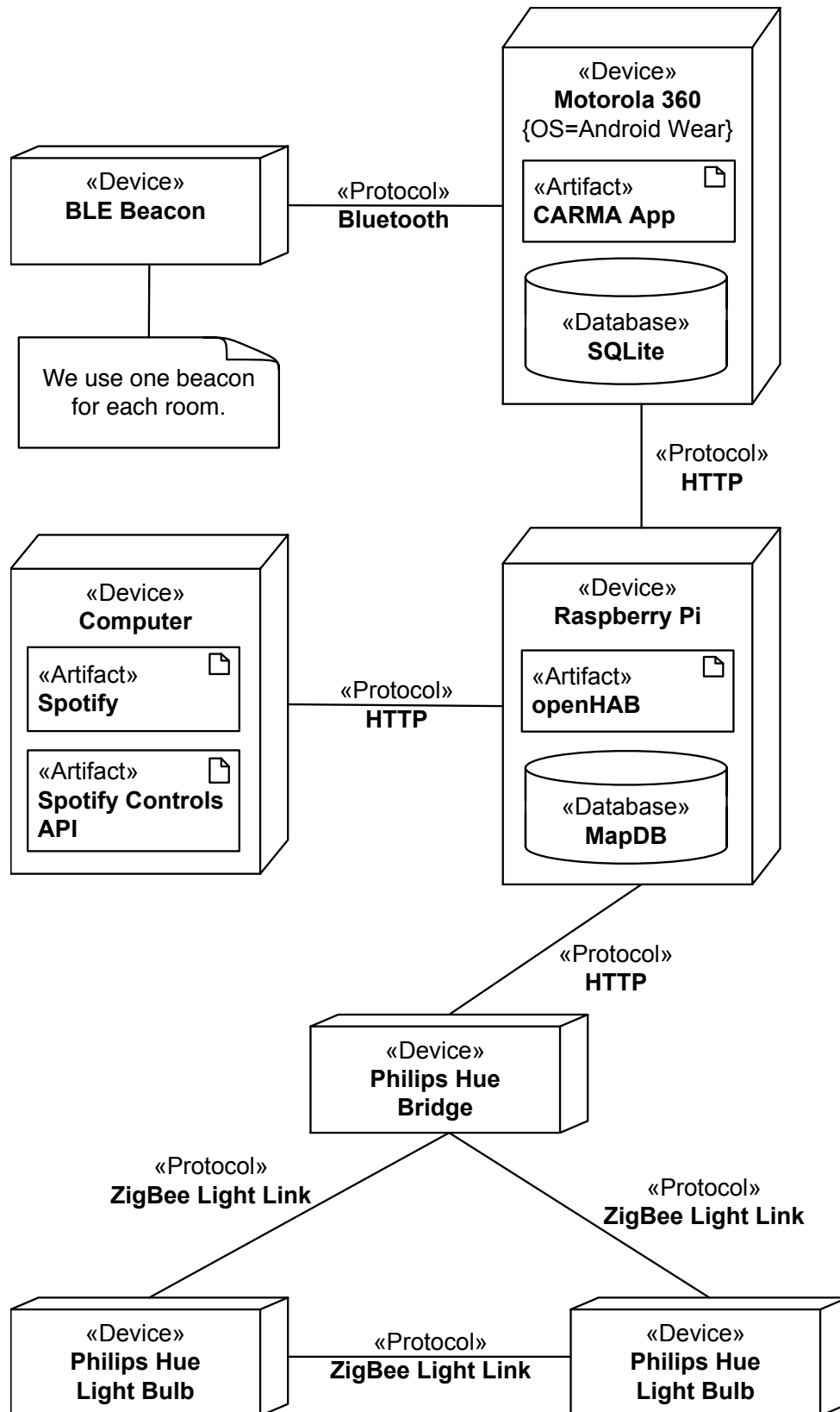


Figure 4.1: Deployment diagram of the prototype developed in the project.

4.1.1 Devices

Below we present the features implemented as part of the project. Features are grouped by the hardware the software is running on.

Smartwatch

The user utilizes the Android Wear smartwatch to perform gestures which starts the context recognition process. Furthermore the smartwatch is used for positioning the user and configuring parts of the system. The following features are implemented on the watch.

- Retrieval of available devices and actions from openHAB.
- Configuration of the smartwatch application based on rooms and beacons registered in openHAB.
- Training of gestures using the 1¢ gesture recognizer as described in Section 3.2.
- Recognition of gestures using the 1¢ gesture recognizer.
- Positioning of the user using Estimote BLE beacons as described in Section 3.1.
- Setup of gesture configurations, *i.e.* a combination of a gesture, a room and an action.
- Recognition of the context using a Bayesian network as described in Section 3.3.
- Presentation of a list of actions if one single action cannot be determined but rather we have a set of actions to choose from.
- Configuration and use of virtual positions, *i.e.* allowing the user to manually determine which room he is in.

BLE Beacon

The Bluetooth Low Energy beacons are used to position the user indoor. We have not implemented any features on the BLE beacons but in order to position the user, the smartwatch reads the RSSI values from the Bluetooth beacons and based on those values determines which room the user is in. This is described in detail in Section 3.3.3.

Raspberry Pi

The Raspberry Pi runs openHAB, which was described in Section 2.5. openHAB is the home automation software utilized in the project. The software receives requests over HTTP. Each request is then translated to an equivalent request using an appropriate protocol, *e.g.* Bluetooth, HTTP or MQTT. The translated request is forwarded to a controllable device. The following features were implemented on openHAB.

- We implemented an openHAB plugin for configuring which beacons are installed in the rooms of the users house or apartment.
- openHAB was configured to support the actions in the scenario presented in Section 1.2, *e.g.* we implemented rules to toggle a lamp between the on and off states as this is not directly supported by openHAB.

Computer

The desktop machine serves as the media centre used for testing while developing the solution. In order to control Spotify, we developed a smart application that runs on the desktop machine. The application receives requests over HTTP and forwards them to the Spotify client for OS X which is an AppleScript scriptable application, meaning that it provides a terminology that scripts can use to send *events* to the application. An event triggers a handler within the application which in

some way modifies the application. For the Spotify application, events includes “next”, “previous” and “playpause”. When the application receives an event, an appropriate action is taken in the application.

The following feature was implemented in the application running on the desktop machine.

- Skip to next track, skip to previous track, play and pause the music in Spotify by issuing HTTP requests to the application.

Philips Hue Bridge

A bridge provided by Philips is installed in the users home. The bridge communicates with the Philips Hue lights using the ZigBee protocol. The protocol is described later in this section. We have not implemented any features on the bridge but use it in order to control the Philips Hue light bulbs.

Philips Hue Light Bulb

The light bulbs provided by Philips are controlled using gestures on the smartwatch, *i.e.* using the CARMA application. The bulbs receive commands from the Philips Hue bridge when they need to change their state, *e.g.* turn on, turn off, change color or change temperature. The bulbs can communicate with each other using the ZigBee Light Link protocol to extend the range of the network by forwarding commands between bulbs.

4.1.2 Known Issues

The following describes known issues in the implementation.

- There is an issue with the implementation of the gesture recognizer that causes the smartwatch application to crash if recognition is started while little to no measurements from the accelerometer are available to the gesture recognizer. In order to fix the issue, recognition should most likely be abandoned all together when little information is available. We have only seen the issue when users have accidentally stopped recognition immediately after starting it.
- Sometimes the application will hang, *i.e.* “freeze”, when context recognition is started. The fix for the issue is most likely to perform the context recognition on another thread.
- In order to present the settings (see Figure 4.6(c)), the user must scroll from right to left on the gesture recognition screen (see Figure 4.6(a)). The list of settings is meant to be scrolled vertically but it seems that the horizontal scroll gesture interferes with the vertical scroll causing the list of settings not to scroll. It is, however, possible to tap in the top and the bottom of the list to change the selected setting.

4.1.3 Missing Implementation

While not part of the requirements presented in Section 1.6, we presented a model for including the state of the system in Section 3.3. The system state was introduced in the model to illustrate how contextual information different than the gesture performed by the user and the users location can be included in the system.

The following features were not implemented.

- Inclusion of the system state in the Bayesian network used for recognizing the context.

- Reuse of the Bayesian network. The Bayesian network is created every time the recognition is started. When the network is created once, it could be reused for future computations of beliefs.

While inclusion of the system state is not implemented, a naive implementation of the system state is trivial to implement. The Android Wear application can periodically request the state of all controllable devices registered in openHAB and based on the response, populate the system state nodes in the Bayesian network with the appropriate states. The nodes will typically have hard evidence on one of the states as openHAB is considered to hold the truth of the systems state, e.g. it knows if a television is on or not and if music is playing or not.

Reuse of the Bayesian network could be implemented by creating the network when the application whenever the network changes, *i.e.* new gesture configurations are created. The network could then be stored either in memory or on disk and loaded. For sufficiently large networks, this may be faster than recreating the network whenever it is needed.

4.1.4 HTTP

As shown in Figure 4.1 we use HTTP for communication between the smartwatch, the Raspberry Pi and the computer acting as media centre. HTTP, or Hypertext Transfer Protocol, is a protocol generally used for exchanging *hypertext*, text which includes hyperlinks. HTTP is a client-server protocol, in which a client sends a request to the server which in turn sends a response. Resources provided by the server are identified by a URL, a Uniform Resource Locator. An example of a URL is “<http://mysite.com/index.html>” which requests the “index.html” file on the “mysite.com” hostname using the “http” protocol. When a web server returns a response, it will return an HTTP status code as part of the response. Examples of this includes the status code 200 for success, 404 for a resource not found and 400 for a bad request. HTTP utilizes TCP/IP to transfer information. TCP transfers information in small packets between machines and IP is responsible for addressing machines in a network and routing the information between the machines.

In our prototype we use a framework appropriate for the platform to issue requests and send a response, *e.g.* in the case of the smartwatch application we use the Android Volley framework for making requests¹. In the case of the Spotify Controls API, we use the Express² framework for building the server. The requests are sent to either openHAB or the Spotify Controls API which will return a response formatted using JSON. In the case of a simple request, *e.g.* for changing the state of the music centre, the JSON response, as well as the HTTP status code, indicates whether or not the request was successful. For a more complex request, *e.g.* when requesting the available items in openHAB, the response will contain all available items in openHAB formatted as JSON along with a HTTP status code.

HTTP defines methods for sending requests to a server. Amongst others are GET, POST and PUT, which are generally used for retrieving content, creating content and updating existing content, respectively. The Spotify Controls API uses the HTTP method PUT for changing the state of the player. Likewise the openHAB API also uses the GET, POST and PUT where appropriate.

4.2 Programming Languages

In order to give a better understanding of the implementation details, we present the programming languages used for the implementation with the reasoning of the language choices.

¹For more information about Google Volley, refer to <https://developer.android.com/training/volley/>

²For more information about Express, refer to <http://expressjs.com>

4.2.1 Android Wear

The primary application presented in this report is an Android Wear application. While Google's Android framework APIs are targeted towards a Java environment, Google does provide Android NDK, a toolset for writing parts of an Android application in the C or C++ programming languages. Google emphasizes that Android NDK is intended only for parts of an application and is generally not suited for most applications. The Android SDK is meant to be used when reusing existing code libraries or frameworks written in C or C++.

Alternative tools for developing to the Android platform includes Xamarin³ and Phonegap⁴ in which developers write applications in either C# or HTML and CSS respectively. The tools are targeted towards cross-platform development, *i.e.* developing for multiple platforms using the same codebase.

For the purpose of the prototype, we have no interest in supporting other platforms than the Motorola 360. Furthermore we are comfortable with the Java programming language. Therefore we chose to write the application in Java.

4.2.2 Raspberry Pi

We developed an addon for the openHAB which runs on the Raspberry Pi. Because openHAB provides a Java based framework for writing addons and we were already writing Java for the Android Wear platform, we decided to write the addon in Java.

4.2.3 Desktop Machine

The small application created for controlling a Spotify client running on a desktop machine using HTTP requests, was developed in JavaScript using the Node.js framework⁵. The client calls a script written in AppleScript.

Because the application does not depend on APIs from other services but merely invokes a local AppleScript, the choice of language and framework was based on our familiarity with the two.

4.3 Component Diagram

Figure 4.2 shows a component diagram of the primary components involved in the application developed for the Android Wear smartwatch. The component diagram illustrates which key components are involved in the software and how they interact with each other. The diagram focus on the smartwatch as most of our development was concentrated on that platform.

Below is a brief description of the components in the diagram. Notice the note in the diagram. Components which communicate without an *assembly connector*, *i.e.* communicating using a required and a provided interface, communicate without the use of Java interfaces, *e.g.* invocation of public methods that are not part of a Java interface. Since the components are still considered to be independent of each other, it makes sense to revisit the implementation to communicate using assembly connectors, *i.e.* Java interfaces. When information is passed without the use of assembly connectors, the edge is annotated with the information being passed between the components.

PositionManager Responsible for determining which room the user is in.

³For more information on Xamarin, refer to <https://www.xamarin.com>.

⁴For more information on Phonegap, refer to <http://phonegap.com>.

⁵For more information on Node.js, refer to <https://nodejs.org>.

PositionContextualInformationProvider Provides the context recognizer with information about the users position. The provider encapsulates the Room and Room_Action nodes of the Bayesian network and computes evidence for all rooms.

GestureContextualInformationProvider Provides the context recognizer with information about the gesture the user performed. The provider encapsulates the Gesture and Gesture_Action nodes of the Bayesian network and computes evidence for all gestures.

ContextRecognizer Determines an appropriate action to trigger based on the information provided by PositionContextualInformationProvider and GestureContextualInformationProvider.

GestureRecognizer Given X, Y and Z-accelerations obtained from the accelerometer of the smartwatch, the component scores the gesture templates and thus determines the gestures the user is likely to have made.

OpenHABClient Communicates with openHAP over HTTP

App UI The component represents the UI of the Android Wear smartwatch application.

For a more detailed description of the components involved in the context engine, refer to Section 4.5.

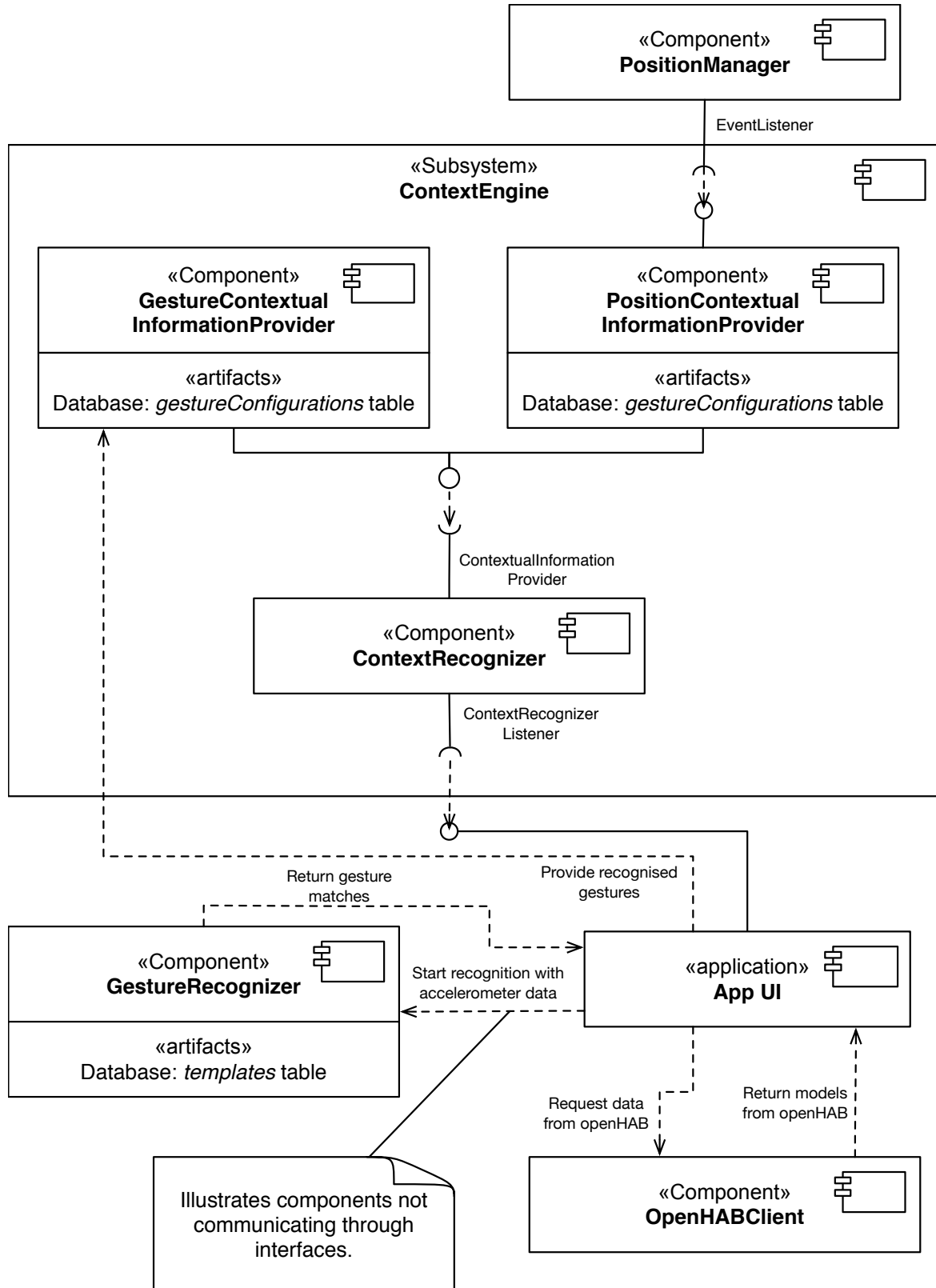


Figure 4.2: Component diagram showing the primary components of the Android Wear application.

4.4 Bayesian Network

In Sections 2.7 and 3.3 we described our motivation to use a Bayesian network for determining the appropriate action to trigger given a context. In order to save time on the implementation of the Bayesian network, we decided investigate third party implementations.

Our requirements for the implementation were the following:

- The implementation should be correct, *i.e.* given a set of nodes with states, edges, probabilities and evidence the computed belief of the nodes should be correct.
- The implementation should support soft evidence as described in Section 3.3.
- The implementation should be written in Java.

We chose to only consider implementations written in Java as we are familiar with the language and we were confident that implementations written in Java would run on Android platforms with only little work required.

The correctness of the implementation was validated using Hugin, a decision making tool previously described in Section 2.8.2. Given a set of nodes with states, edges, probabilities and evidence on the states, the resulting belief of the states should equal the resulting belief when the same configurations were made in Hugin.

When investigating third party solutions, we found that only few implementations supported soft evidence. Since we utilize this in the design of the Bayesian network, the implementation must support it.

We found that the Jayes⁶ and Bruse⁷ implementations fulfilled our requirements. We chose to use the Jayes framework for the following reasons.

- The framework is maintained by Eclipse Foundation, who use it in Code Recommenders, a tool for intelligent code completion⁸. The implementation being used in a large project and maintained by a foundation may be the reason it is more widespread and more information, such as documentation, is available.
- The framework has unit tests which eases the process of getting started using the framework as the unit tests help document how the framework is intended to be used.

4.5 Context Engine

In Sections 3.3 and 3.4, the reason for introducing action nodes at the middle level of the Bayesian network and thus “translating” probabilities of gestures and rooms at the uppermost level to probabilities of actions is described. Modelling the Bayesian network in this way results in a modular design in which contextual information observed by different sensors can be independent. In the following we will briefly describe the realization of this design, which we refer to as the *context engine*. We will also discuss the benefits of the engine.

Figure 4.3 shows a class diagram of all classes and interfaces involved in the context engine. Below is a brief description of the classes and interface.

ContextualInformationProvider Observes a delimited area of the context and provides information about the area to the ContextRecognizer. The provided information is encapsulated in a ProvidedContextualInformation model. Examples of ContextualInformationProviders include the GestureContextualInformationProvider and the

⁶The Jayes implementation is available at <http://www.eclipse.org/recommenders/jayes/>.

⁷The Bruse implementation is available at <https://github.com/slangevi/bruse>.

⁸More information about the Eclipse Code Recommenders is available at <http://www.eclipse.org/recommenders/>.

PositionContextualInformationProvider which provide information about the gesture performed and the room the user is in respectively. Providers encapsulate one or more nodes in the Bayesian network presented in Section 3.3. It encapsulates a node at the middle level in the network and zero or more nodes at the uppermost level. For example, the **GestureContextualInformationProvider** encapsulates the **Gesture** and **Gesture_Action** nodes. The providers are described in greater detail later in this section.

ContextualInformationListener Objects conforming to the interface can get a callback when a **ContextualInformationProvider** either has the necessary contextual information or fails to retrieve it. The **ContextRecognizer** pass these as anonymous classes in Java.

ProvidedContextualInformation Encapsulates information from a **ContextualInformationProvider**. The model contains the node which should be parent to the **Action** node, a node to apply evidence to and the soft evidence to apply to the node. The parent node, which resides at the middle level in the Bayesian network, and the node to apply evidence to can be the same.

ContextRecognizer The recognizer orchestrates the retrieval of contextual information from the providers. Because a provider is not required to provide its contextual information instantly, the recognizer will timeout and thus cancel the provider if it takes too long to deliver the information.

ContextRecognizerListener When recognition completes, the context recognizer informs a listener about the outcome. Objects conforming to the **ContextRecognizerListener** interface can be informed about the outcome.

ContextOutcome A context outcome encapsulates an action that can be triggered and the probability that the action should be triggered. Context outcomes are the result of performing context recognition. The outcomes are parsed to the object conforming to the **ContextRecognizerListener** interface.

Because the Bayesian network is modelled to “translate” probabilities of states on the uppermost level to probabilities of actions in the system at the middle level, we can implement the contextual information providers to be entirely independent of each other and have each provider encapsulate all the information it needs. In order to use the provider, it is added to the context recognizer which needs to know nothing about the contextual information encapsulated but only the probabilities of the states, *i.e.* the **ProvidedContextualInformation**, as defined by the **ContextualInformationProvider** interface.

Figure 4.4 shows a class diagram of the contextual information providers registered with the context recognizer in the prototype developed during the project. Below is a brief description of the classes and interfaces.

Beacon A representation of a beacon installed in a room. Beacons are retrieved from openHAB over HTTP using the REST API.

Room A representation of a room which the user can be in. Rooms are retrieved from openHAB over HTTP using the REST API.

PositionManager The manager continuously listens for changes to the users position using the Estimote SDK. Based on RSSI measurements received by the Estimote SDK, the manager determines which room the user is in.

EventListener Part of the **PositionManager**. Objects conforming to the interface can be informed when the manager registers the user in a room.

GestureContextualInformationProvider Encapsulates the **Gesture** and **Gesture_Action** nodes in the Bayesian network presented in Section 3.3. When gesture recognition ends, the provider is updated with the matches which it stores and use to compute the evidence when asked to provide its contextual information as described in Section 3.3.2.

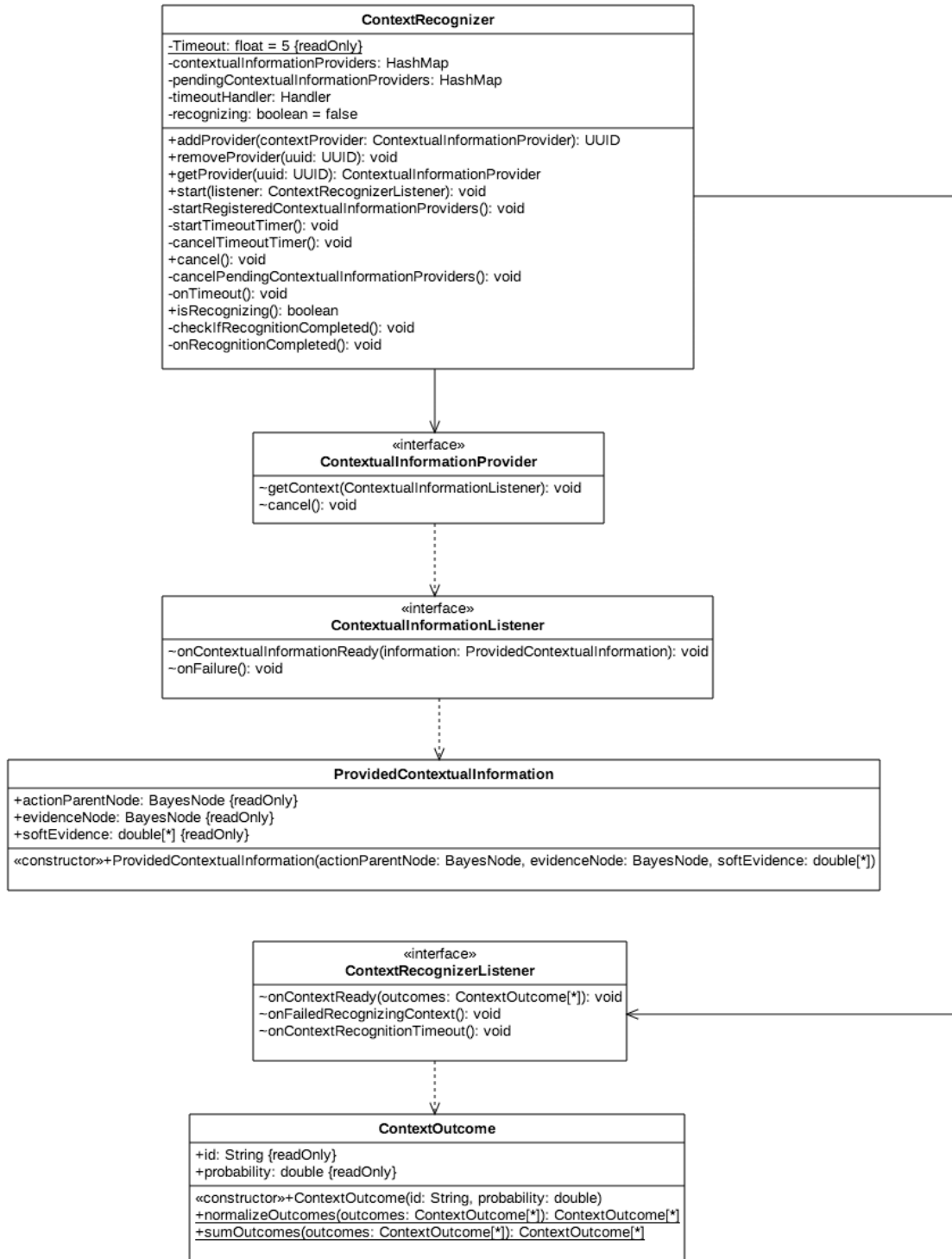


Figure 4.3: Class diagram showing all classes and interfaces involved in the context engine.

PositionContextualInformationProvider Encapsulates the Room and Room_Action nodes in the Bayesian network. Based on the observations made by the `PositionManager`, the provider

calculates probabilities for the user being in each room as described in Section 3.3.3.

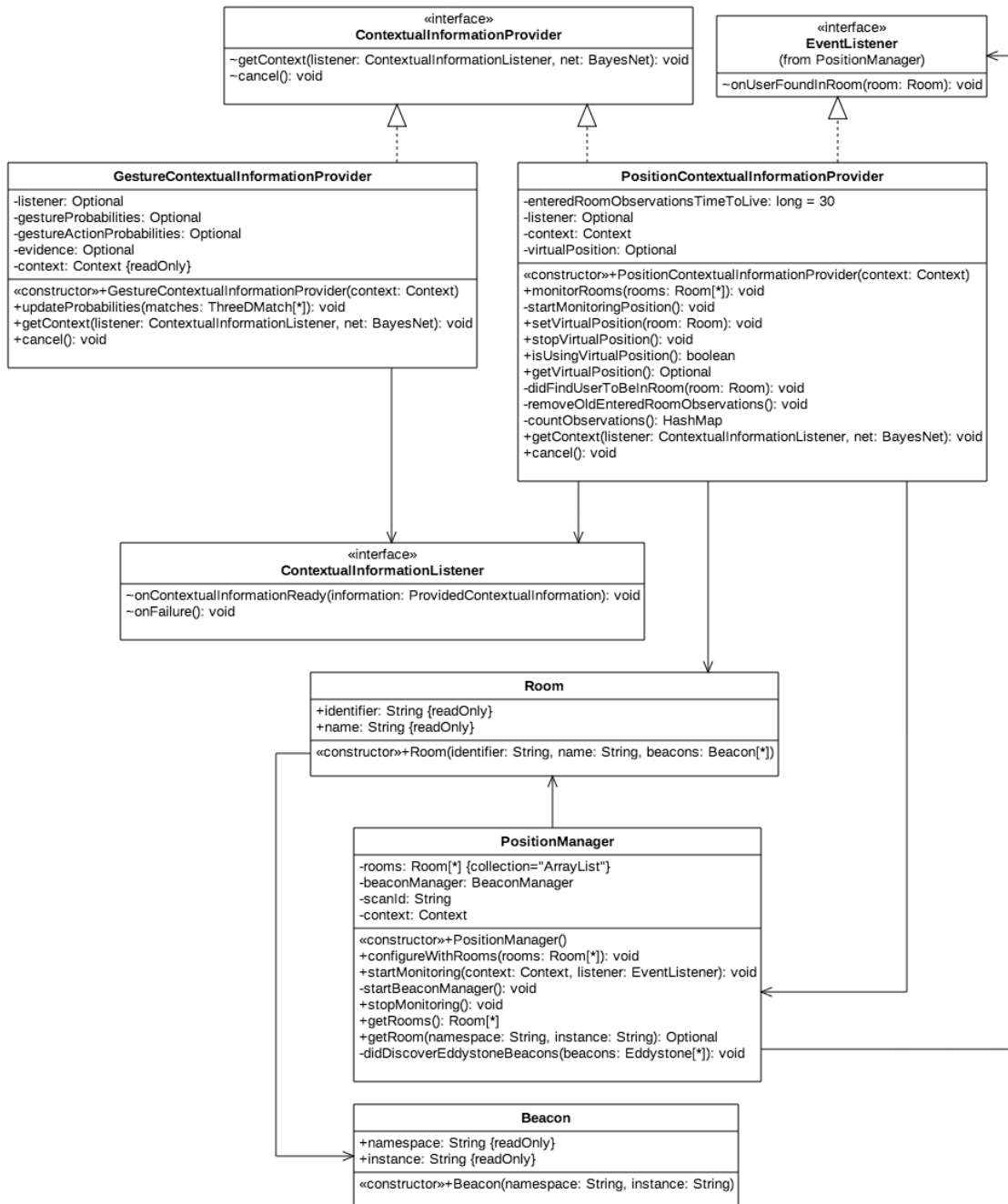


Figure 4.4: Class diagram showing the classes and interfaces involved in the providers.

4.6 Communication with openHAB

For a detailed description on the implementation used for communication with openHABs API, refer to Appendix B. When communicating with openHAB, we must send actions supported by the

Table 4.1: Supported actions for each item type in openHAB [56].

Item type	Description	Action types
Color	Color information (RGB)	ON, OFF, INCREASE, DECREASE, Percent, HSB
Contact	Item storing status of e.g. door/window contacts	OPEN, CLOSE
DateTime	Stores date and time	
Dimmer	Item carrying a percentage value for dimmers	ON, OFF, INCREASE, DECREASE, Percent
Group	Item to nest other items / collect them in groups	
Number	Stores values in number format	Decimal
Player	Allows to control players (e.g. audio players)	PLAY, PAUSE, NEXT, PREVIOUS, REWIND, FASTFORWARD
Rollershutter	Typically used for blinds	UP, DOWN, STOP, MOVE, Percent
String	Stores texts	String
Switch	Typically used for lights (on/off)	ON, OFF

Table 4.2: Actions supported by the smartwatch application.

Item type	Action types
Dimmer	ON, OFF, INCREASE, DECREASE, 10, 20, 30, 40, 50, 60, 70, 80, 90
Switch	ON, OFF

items in openHAB.

Eclipse SmartHome, which openHAB is built on top of, has a set of supported item types. An item type defines which actions an item supports. Table 4.1 shows a list of the item types supported by Eclipse SmartHome and thus also by openHAB. For each item type, the actions supported by the item type is listed. Item types DateTime and Group are special items in openHAB which has no support for actions but is used to hold a date and time and a group of other items respectively. The action types “Percent”, “HSB”, “Decimal” and “String” are not exact actions that can be send to openHAB but refers to type of action, e.g. a lamp supporting the “Percent” action can receive a request containing “50” and adjust its brightness to 50%. The other action types are exact actions that can be send in a request to openHAB.

In the smartwatch application, we create a list of supported actions for each item. A supported action for an item is represented as the string, which is sent to openHABs API when an action should be triggered. For example, an item of type Switch has the set of actions {“ON”, “OFF”}. For the sake of this prototype, we only support a limited subset of item types and actions. The supported types and actions are shown in Table 4.1.

4.7 Prototype

This section presents the prototype of the Android Wear application as well as the *binding*, i.e. an addon providing functionality specific to a problem domain, for openHAB developed during the

project.

4.7.1 Android Wear

Figure 4.5 shows a navigation diagram for the user interface of the smartwatch application. The letters in the top right corner of the nodes in the diagram references the screenshots shown in Figure 4.6. The diagram shows that the application can be considered divided into the following two areas.

Primary use We refer to the parts of the application which the user uses the most as the *primary use*. This involves the recognition of gestures as well as the picker presented when an action to trigger could not be determined after context recognition.

Configuration The user will need to configure the application, *i.e.* train gestures and create gesture configurations. We refer to this part of the application as *configuration*.

When opening the application, the user is presented with a screen to perform gesture recognition. This enables the user to start context recognition fast, whereas he needs to dive deeper into the navigation in order to train gestures and create gesture configurations, *i.e.* features that he will not use as often as the context recognition. For a list of the supported features in the application, refer to Section 4.1.

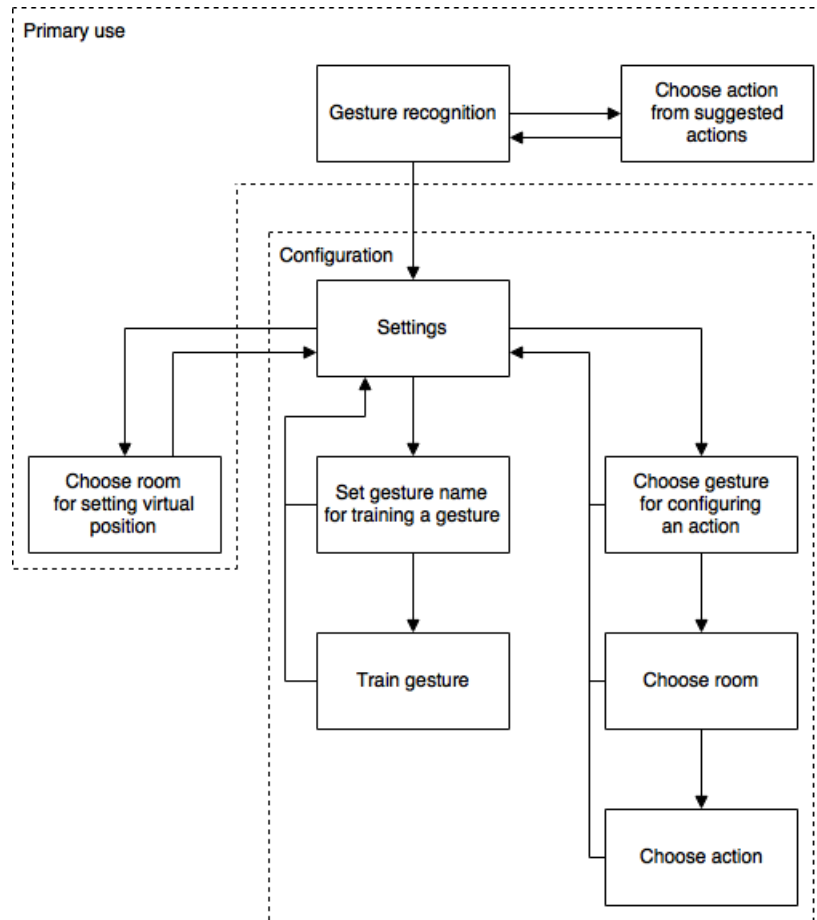
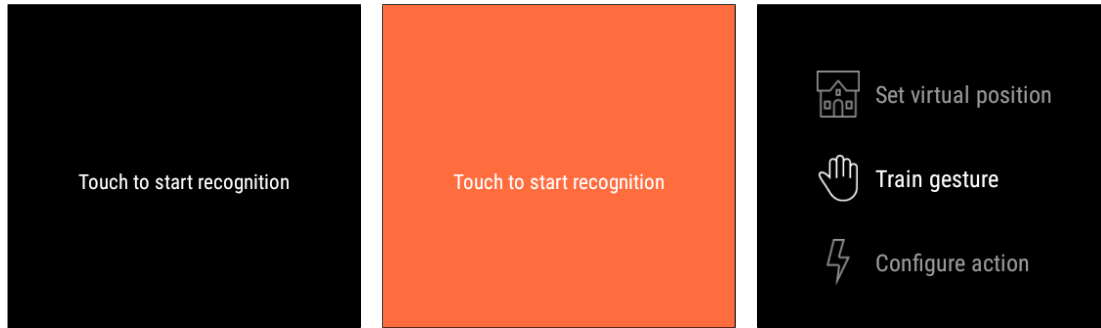
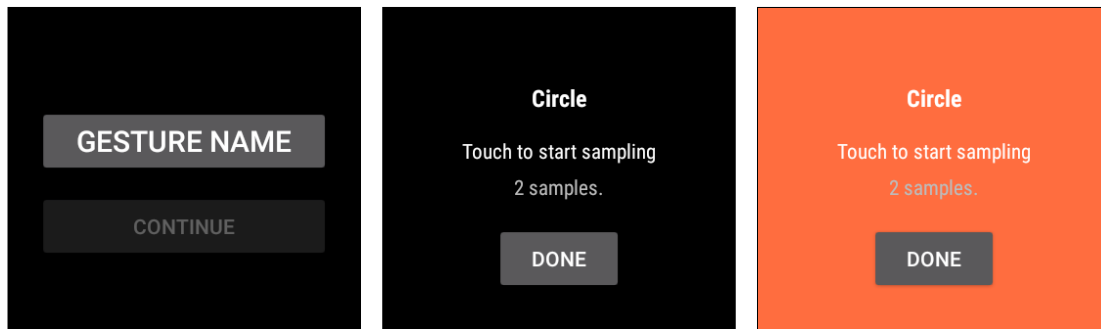


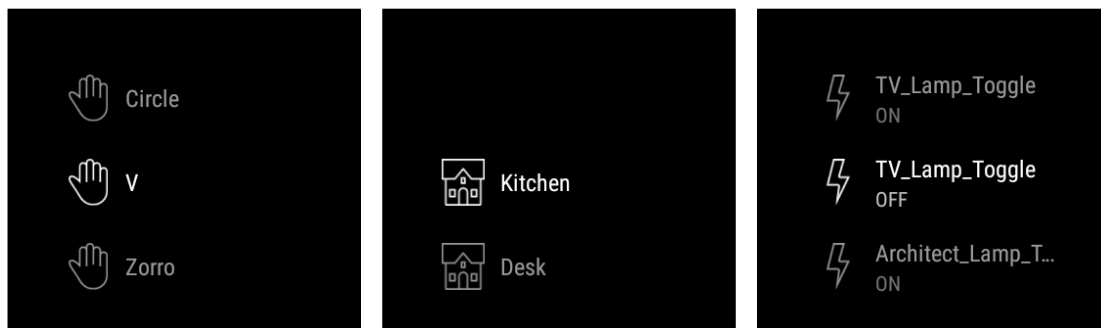
Figure 4.5: Navigation diagram for the user interface of the Android Wear application. Letters in the top right corner of each node refers to the screenshots in figure Figure 4.6.



- (a) Screen on which the user can start gesture recognition by touching the screen.
- (b) Screen on which the user can stop gesture recognition by touching the screen. The orange color indicates that recognition has been started.
- (c) Settings screen from which the user can set his virtual position, train a gesture and create a new gesture configuration. When a virtual position is set, the setting can be selected again to unset the virtual position.



- (d) Screen shown when the user starts training a gesture. Tapping the "Gesture name" button will present the user with a speech recognition interface from which he speaks the name of the gesture to configure it.
- (e) Screen shown when training a gesture and the gesture recognizer has not been started. We see that two gesture samples, i.e. gesture templates, have been trained.
- (f) Screen shown when training a gesture and the gesture recognizer has been started, i.e. the user is performing a gesture.



- (g) Gesture picker showing the gestures the user has trained. The gesture picker is used when creating a gesture configuration.
- (h) Room picker shown when the user creates a gesture configuration. A similar picker is also shown when setting a virtual position.
- (i) Action picker shown when creating a gesture configuration. A similar picker is shown when the context engine suggests multiple actions.

Figure 4.6: Screenshots of the prototype developed for the Android Wear smartwatch.

4.7.2 openHAB

We developed a *binding* for openHAB. A *binding* is an addon for openHAB which relates to a specific problem domain. For example, there is a binding for communicating with Philips Hue lights and a binding for controlling a Sonos media centre. We have created a binding for configuring the rooms and beacons in a users home.

We chose to place the configuration of the rooms and beacons in openHAB, as this is a one time configuration which can be shared amongst the users in a home where as the creation of gesture configurations is placed on the watch as this can be independent for each inhabitant of a home. The rooms and beacons are synchronized to the smartwatch when the application is loaded.

A room has a name and when it is created, openHAB assigns it a UID. When creating a beacon, the user must enter the UID assigned by openHAB in order to specify which room the beacon is placed in. Furthermore the user should enter the Eddystone namespace and instance, both described in Section 3.1. This identifies the specific beacon. When the smartwatch registers a beacon, it can determine which room the user is in by the beacons reference to a room.

In order to control Spotify running on a desktop machine, we created the Spotify Controls API. We configured openHAB to communicate with the API using the default Exec binding, a binding which executes a shell command when some event occurs in openHAB.

4.8 Version Control

We use version control management for keeping track of changes to the codebase developed through the project as well as this report. Below is a list of advantages of using version control management.

- Collaborators can work on the same file at the same time, due to the way changes to the file can be *merged*. This is in contrary to a shared folder, e.g. a Dropbox folder which always synchronize the most recent version of a file with a central location.
- Keeping track of what files were changed, what was changed in the files and who changed it.
- When publishing changes to the files, authors typically tag the changes with a message with a description of the changes making their intention clear to collaborators.
- Changes can be rolled back to a previous state.
- Collaborators of a project can *branch* out from the main codebase to create changes without touching the currently stable code base. When their changes are done, they can *merge* in their changes to the stable codebase.
- Depending on the amount of collaborators and the system used, the codebase is inherently backed up.

There are several software solutions for version control management, including Git, Subversion, CVS and Bazaar. When choosing a system to use, we decided to only look into the Git and Subversion as we have experience with the two.

The key difference between Git and Subversion is, that Git is decentralized and Subversion is centralized. When using Git, collaborators have a local copy of the entire repository in which the codebase resides. Collaborators then push their changes to a central location when they are done working on a feature or a fix. With Subversion, collaborators are working in a central online repository, meaning that the version control features are unavailable when there is no connection to the repository.

We chose Git over Subversion because of it being decentralized. This provides two advantages over Subversion.

- When we are working with no or an unstable internet connection, version control features are still available. Several times during the project we have worked with no internet connection.
- We believe that a decentralized and local system provides extra safety in terms of backups. All collaborators always have a backup of the repository on their local machines. In addition, we push the changes to GitHub, a website for hosting Git repositories, from which we also pull the changes other collaborators make.

Evaluation

This chapter describes the test we setup in order to evaluate our solution and presents modifications that can be made to the solution in order to improve the results.

5.1 User Test

A user test was performed in order to investigate the accuracy of the system given data from different users. The test is presented in the following section.

5.1.1 Goal

The goal of the test was to see how well the system performed when used by other people as well as to determine if it achieved the requirement of triggering the correct actions 80% of the time as specified in Section 1.6.

5.1.2 Setup

The setup of our user test consisted of a Macbook Pro running OpenHAB and Spotify, two Philips Hue lamps and two Estimote beacons. Seven people were asked to train four unique gestures, creating ten templates per gesture resulting in a total of 40 gesture templates per person. The gestures used were:

- Circle
- Swipe Left to Right
- V
- Zorro

The people who participated in the test were fellow masters students and as such were adept at using modern technology, however none of them had any prior experience with our project.

The setup was limited to a subset of the smart devices and gestures presented in the scenario in Section 1.2, because we did not have the necessary hardware and space available to perform a test of the entire scenario as well as to keep the invested time for each participant to a minimum.

The gesture templates stored in the database were removed before each test so each participant was only using his own templates and not those created by others.

The participants were instructed how to perform the gestures but were allowed to scale them to their personal preference, *e.g.* either create large circles or small circles.



Figure 5.1: Image of the setup used for the user test.

5.1.3 Method

Once the gesture templates had been created the participants were asked to complete seven tasks, each of which required them to perform a given gesture five times. The test took place in a single room but to simulate the participant moving between two different rooms, only one Estimote beacon would be turned on at a time. A list of the gestures and locations is shown in Table 5.1, along with the OpenHAB actions they were supposed to trigger.

Action	Gesture	Location
Shelves_Lamp_Toggle	V	Home Office
Spotify_PlayPause	Circle	Home Office
Spotify_Next	Swipe Left to Right	Home Office
Architect_Lamp_Toggle	V	Living Room
TV_Lamp_Toggle	Zorro	Living Room
Spotify_Next	Swipe Left to Right	Home Office (Virtual Position)
Shelves_Lamp_Toggle	V	Home Office

Table 5.1: The actions, gestures and locations that were used during the user test.

For each participant and task, the success rate of the actions triggered was calculated as the *number of times the intended action was triggered* divided by the *number of attempts* and can be found in Figure C.9. Figures C.10 and C.11 also shows the success rate of gestures and locations.

5.1.4 Results

The average success rate of actions is below the 80% requirement specified in Section 1.6 with a combined average of 44%. Triggering the correct action less than half of the time is not a satisfactory result and is thus something that needs to be looked into. The locations are correctly identified in the majority of the cases with an average success rate of 83% which indicates that the positioning is reliable.

One source of the inaccuracy comes from the low average success rates of the gestures. Figure C.8 shows that the gesture success rates are generally higher than those of actions with an average of 56%. However differences between the average gesture success rates of the participants

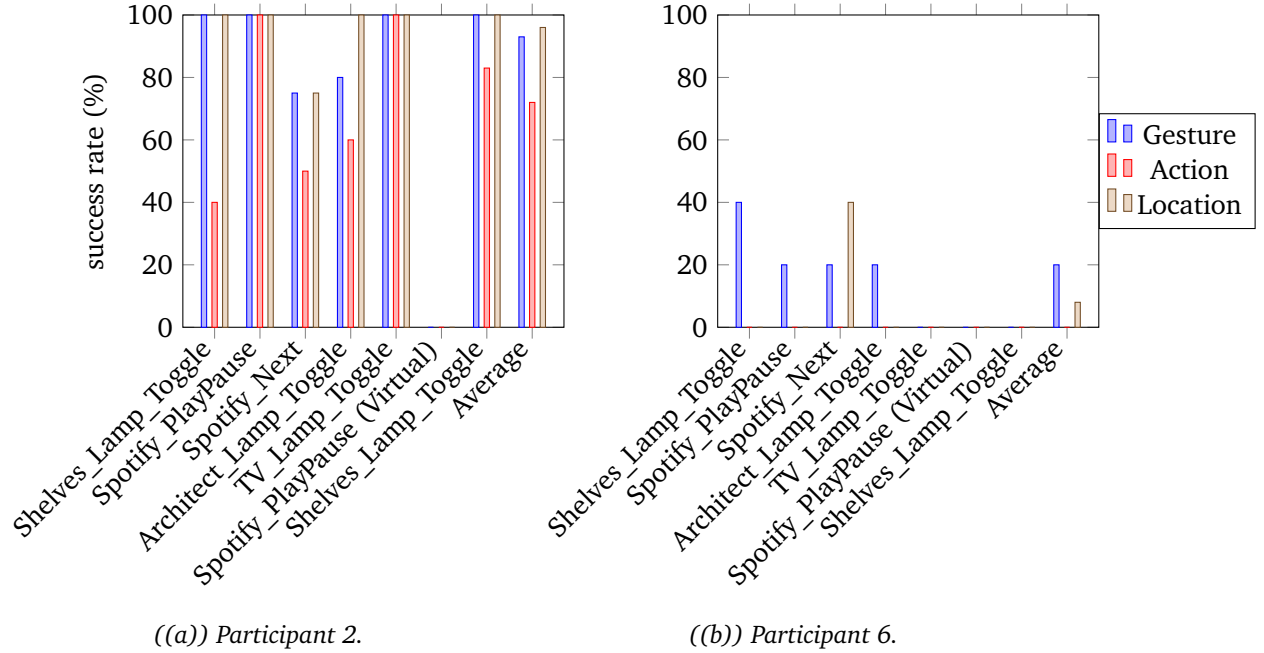


Figure 5.2: Average success rate for gestures, actions and locations for the participants who performed the best and worst respectively.

are noticeable. The most noticeable example is comparing Figures 5.2(a) and 5.2(b). Something to note here is that both of these participants, as well as participant 1 experienced some technical difficulties which prevented them from completing the *Spotify_PlayPause (Virtual)* task. Participant 2 has an average success rate for gestures of 93% and participant 6 has an average success rate of 20%. While our lowest success rate is below the one reported in the paper about the \$3 gesture recognizer, our results are not much different from the ones presented in the paper with \$3 having a success rate between 58% and 98% [40]. This is a significant difference and indicates that possibly something went wrong during the tests of participant 6, or the gesture recognizer is not capable of handling people with different capabilities of performing these gestures.

Another source of the inaccuracy of actions is the way we have modeled the context engine. Looking at the success rates for *Shelves_Lamp_Toggle* in Figure 5.2(a) shows that the correct gesture and location was always identified, yet the correct action was only triggered 40% of the time.

In a single trial of participant 2 the scores in Table 5.2 were recorded. The gesture *V* correctly received the highest belief in the *Gesture* node but because *V* is configured to be used with two actions, *Shelves_Lamp_Toggle* as well as *Architect_Lamp_Toggle*, the *Gesture_Action* node divides the beliefs amongst these. The gesture *Circle* did not match as well as *V* and only received approximately half of the belief value of *V*. However, since *Circle* is only configured to trigger a single action, its belief value remains intact in the *Gesture_Action* node. This results in the beliefs of *Spotify_PlayPause* and *Shelves_Lamp_Toggle* being approximately equivalent even after the location beliefs have been applied.

As such it is easier to trigger actions that only have a single gesture associated with them than actions that have multiple.

5.1.5 Threats

The following are threats, that could potentially have impacted the results from the user test.

Intended Action	Shelves_Lamp_toggle
Intended Gesture	V
Intended Location	Home_Office
Gesture	Belief
Circle	33.18003597
Swipe Left to Right	0
V	66.81996403
Zorro	0
Gesture_Action	
Spotify_PlayPause	33.18003597
Spotify_Next	0
Shelves_Lamp_Toggle	33.40998201
Architect_Lamp_Toggle	33.40998201
TV_Lamp_Toggle	0
Room	
Living Room	0
Home Office	100
Room_Action	
Spotify_PlayPause	33.33333333
Spotify_Next	33.33333333
Shelves_Lamp_Toggle	33.33333333
Architect_Lamp_Toggle	0
TV_Lamp_Toggle	0
Action	
Spotify_PlayPause	33.25668465
Spotify_Next	16.66666667
Shelves_Lamp_Toggle	33.37165767
Architect_Lamp_Toggle	16.70499101
TV_Lamp_Toggle	0

Table 5.2: Belief values for the different nodes in the bayesian network of the context engine for a single trial of Participant 2.

- The system was only tested with users who had never tried to use the system before. Users who have more experience with the system would likely perform gestures with a higher accuracy, thus potentially triggering the correct actions more often.
- The chosen gestures might be too similar and thus the gesture recognizer might be unable to precisely determine which gesture the user performed.
- As part of the gesture recognition, a gesture trace is resampled in order to consist of the same amount of points as the templates. We use a resampling rate of 64 to be consistent

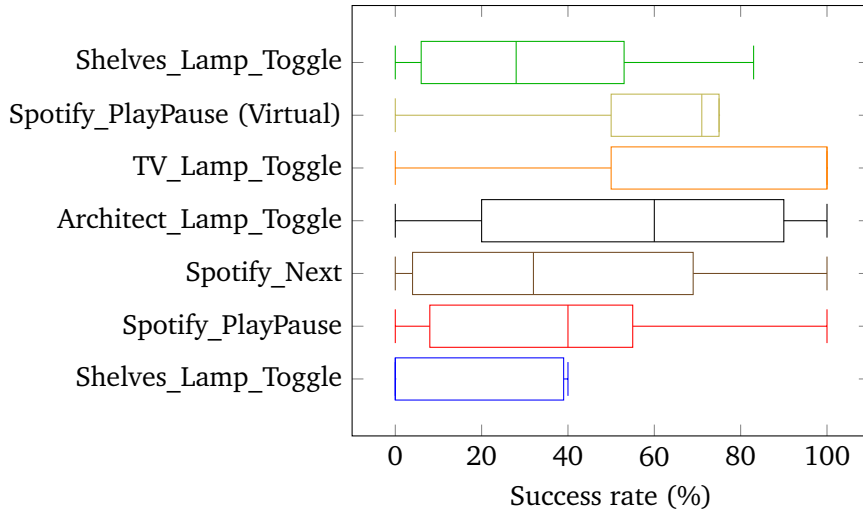


Figure 5.3: Action success rate across all participants.

with 1¢ [32]. \$3 uses a resampling rate of 150 [40]. As our solution is a combination of both 1¢ and \$3, it could be that a higher resampling rate result in a higher accuracy.

5.1.6 Conclusion

From the results of the user test we can see that the average success rate of actions falls below the 80% stated in Section 1.6 with a value of 44%. The two primary causes of this is likely the low average success rate of gestures and the way the context engine is modeled.

The gesture recognition performs sufficiently well for some participants, like participant 2, but the success rate varies too much between different participants. The success rate of the gestures could perhaps be improved if the participants had been asked to practice the gestures before training and using them. It may also work better if the users were allowed to make up their own gestures instead of using the ones selected by us.

The success rate of the actions could be improved by using a different model as the current model favors actions that have only a single gesture associated with it, as seen the example shown in Table 5.2. An alternative model could be one where the *Gesture* and *Room* nodes were direct parents of the *Action* node and removing the *Gesture_Action* and *Room_Action* nodes.

From the user test we found the accuracy of the system to be unsatisfying. The poor accuracy is likely to be due to the model used for context recognition, *i.e.* the Bayesian network and the design of it. If more comprehensive tests in Hugin were done before implementing the system, we might have been able to discover the issue earlier.

One approach for discovering the issue earlier could have involved some user testing as we could have validated that the gesture recognizer worked, recorded gesture traces from multiple users and using the data, design the and test the system.

5.2 Alternative Models

As the user test described in Section 5.1 showed that the system had an accuracy of 44%. This section describes some of the alternative models that were considered and whether or not they addressed the issues raised in Section 5.1.6.

5.2.1 Bayesian Network With a Single Action Node

The Bayesian network used on the context engine consists of an Action node and its parent nodes are Gesture_Action and Room_Action. These two parent nodes were intended to make the network more modular to support additional context information later but in this model the probability of each action is calculated multiple times based on different prior probabilities. Hence we considered an alternate model in which the Gesture and Room nodes were direct parents of the Action node.

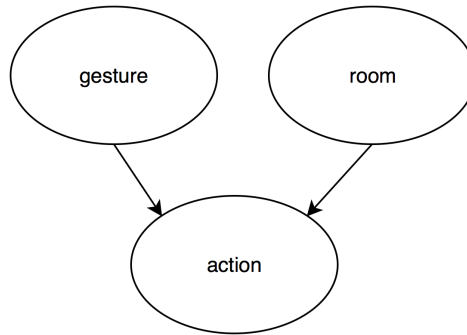


Figure 5.4: Bayesian network without the intermediate gesture_action and room_action nodes.

This alternate network was tested with the belief values of the Gesture and Room nodes of participant 7 and a new success rate of 36% was computed. Compared to the previous success rate of 41% for the participant, this model did not prove to be an improvement over the existing one and was discarded.

5.2.2 Influence Diagram

Influence diagrams can be regarded as Bayesian networks extended with decision variables and utility functions [38]. Figure 5.5 shows an influence diagram modelling a context engine. The model does not correspond exactly with the model previously presented in this report but is an alternative model.

As for the graphical representation of an influence diagram, the square nodes represent *decision nodes*, i.e. something we must decide to do or not to do. Diamond nodes represent *utility nodes* describing a *utility function* and oval nodes represent *uncertainty nodes*. In an influence diagram we are generally interested in taking the decision that results in the highest utility, therefore utility can be regarded a measurement of the quality of a decision.

In the model presented in Figure 5.5, the utility is a function of the gesture, room and action. We assign a high utility to combinations of gesture and room that are part of a gesture configuration.

Influence diagrams provide a natural way of including the system state, as actions which it does not make sense to trigger given the current state of the system, can be assigned a very low utility.

Due to inference in the probabilistic network, actions are assigned a utility when soft or hard evidence on the gesture and room nodes is available. The utility of an action is shown with dark green bars below the name of the action in Figure 5.6. For example, the utility of the *Shelves_Lamp_Toggle* is 6682, making it the action with the highest utility and thus the action that should be triggered¹.

¹More information on influence diagrams in Hugin is available at <http://www.hugin.com/technology/getting-started/ids>

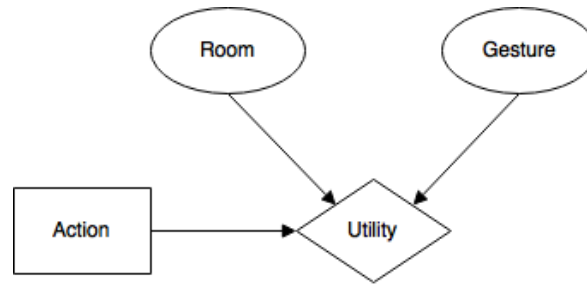


Figure 5.5: Example model of the context engine using an influence diagram.

The scenario discussed in Section 5.1 in which the belief values of a gesture was divided when the gesture was associated with multiple different actions is addressed in Figure 5.6 where the correct action has a greater utility and can be triggered with greater confidence.

When testing the Bayesian network with a subset of the recorded data from the user test, we found that when using the influence diagram, we were able to trigger the correct action more often. During the user test a correct action was triggered 36% of the time for participant 7. Using the exact same beliefs on gesture and room nodes in the influence diagram, the correct action is triggered 50% of the time.

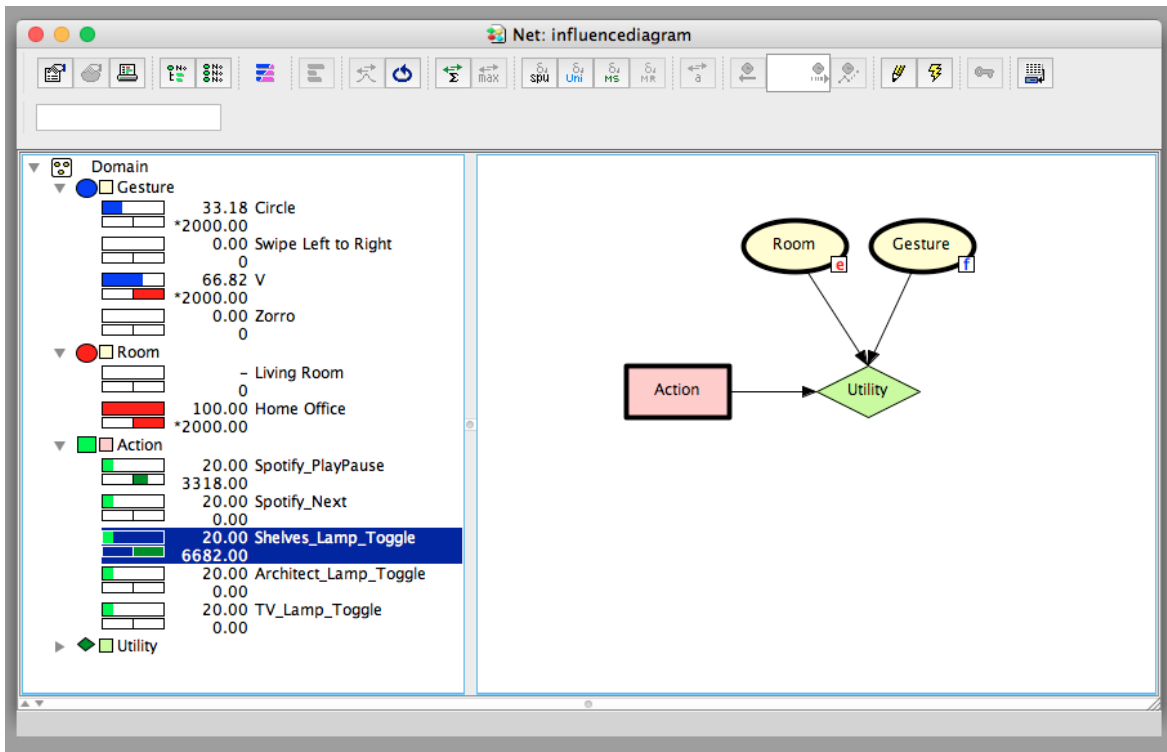


Figure 5.6: Screenshot of example influence diagram in Hugin. The screenshot shows that the influence diagram solves the problem of a single gesture being bound to multiple actions, can result in an incorrect action being triggered because the belief is reduced as shown in Table 5.2 and discussed in Section 5.1.

Table 5.3: Example beliefs of the Gesture node for participant 7 when attempting to trigger an action. The beliefs are shown before the computations were improved and after.

Action	Belief before	Belief after
Spotify_PlayPause	28.39	42.24
Spotify_Next	30.06	16.67
Shelves_Lamp_Toggle	22.87	28.88
Architect_Lamp_Toggle	6.20	12.21
TV_Lamp_Toggle	12.48	0

Table 5.4: Average scores of gestures used when computing the beliefs for the “Belief after” column shown in Table 5.3.

Gesture	Score
Circle	89.80
Swipe Left to Right	116.03
V	94.03
Zorro	110.54

5.2.3 Calculation of Gesture Beliefs

In Section 3.3.2 the computation of beliefs on the Gesture node in the Bayesian network is specified. We only consider gesture templates with a score of 70 or below and then compute the average score. We found that this approach may cause issues, because a single gesture template may have a very low score but all other gesture templates with the same name have very high scores, thus the single gesture template is an outlier potentially causing us to consider an incorrect gesture recognized.

Instead we propose computing the average before filtering the gesture templates. Naturally, the average of each gesture will be larger as we include templates with a higher score in the computation and as such the threshold for accepted gesture should be higher.

Using the scores of the gestures performed by participant 7 in the user test, we computed how many actions would be correctly triggered when computing the average of the scores first and then filter them based on a threshold. We chose a threshold of 100. We found that 45% of the actions were correctly triggered when we adjusted the computations of the gesture beliefs. This is in contrast to 41% before the adjustments were made. While the adjustments are not a big impact, this can be considered an indication that the computations of beliefs in the Gesture node should have been performed differently.

We also found that when changing the way we compute beliefs for the Gesture node, we are generally able to put a greater belief on the correct action. This is exemplified in Table 5.3. The beliefs are taken from a scenario where the user desires to trigger the *Spotify_PlayPause* action. Note that in the beliefs before the improvements to the computations, *Spotify_Next* has the greatest belief where as after the improvements are made, the *Spotify_PlayPause* has the highest belief. The average score used when computing the improved beliefs are shown in Table 5.4. Note that only the Circle and V gestures have an average score below 100 and thus only those two gestures are assigned a belief greater than zero. Before improving the computations, all four gestures were assigned a belief greater than zero.

Conclusion

This chapter concludes on the solution described in this report, and implemented in the project.

6.1 Project Results

In this section, we will conclude on the results from investigating the problem statement presented in Section 1.4:

How can we design and implement a system that utilizes contextual information for controlling a smart home using a wearable in a gesture driven solution?

We have presented an approach for recognizing gestures and once a gesture is recognized beginning to recognize the context and trigger an appropriate action. The system presented has been designed and implemented on an Android Wear smartwatch and a Raspberry Pi, allowing users to control a music centre and smart bulbs.

We intended to model a generic engine for recognizing context based on various sources for contextual information. Our concrete implementation of the system utilizes BLE for positioning the user and a gesture recognizer derived from 1¢ [32] and \$3 [40]. In practice the context engine suggests different actions for the same gesture depending on the position of the user in his smart home.

As described in Section 5.1, the system triggers the correct action 44% of the time. According to the requirement specification presented in Section 1.6, an accuracy of at least 80% was desired and as such, the system is not satisfyingly accurate.

We presented a concrete design and implementation for a system utilizing contextual information to control a smart home. As the accuracy of the system is unsatisfying, better designs of such a system may exist. We have investigated an alternative design of the Bayesian network and found that it was not more accurate. We also presented an influence diagram to potentially replace the Bayesian network. The diagram proved to perform better when tested with data from a single participant. Further work on this project should investigate the possibility of using influence diagrams as the model for the context engine.

6.2 Future Work

Based on the evaluation presented in Chapter 5 and the project results presented in Section 6.1, we believe that a context-aware smart home controlled using gestures is feasible in the future. In this section, we present functionality and design changes that would make sense to investigate, in case future work is to be done on the project.

6.2.1 Continuous Recognition of Gestures

Investigation of battery efficient approaches for continuous gesture recognition could be beneficial. The current implementation requires the user to open the Android Wear application, tap to start recognizing a gesture, perform the gesture and then tap again to stop the gesture recognition. From the tests we conducted with users, we found the approach to be unsuitable as starting the recognition must be done with the arm on which the watch is mounted, held in a position ready to do the gesture and stopping the gesture must be done in the position where the gesture ended.

In order to avoid this, we imagine a solution where the smartwatch continuously attempts to recognize gestures based on accelerometer data, even when the smartwatch application is in the background. Such a solution would solve the following two issues.

The need to open the application to perform recognition When gesture recognition can be done with the application put into the background, *i.e.* not launched and visible on the screen of the smartwatch, there is no longer a need to open the application thus making it faster to control the smart home using gestures.

Starting and stopping the recognition The user no longer has to manually start and stop the gesture recognition after performing each gesture. The continuous gesture recognizer should automatically detect when the user starts performing a gesture and when he stops performing it.

Research has been conducted in continuous gesture recognition, or *real time gesture recognition* as it is referred to in [19]. The authors have implemented a solution for continuous recognition using a Kinect infrared camera to detect motions and claim that it works with other sensor data as well, *e.g.* accelerations from an accelerometer [19]. The article does not state an accuracy of the recognition.

6.2.2 Inclusion of System State

In the design of the Bayesian network used for context recognition presented in Section 3.3, we introduced the state of the system as contextual information and as explained in Section 4.1, the functionality was not implemented. The point of introducing the state of the system, is to achieve lower belief values of actions that it does not make sense to trigger given the current state of the system. For example, an action for changing channel on the television would have a lower belief value when the television is turned off, than when it is turned on.

When excluding the system state, the context engine may suggest such actions and since it is fair to assume that the user knows at least part of the system state, the suggested action is likely not the intended one. By including the system state, we could potentially reduce the risk of suggesting unintended actions.

6.2.3 Inclusion of User History

Looking further into inclusion of concepts from machine intelligence could present interesting possibilities for the project. Using machine intelligence, attempts to suggest a user actions based

on his historical behaviour could be made. For example, if the system detects that the user is often in a specific room at a specific time of the day, a stronger belief could be applied to that room at that time of the day. It is also a possibility to detect actions that the user is likely to trigger at certain times of the day. If the user often turns on specific lights when he gets home from work at 17:00, a stronger belief could be applied to the actions turning those lights on.

Short-term historical data, *i.e.* data of the users behaviour collected minutes or hours ago, could be used to further determine the context that the user is in. If he interacted with his television within that past five minutes, the belief of actions related to the television may be enhanced. This is based on the assumption that if the user recently interacted with his television, he may want to interact with it again. Further research into the behaviour of users in a smart home and their interactions with electronic devices could help confirm or dismiss this hypothesis.

6.2.4 Improve Handling of Uncertainties

In Section 1.2.1, we presented suggestions for handling uncertainties in the system, *e.g.* when a performed gesture was not recognized. In that case, we suggest presenting a list of actions for smart devices the user has recently interacted with. The list of actions could be prioritized, *e.g.* ordered by how often the user triggers the action.

This functionality was not implemented in the project. Further research can investigate how we can improve handling of uncertainties.

6.2.5 Investigate Alternative Designs for Context Recognition

We have found the Bayesian network designed and implemented in this project to produce inaccurate results, especially when a gesture is associated with multiple actions, and as such future research should focus on investigating alternative designs for the context engine.

In Section 5.2 we proposed using an influence diagram rather than the Bayesian network. Further research can focus on determining whether modeling the context engine using an influence diagram would result in an accuracy exceeding 44%. Furthermore future research could focus on determining if alternative models of the Bayesian network would produce an improved accuracy, as suggested in Section 5.2.

Bibliography

- [1] Homeport. <https://github.com/home-port/HomePort>.
- [2] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.
- [3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.
- [4] *Proximity Beacon Specification*. Apple, 9 2015. Release R1.
- [5] SIG Bluetooth. Architecture & terminology overview. In *Bluetooth SIG: San Jose, CA, USA Bluetooth* [6].
- [6] SIG Bluetooth. The bluetooth core specification, v4. 0. *Bluetooth SIG: San Jose, CA, USA*, 2010.
- [7] Cristiana Bolchini, Carlo A Curino, Elisa Quintarelli, Fabio A Schreiber, and Letizia Tanca. A data-oriented survey of context models. *ACM Sigmod Record*, 36(4):19–26, 2007.
- [8] J. Bronsted, PP Madsen, A. Skou, and R. Torbensen. The homeport system. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1–5, Jan 2010. doi: 10.1109/CCNC.2010.5421606.
- [9] Maurizio Caon, Yong Yue, Julien Tscherrig, Elena Mugellini, and O Abou Khaled. Context-aware 3d gesture interaction based on multiple kinects. In *Proceedings of The First International Conference on Ambient Computing, Applications, Services and Technologies, AMBIENT*, 2011.
- [10] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR workshop on recommender systems*, volume 60. Citeseer, 1999.
- [11] Diane J Cook and Sajal K Das. How smart are our environments? an updated look at the state of the art. *Pervasive and mobile computing*, 3(2):53–73, 2007.
- [12] Estimote. What is beacon monitoring?, . <http://developer.estimote.com/ibeacon/tutorial/part-2-background-monitoring/>.
- [13] Estimote. What is Eddystone?, . <http://developer.estimote.com/eddytone/>.

- [14] Estimote. What is iBeacon?, . <http://developer.estimote.com/ibeacon/>.
- [15] Ramsey Faragher and R Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In *Proceedings of the 27th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+’14)*, 2014.
- [16] Paulo Ferreira and Pedro Alves. *Distributed context-aware systems*. Springer, 2014.
- [17] Jacob Fraden. *Handbook of modern sensors: physics, designs, and applications; 5th ed*. Springer, Cham, 2016. URL <http://cds.cern.ch/record/2112745>.
- [18] Craig Gilchrist. *Learning iBeacon*. Packt Publishing Ltd, 2014.
- [19] Nicholas Gillian and Joseph A Paradiso. The gesture recognition toolkit. *The Journal of Machine Learning Research*, 15(1):3483–3487, 2014.
- [20] Google. Bluetooth | android developers, . <http://developer.android.com/guide/topics/connectivity/bluetooth.html>.
- [21] Google. Broadcastreceiver | android developers, . <http://developer.android.com/reference/android/content/BroadcastReceiver.html>.
- [22] Google. Creating wearable apps | android developers, . <https://developer.android.com/training/wearables/apps/index.html>.
- [23] Google. Motion sensors, . http://developer.android.com/guide/topics/sensors/sensors_motion.html.
- [24] Google. Sensors overview, . http://developer.android.com/guide/topics/sensors/sensors_overview.html.
- [25] Google. Eddystone protocol specification, . <https://github.com/google/eddystone/blob/master/protocol-specification.md>.
- [26] Google. Eddystone-tlm, . <https://github.com/google/eddystone/tree/master/eddystone-tlm>.
- [27] Google. Eddystone-uid, . <https://github.com/google/eddystone/blob/master/protocol-specification.md>.
- [28] Google. Eddystone-url, . <https://github.com/google/eddystone/tree/master/eddystone-url>.
- [29] Naresh Gupta. *Inside Bluetooth low energy*. Artech house, 2013.
- [30] Jens Emil Gydesen, Kasper Lind Sørensen, and Simon Binderup Støvring. Point and control with gestures in a smart home. Technical report, Aalborg University, 2015.
- [31] David Heckerman. A tutorial on learning with bayesian networks. In *Innovations in Bayesian networks*, pages 33–82. Springer, 2008.
- [32] James Herold and Thomas F Stahovich. The 1¢ recognizer: a fast, accurate, and easy-to-implement handwritten gesture recognition technique. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, pages 39–46. Eurographics Association, 2012.

-
- [33] SmartThings Inc. Smart home. intelligent living., 2015.
<http://www.smarththings.com/>.
- [34] Vandrico Solutions Inc. List of wearables, September 2015.
<http://vandrico.com/wearables/list>.
- [35] Berg Insight. Smart homes and home automation – 3rd edition, 2014.
http://www.berginsight.com/ShowReport.aspx?m_m=3&id=201.
- [36] Frank Jensen, Madsen Lang, AL Madsen, et al. Hugin-the tool for bayesian networks and influence diagrams. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 2005.
- [37] Juha Kela, Panu Korpipää, Jani Mäntyjärvi, Sanna Kallio, Giuseppe Savino, Luca Jozzo, and Di Marca. Accelerometer-based gesture control for a design environment. *Personal Ubiquitous Comput.*, 10(5):285–299, July 2006. ISSN 1617-4909. doi: 10.1007/s00779-005-0033-8. URL <http://dx.doi.org.zorac.aub.aau.dk/10.1007/s00779-005-0033-8>.
- [38] Uffe B Kjaerulff and Anders L Madsen. Bayesian networks and influence diagrams. *Springer Science + Business Media*, 200:114, 2008.
- [39] B. Kloeck, S. Suzuki, S. Tsuchitani, M. Miki, M. Matsumoto, K. Sato, A. Koide, N. Ichikawa, Y. Kawai, and H. Ebine. Capacitive type semiconductor accelerometer, September 14 1993. URL <https://www.google.com/patents/US5243861>. US Patent 5,243,861.
- [40] Sven Kratz and Michael Rohs. A \$3 gesture recognizer: Simple gesture recognition for devices equipped with 3d acceleration sensors. In *Proceedings of the 15th International Conference on Intelligent User Interfaces, IUI '10*, pages 341–344, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-515-4. doi: 10.1145/1719970.1720026. URL <http://doi.acm.org.zorac.aub.aau.dk/10.1145/1719970.1720026>.
- [41] Microsoft. Get started with the microsoft band sdk, .
<http://developer.microsoftband.com/bandSDK>.
- [42] Microsoft. Microsoft band sdk documentation, . <http://developer.microsoftband.com/Content/docs/Microsoft%20Band%20SDK.pdf>.
- [43] George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.
- [44] Motorola. Moto 360. <http://www.motorola.com/us/products/moto-360>.
- [45] openHAB UG. Openhab hardware faq, .
<https://github.com/openhab/openhab/wiki/Hardware-FAQ>.
- [46] openHAB UG. What is openHAB?, .
<http://www.openhab.org/features/introduction.html>.
- [47] openHAB UG. openhab, 2015. <http://www.openhab.org/>.
- [48] openHAB UG. openhab supported technologies, 2016.
<http://www.openhab.org/features/supported-technologies.html>.

- [49] Michael J Pazzani, Jack Muramatsu, Daniel Billsus, et al. Syskill & webert: Identifying interesting web sites. In *AAAI/IAAI*, Vol. 1, pages 54–61, 1996.
- [50] Pebble. Accelerometerservice // pebble developers. https://developer.pebble.com/docs/c/Foundation/Event_Service/AccelerometerService/.
- [51] ASM Rahman, Jamal Saboune, and Abdulmotaleb El Saddik. Motion-path based in car gesture control of the multimedia devices. In *Proceedings of the first ACM international symposium on Design and analysis of intelligent vehicular networks and applications*, pages 69–76. ACM, 2011.
- [52] Reemo. For seniors and families. <http://www.getreemo.com/for-seniors-families/>.
- [53] Samsung. Samsung gear s2 - the official samsung site, . <http://www.samsung.com/global/galaxy/gear-s2/#!/spec>.
- [54] Samsung. Compatible Products | SmartThings, . <https://www.smarththings.com/compatible-products>.
- [55] Samsung. Overview | SmartThings Developers, . <http://developer.smarththings.com>.
- [56] Eclipse SmartHome. Eclipse SmartHome Items. <https://eclipse.org/smarthome/documentation/concepts/items.html>.
- [57] Thad Starner, Jake Auxier, Daniel Ashbrook, and Maribeth Gandy. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring. In *Wearable computers, the fourth international symposium on*, pages 87–94. IEEE, 2000.
- [58] Statista. Wearable device market value from 2010 to 2018 (in million u.s. dollars), 2015. <http://www.statista.com/statistics/259372/wearable-device-market-value/>.
- [59] Danmarks Statistik. Bol103, boliger efter område, beboertype, anvendelse, antal værelser, boligstørrelse i kvm. og husstandsstørrelse. <http://www.statistikbanken.dk/10064>.
- [60] Todd Andrew Stephenson. An introduction to bayesian network theory and usage. Technical report, IDIAP, 2000.
- [61] Tizen. Bluetooth: Managing bluetooth devices, . <https://developer.tizen.org/development/tutorials/native-application/network/bluetooth>.
- [62] Tizen. Sensor: Using sensors and managing sensor events, . <https://developer.tizen.org/development/tutorials/native-application/system/sensor>.
- [63] Eben Upton. Raspberry Pi Zero: The \$5 Computer, 2015. <https://www.raspberrypi.org/blog/raspberry-pi-zero/>.
- [64] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pages 47–56, 2000.

- [65] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. Gestures without libraries, toolkits or training: a \$1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 159–168. ACM, 2007. URL <http://dl.acm.org/citation.cfm?id=1294238>.

Housing Types

Table A.1: Sizes of inhabited housings in Denmark as of 2015. Data from Danmarks Statistik [59].

Inhabited housings in DK, 2015	
-50 m2	149 433
50-74 m2	528 552
75-99 m2	603 484
100-124 m2	438 493
125 - 149 m2	357 712
150 - 174 m2	252 164
175+ m2	292 739
Unspecified	5 761

Communication With openHAB

The open source home automation hub openHAB described in Section 2.5 exposes an API with a REST architecture. Clients can communicate with the API over HTTP.

The communication with the openHAB API consists of the following two components.

REST client A base REST client suitable for communicating with a REST API which carries its data using JSON. The client implements base methods for interacting with a REST API. This includes retrieving, deleting, updating and creating entities. The base client parses data received by the API into models.

openHAB client The client builds on top of the base REST client and adds openHAB specific methods. This includes retrieving things and items as well as updating the state of an item.

Figure B.1 shows a class diagram of the REST client. The involved components are briefly described below.

RESTClient The base REST client which is responsible for performing requests and mapping response to models that can be further processed or displayed in the application.

RequestQueue Responsible for creating worker threads for network requests.

ResultListener Interface implemented by objects that should receive a result when a network request completes, either because of a failure or because of a successful response.

EntityBuilder Interface implemented by objects mapping from the received JSON to models.

Result Encapsulates a network result. A result can either be a success or a failure. In case of a success, the result will contain a the value received from the API. In case of a failure, the result must contain an error.

Figure B.2 shows a class diagram of the openHAB client. The involved components are briefly described below.

OpenHABClient A specialization of the base REST client implementing openHAB specific functionality.

BooleanResult Similar to the Result, this either represents a success or a failure. In the case of a success, the BooleanResult does not contain a value.

Item Model representing an item in openHAB.

Thing Model representing a thing in openHAB.

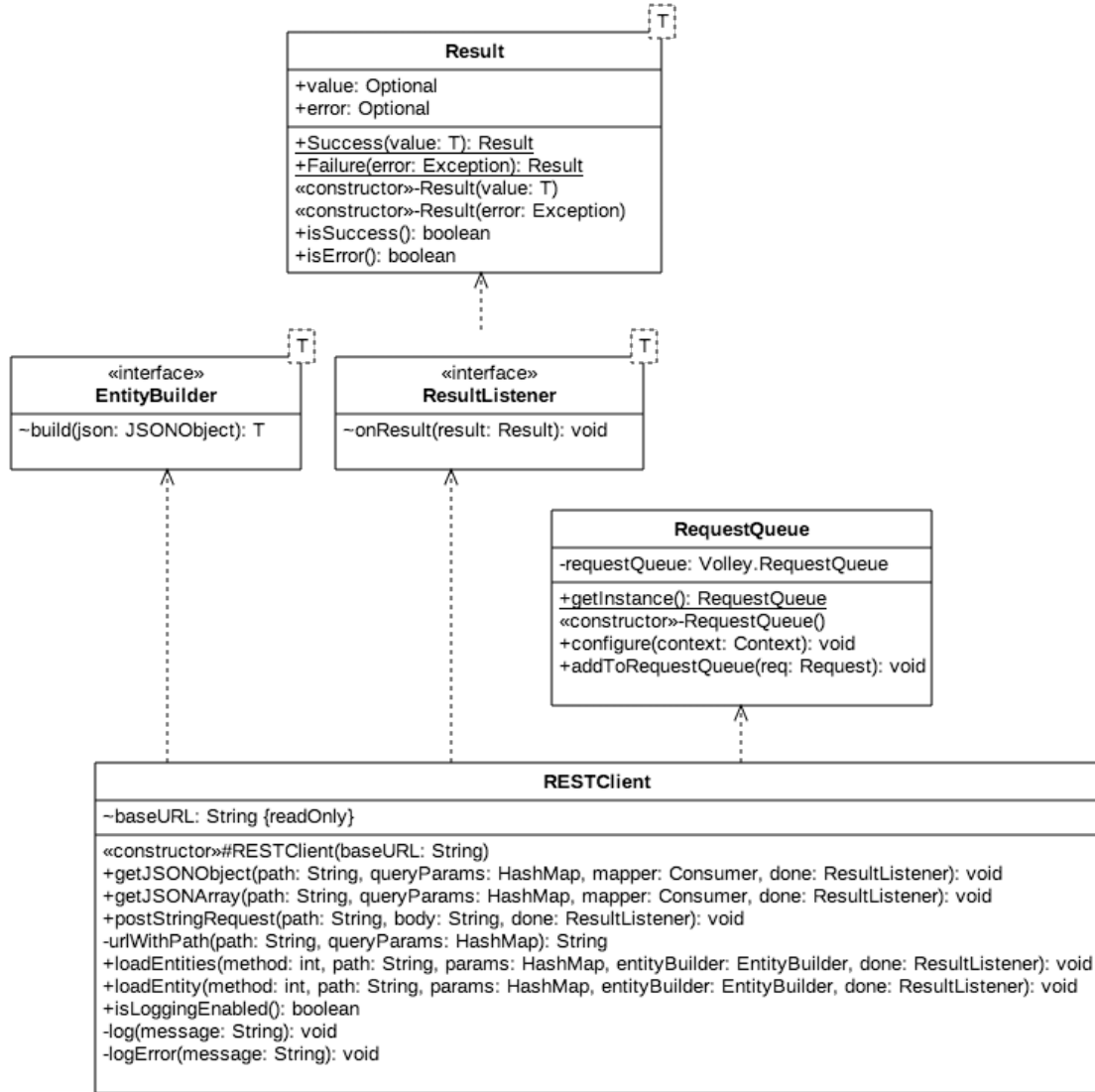


Figure B.1: Class diagram showing the architecture of the REST client used for communicating with a REST API.

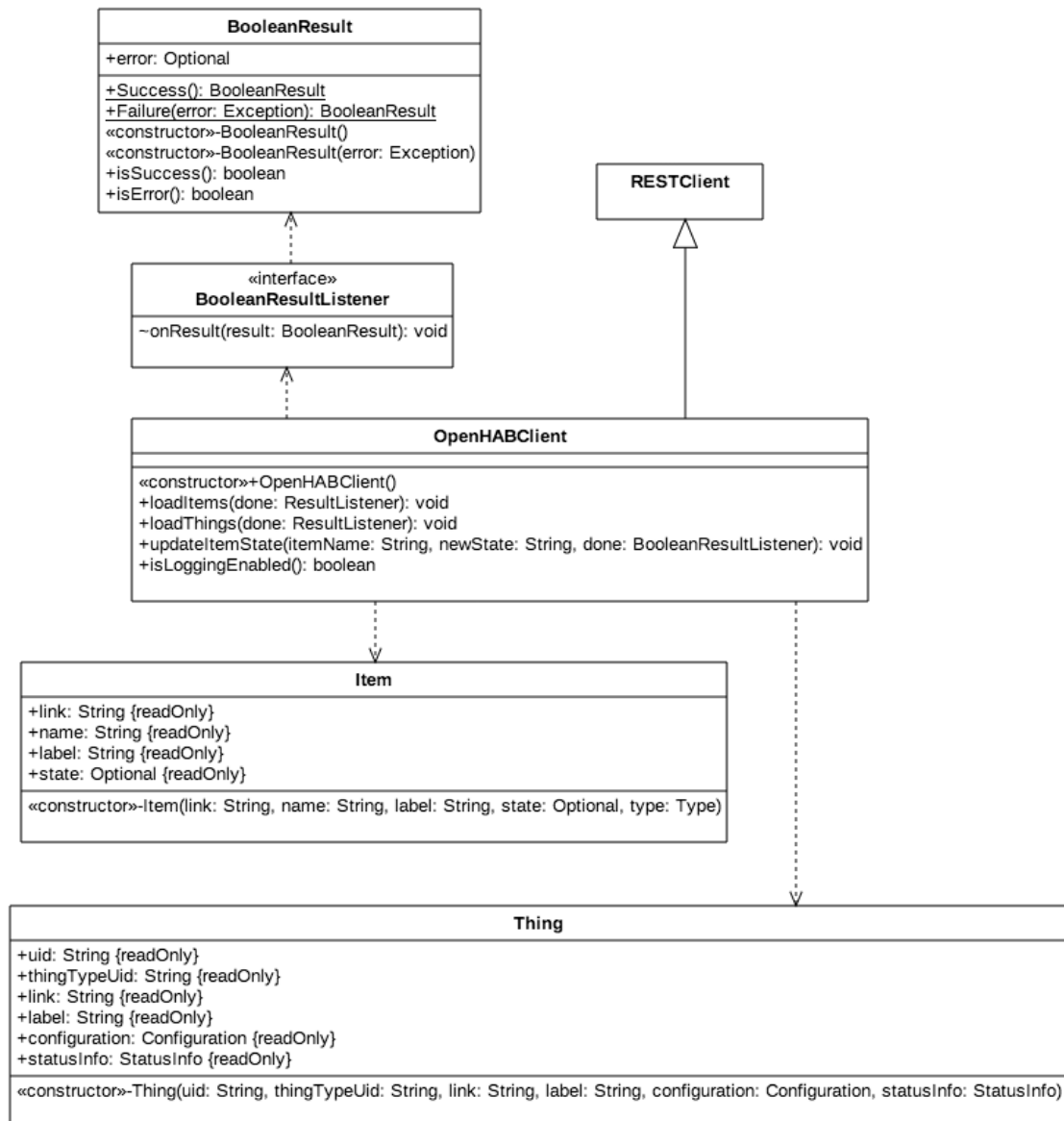


Figure B.2: Class diagram showing the architecture of the openHAB client used for communicating with the openHAB API

User Test Results

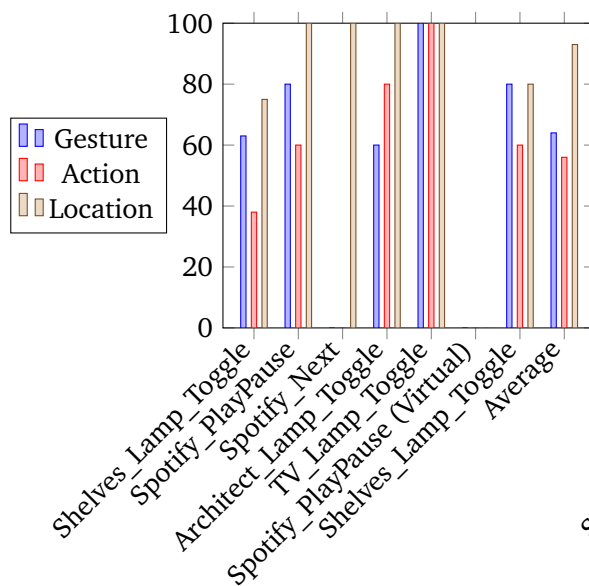


Figure C.1: Participant 1

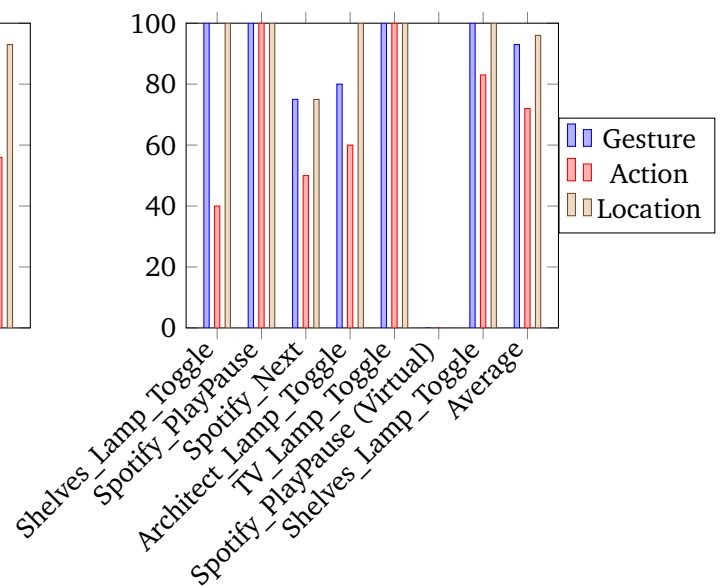


Figure C.2: Participant 2

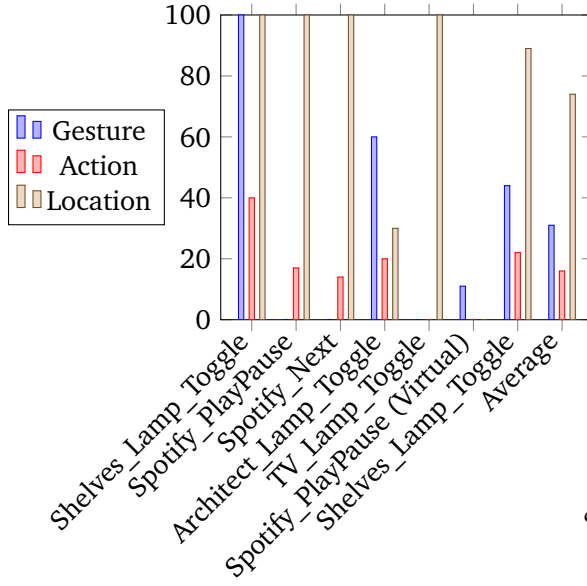


Figure C.3: Participant 3

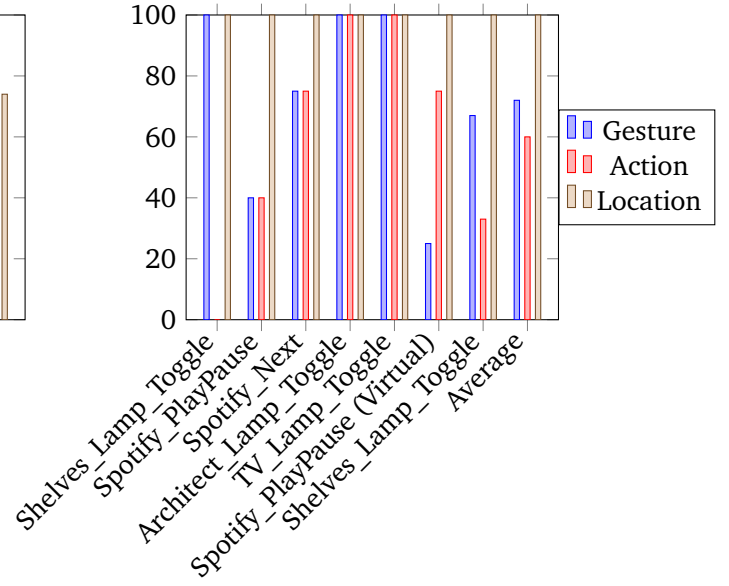


Figure C.4: Participant 4

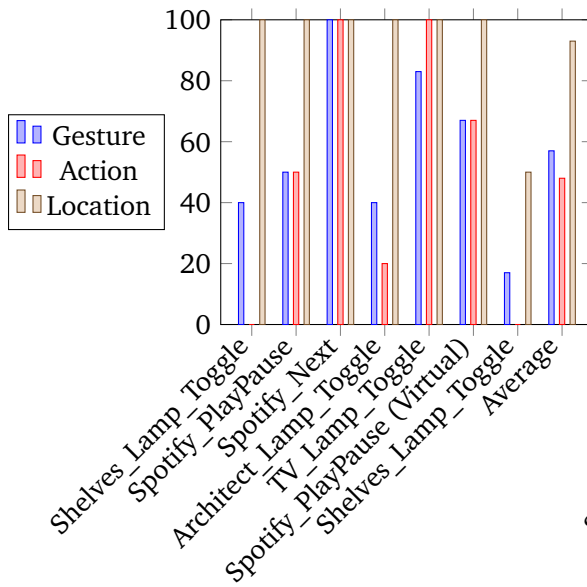


Figure C.5: Participant 5

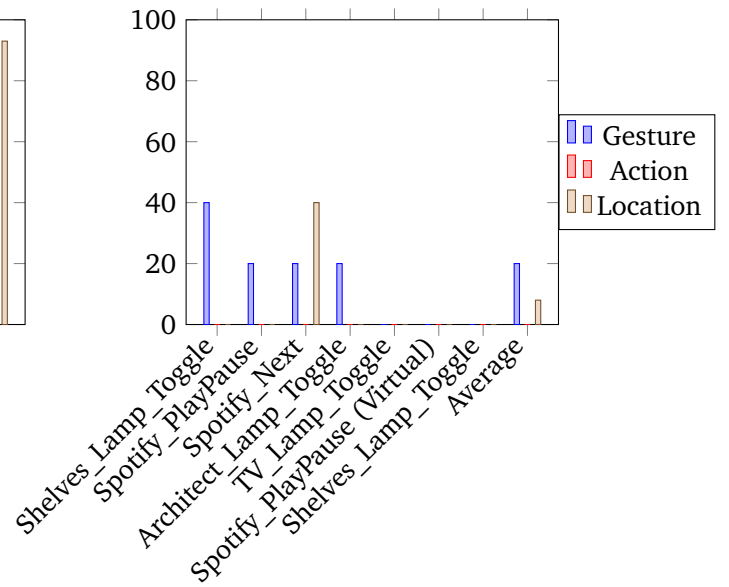


Figure C.6: Participant 6

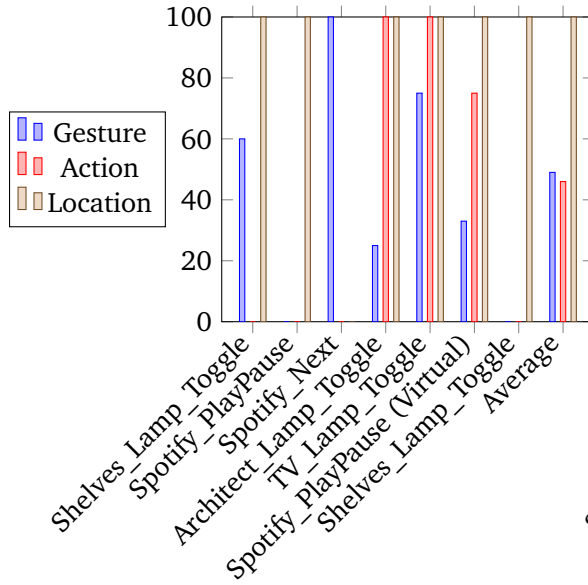


Figure C.7: Participant 7

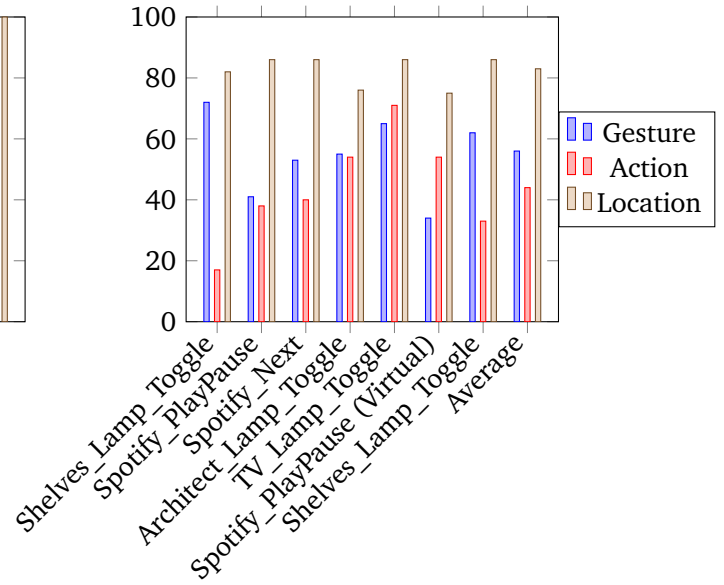


Figure C.8: All participants combined

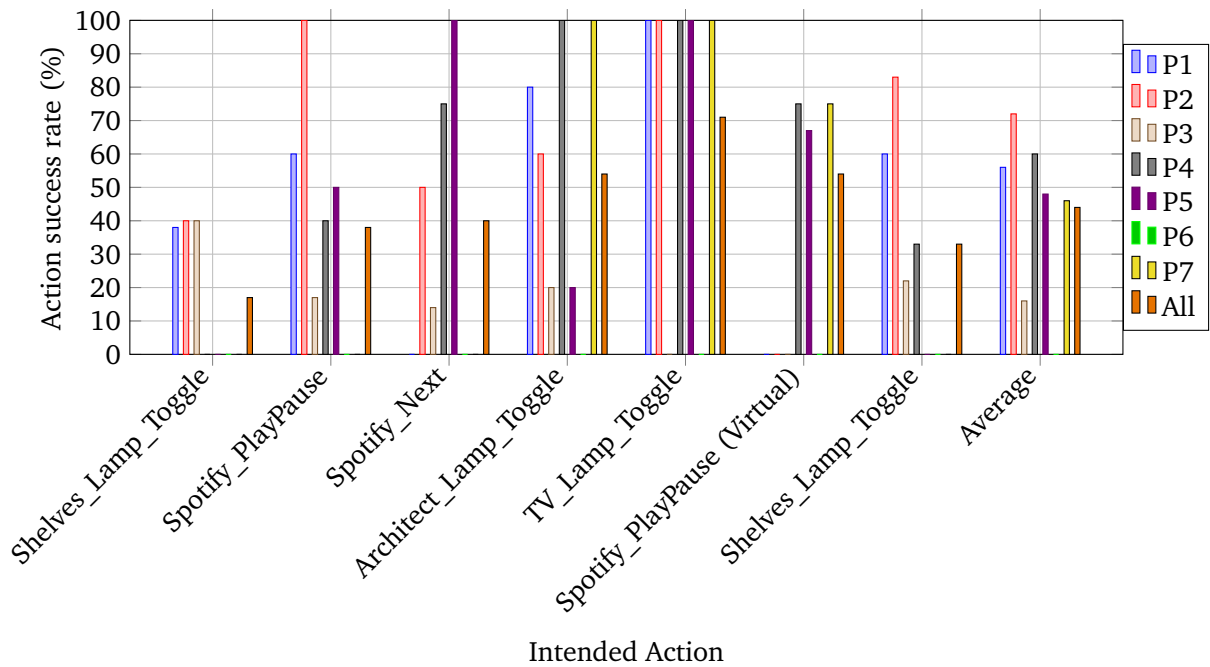


Figure C.9: The rate at which the correct action was triggered for each participant. The last bar for each action is the average success rate of all participants. The last group of bars is the average success rate per user, across all actions.

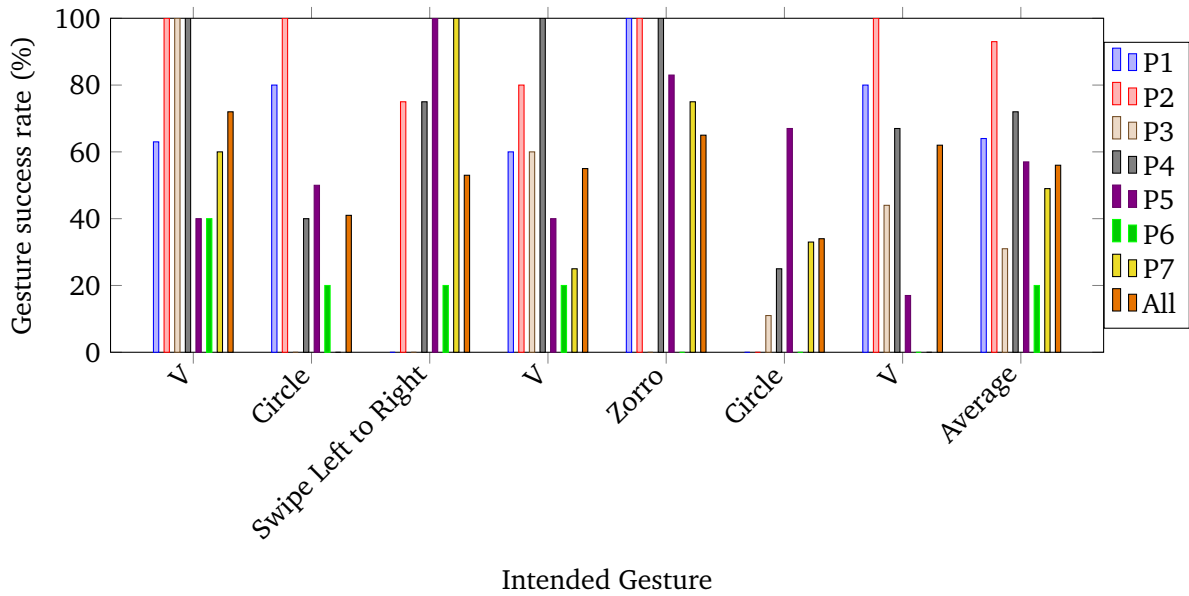


Figure C.10: The success rate for gestures. The last bar for each gesture is the average success rate of all participants. The last group of bars is the average success rate per user, across all gestures.

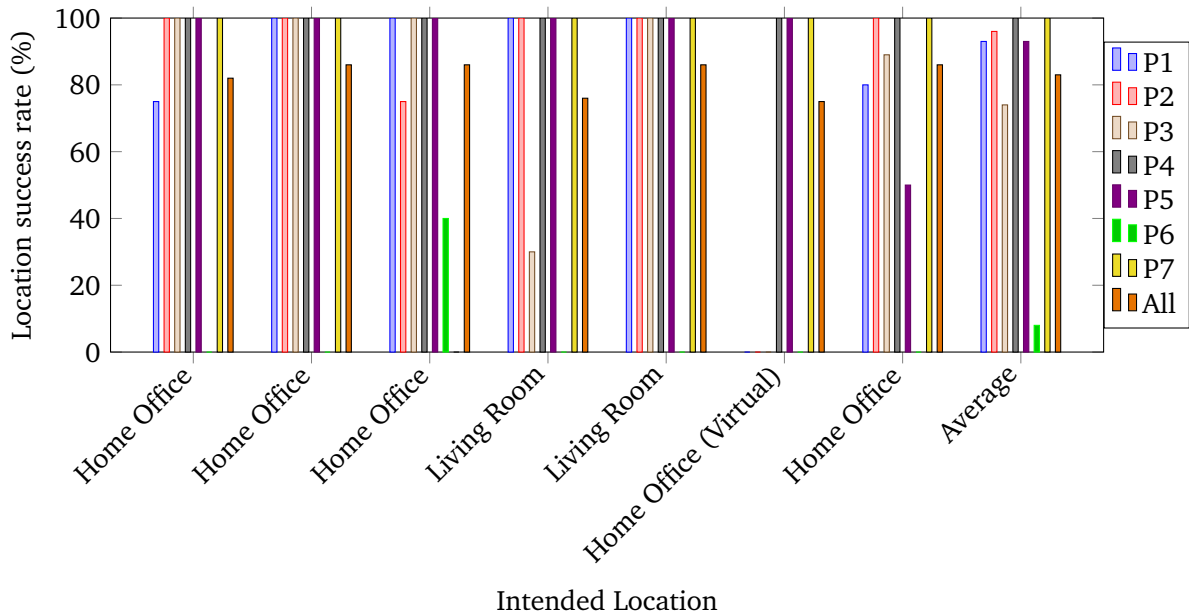


Figure C.11: The success rate for locations. The last bar for each location is the average success rate of all participants. The last group of bars is the average success rate per user, across all gestures.