



Vrije Universiteit Brussel

Faculteit Wetenschappen
Departement Computerwetenschappen

Recognition of Deictic and Iconic Gestures For Home Automation

Proefschrift ingediend met het oog op het behalen van de titel Master in de Toegepaste Computerwetenschappen, door:

Brecht Van Laethem

Promotor: Prof. Dr. Beat Signer
Begeleider: Lode Hoste

JUNI 2012





Vrije Universiteit Brussel

Faculty of Science
Department of Computer Science

Recognition of Deictic and Iconic Gestures For Home Automation

Graduation thesis submitted with intention to obtain the degree of Master in Applied Computer
Science by

Brecht Van Laethem

Promoter: Prof. Dr. Beat Signer
Advisor: Lode Hoste

JUNE 2012



Samenvatting

Sinds de opkomst van smartphones en tablets, zijn multi-touch- en gebaar gebaseerde interactie ingeburgerd in het dagelijks leven van mensen. In deze thesis, zouden wij graag de overgang van een traditionele toetsenbord-en-muis setup naar een natuurlijke gebaren gebaseerde interactie nog verder willen bevorderen. Recente vooruitgangen in camera technologie, zoals de Kinect van Microsoft, laten ons toe om de bewegingen van lichamen te detecteren. Hierdoor wordt het mogelijk om te interageren met huishoudelijke apparaten met behulp van gebaren zonder de nood van een fysieke afstandsbediening. Het is echter verre van triviaal om computers te programmeren voor het herkennen van specifieke bewegingen in real time uit een continue stroom van video-informatie.

Eerst moet een gebaren spotting analyse toegepast worden om aanwijzingen te vinden of en wanneer een persoon een gebaar uitvoert of niet. Dit is belangrijk voor zogenaamde ‘always-on’ interfaces omdat we moeten proberen onbedoelde activaties van huishoudelijke apparaten te beperken. Een gebaren classificatieproces beslist vervolgens welk gebaar uij een vooraf gedefinieerde set van gebaren was uitgevoerd. Voor huisautomatisering varieert dit van het in- of uitschakelen voor een apparaat tot het wijzigen van de tv-zender of het geluidsvolume. Tot slot wordt een locatie schatting gebruikt om het herkende gebaar te koppelen aan een specifiek apparaat.

In dit werk laten we zien hoe Mudra, een regel-gebaseerd gebaren herkenning aanpak, kan worden gebruikt voor het spotten van een gesture door het efficiënt afleiden van semantische informatie uit een continue input stream. Daarnaast hebben we een hybride aanpak ontwikkeld door de integratie van Mudra en iGesture met behulp van XML-RPC. Hiermee tonen we aan hoe extra gesture classificatietechnieken gunstig zijn voor het optimaliseren van hoge precisie. Tevens ontwikkelden we een hulpprogramma voor kamer modelatie met als doel gebaren interactie met de KNX-ondersteunde huistoestellen mogelijk te maken. Ten slotte hebben we onze aanpak met enkele gebaren gevalueerd die gebruikt kunnen worden voor toekomstige gebaren interactie.

Abstract

Since the emergence of smartphones and PCs, multi-touch and gesture-based interaction have been introduced in people's everyday life. In this thesis, we would like to push this change from the traditional keyboard-and-mouse setup to natural, gesture-based interaction even further. Recent advances in camera vision technology, such as the Microsoft Kinect, allows us to track the movement of human bodies. This might allow us to interact with home appliances using gestures without requiring any physical controller at all. However, programming computers to recognise specific movements in real-time from a continuous stream of video information is far from trivial.

First, a gesture spotting analysis has to be performed to find important cues whether and when a person was performing a gesture or not. This is important for so-called 'always-on' interfaces as we need to try to limit unintended behavior towards home appliances. Next, a gesture classification process takes care of deciding which gesture of a predefined vocabulary was executed. For home automation, this varies from enabling or disabling a device to changing the tv channel or lowering the sound volume. Finally, target estimation deals with directing the decoded gesture to a specific device.

In this work, we show how Mudra, a rule-based gesture recognition approach, can be used for gesture spotting by efficiently deriving semantic information from the continuous input stream. We developed a hybrid approach by integrating Mudra and iGesture using XML-RPC and demonstrate how additional gesture classification techniques are beneficial for optimising high precision. A room modeling tool was developed to target gestural interaction with the intended KNX-based home appliances, and finally we evaluated our approach with a scenario of deictic and iconic gestures that could be used for future human-home automation interaction.

Acknowledgements

I would like to thank:

- Prof. Dr. Beat Signer, my promoter.
- Lode Hoste, for his guidance and support during this thesis.
- Pattyn Domotica, for letting us experiment with their home automation setup.

Contents

1	Introduction	1
1.1	Context	1
1.1.1	Home Automation	1
1.1.2	Poses and Gestures	2
1.1.3	Practical use of gestures in home automation	3
1.2	Problem description	5
1.3	Approach	6
2	Related Work	7
2.1	Camera-based gesture recognition	7
2.1.1	Color-based recogniser	7
2.1.2	Time of flight	7
2.1.3	Structured light	8
2.2	Gesture recognition algorithms	11
2.2.1	Rules	11
2.2.2	Template matchers	14
2.2.3	Machine learning	17
2.3	Home automation	20
2.3.1	Why gestures?	20
2.3.2	Important properties	21
2.4	Comparison	22
3	Proposed solution	24
3.1	General architecture	24
3.2	Skeleton tracking	25
3.3	Segmentation using declarative rules	26
3.4	Verification: template matcher	28
3.5	Home automation	33
3.5.1	Modeling room	34
3.5.2	Gesture set	37
3.5.3	Conclusion	37

4	Implementation	39
4.1	Skeleton tracking	39
4.2	Segmentation: declarative rules	40
4.3	Second filtering: template matcher	43
4.3.1	Extending iGesture for camera-based sample recording	43
4.3.2	Semi-automated rule generation	44
4.3.3	DTW for hand trajectory classification in iGesture . .	45
4.3.4	Service oriented iGesture	45
4.4	Home automation	47
5	Validation	49
6	Future work	52
6.1	Multiple cameras	52
6.2	Scanning rooms	52
6.3	Other extensions	53
7	Conclusion	54

Chapter 1

Introduction

1.1 Context

1.1.1 Home Automation

Home automation means that one can remotely control one or more appliances in or around his house. These appliances may include lights, heating, audio sources, etc. This allows for improved convenience and comfort. There is a wide range of possibilities from quite simple to very complex. A simple example can be that one has a button in the bedroom that turns off all the lights in the house. A more complex task can be the control of multiple audio speakers in every room of the house. All the audio data is stored on a central server and one can define which music should be played in which room. This interaction between human and computer is typically done using a touchscreen from a dedicated control panel, a tablet or a smartphone.



Figure 1.1: Typical example of a touchscreen device for home automation

A particular branch of home automation, called “assistive domotics”, focuses on supporting elderly and disabled people in their day to day live in order that they do not need to go to a healthcare facility [3]. Examples include that the resident hears a warning when he or she forgot to take the required medication. Another example can be that sliding doors automatically open when the wheel chair of the resident comes near it.

What we try to achieve with this thesis, is targeted to average users and allowing them to control home appliances with gestures. This could be a more user-friendly approach, compared to the current approaches where buttons and touchscreens are still mostly fixed at a particular location (e.g. built into walls) or require users to carry dedicated hardware (e.g. a remote control or smartphone).

1.1.2 Poses and Gestures

A pose is a position of the human body or parts of the human body. Figure 1.2 illustrates this using fingers for the peace sign. A gesture is broader than a pose and captures the dynamic movement of body parts over a period of time. An example is sign language where not only the poses are important but where the complete movement also contains semantic information.



Figure 1.2: A well known gesture is the peace sign

With the recent change in affordability of 3D cameras, such as Microsoft Kinect¹ and SoftKinetic DepthSense², the movement of body parts can be cheaply and accurately monitored by a computer. Gestures can be divided in several categories like beat (rhythmic beating of body parts), deictic (pointing in a certain direction) or iconic (use of body parts to match the situation one tries to narrate like indicating the size of a certain object) [18].

¹Microsoft Xbox Kinect camera, <http://www.xbox.com/kinect>, [24-May-2012]

²SoftKinetic DepthSense camera, <http://www.softkinetic.com/solutions/depthsensecameras.aspx>, [24-May-2012]

1.1.3 Practical use of gestures in home automation

Gestures can be practical for controlling home automated appliances. Imagine someone is watching television in his living room: if he wants to raise the volume, he needs to search the remote control and press the volume button. Furthermore, in a household, one has often more than one television and having multiple remote controls can be confusing. With gesture recognition, this can be done much simpler by just raising a hand.

This is a realistic scenario as major firms are introducing camera-based interaction. For example, on CES 2012 (Consumer Electronic Show) in Las Vegas, Samsung presented a line of smart televisions³. These are televisions with a built-in microphone and a camera. They can be completely controlled via voice and gestures.



Figure 1.3: Samsung TV with built-in camera and microphone (on top)

³Tweakers News: <http://tweakers.net/nieuws/79232/samsung-laet-gebruikers-tv-bediennen-met-spraak-en-gebaren.html>, [24-May-2012]

Other examples include the Kinect of Microsoft which allows users to play games by moving body parts, as shown in Figure 1.4.



Figure 1.4: People playing with Kinect

1.2 Problem description

In order to invoke certain home automation functionality, gestures have to be understood by a computer. This is the research field of gesture recognition where algorithms are developed to recognize a gesture from a certain input. As gesture recognition has been a research subject for more than 30 years [2], we noticed that existing solutions are either focusing on low-level features (e.g. pixel-based camera input) [1, 8, 15] or high-level features (e.g. finger trajectories for multi-touch devices) [11, 16].

However, the newly introduced 3D cameras support accurate skeleton-tracking solutions, which allows us to use high-level features for camera-based gesture recognition. They generate a skeleton from the human body in a continuous stream of data. The Kinect camera and the Microsoft Kinect SDK ⁴ for instance, can recognize the shape of a person and extract the coordinates of different body parts like head, shoulder, wrist, etc. As a result, the programmer receives a constant stream of coordinates of body parts. It is this stream of coordinates that can be used to perform high-level gesture recognition.

In the ideal case, what we want is that one would have a gesture recognition algorithm that can handle this continuous stream. Furthermore, it should be as accurate as possible, optimised for high precision and it should not require a large amount of time to add a new gesture. There are roughly 3 categories of gesture recognition algorithms:

1. Declarative rules: these algorithms are designed to handle a constant stream of input data. Adding a gesture is done by defining the characteristic points of a gesture, and some spatio-temporal constraints on these points. The letter 'Z' will normally have four characteristic points. A possible spatio-temporal constraint can be that the gesture must be performed within a certain timespan. The algorithm will read the continuous stream of input data (which are coordinates) and check whether the coordinates go through the characteristic points of the gesture while respecting the spatio-temporal constraints. On the downside, these algorithms do not scale well for curved motion trajectories [6] which translates in lower accuracy if developers generalise gestures for multiple users.
2. Template matchers: these algorithms will compare the input data (which we call a sample) with a collection of stored gestures (which we call templates). They will check if the sample matches with a template. To do the matching, both the sample as well as the templates need to have a limited length. This length can be different but it can-

⁴Microsoft Kinect SDK, <http://www.kinectforwindows.org>, [24-May-2012]

not be unlimited and should be well segmented (i.e. the start and stop point should be defined). Therefore, these algorithms cannot easily handle a continuous stream of data as input.

3. Machine learning: these algorithms will try to learn a model of different gestures. However, this requires that the user repeats the gestures over and over so the algorithm can learn them. Therefore, adding new gestures can take a considerable amount of time which is not ideal for our scenario as we are still in the exploration phase of these kinds of interfaces.

As one can conclude, neither of these algorithms offer the ideal blend of being able to handle a continuous stream of data, support the fast adding of gestures, combined with high precision. Additionally, there is a lack of modeling tools that integrate home appliances with gestural interaction.

1.3 Approach

Our gesture detection approach consists of a hybrid, two phase filtering. The stream of input data goes through a first filter which is declarative rules. For this we based ourselves on existing work of Mudra [7] where high-level features from multi-touch input were used to easily spot gestures. As mentioned before, these declarative rules lack accuracy so there is a need for verification. This is where the second filter comes in place. We will verify the potential matches of the declarative rules with a template classifier. When the template classifier confirms that the gesture is correct, we have correctly identified a gesture. By using the combination of declarative rules and a template matcher, we combine the good properties of both algorithms. It can handle continuous streams of data, the algorithm does not need to learn the gestures and it should obtain high accuracy. We have extended the iGesture framework to 1) allow users to easily design new gestures and generate rules in a semi-automated manner and 2) to use the existing template-based classifiers of iGesture to verify the results of Mudra.

To target these gesture recognition results to home appliances, we have built a 2D room modeling application where both cameras and devices can be located. This allows us to interpret the direction of a deictic gesture to a home appliance. This modeling application is also integrated with the current state of the art KNX home automation protocol. This protocol is a network layer capable of interacting with all devices in the room. Finally, we provide an evaluation of a number of gestures that can be used in home automation environments and highlight the limitations of current work.

Chapter 2

Related Work

2.1 Camera-based gesture recognition

2.1.1 Color-based recogniser

A lot of work has been done already using traditional color-based cameras, such as webcams but also more expensive HD cameras. The main problem with these devices is that it is very hard to extract the human body parts.

To extract the hand or arm of a person from the image, one needs to do skin color featuring [15]. This means that one has to be able to extract areas with a skin color. As one can imagine, this is not an easy thing to achieve. There are many variations in skin color worldwide. Furthermore, as one has only a two dimensional image, it is very difficult to precisely extract the direction to which one is pointing.

2.1.2 Time of flight

A time of flight camera, such as SoftKinetic's DepthSense camera, use light pulses. The light pulses are emitted and reflected by objects [4]. The camera then gathers the reflected light. By measuring the time between emitting the light and receiving it, one can easily calculate the distance to the camera. This can be done with the following formula:

$$\text{Time measured} = \frac{2 * \text{distance to object}}{\text{speed of light}}$$

One can easily understand that the signal travels 2 times the distance: first from the camera to the object and second from the object back to the camera. The speed of light is a constant ($3 * 10^8$ m/s).

The time needed for the light pulses defines the minimal distance that one can measure with a time-of-flight camera. The light pulse can only be received when one is not emitting light at the same time. Imagine one has a light source that can be turned on and off within a time span of 70 milliseconds. Using the formula from above, we can transform it into:

$$\text{Distance to object} = \frac{\text{time measured} * \text{speed of light}}{2}$$

If we fill in the data (time measured = 70 ms, speed of light constant), we get 10.5m as result. Objects must be at least this far away from the camera, otherwise the light pulse will be back while we are still emitting light.

This allows developers to use 3D information for 1) extracting the human body parts and 2) interpreting deictic gestures.

2.1.3 Structured light

The most prominent example of a structured light sensor is the Microsoft Kinect camera. Microsoft launched the Kinect in November 2010 as an extra input device for their game console Microsoft Xbox 360. It allowed the user to play games with gestures and spoken commands. Already one month later, on December 2010, an open source driver was released in order to connect the Kinect to a PC. The driver was released by PrimeSense, who delivered the depth sensing reference design for the Kinect. About half a year later, on June 2011, Microsoft launched the official SDK. Next to skeleton tracking, it offers also advanced audio capabilities like recording sound, echo cancelation and connection with the Microsoft Speech SDK.

The camera has 2 parts: a projector and an infrared VGA camera. The projector projects a map of infrared dots across the room. These dots are not placed at random positions but on such positions that a camera can correlate a position to a pattern of dots. The infrared camera sees this pattern of dots. This data is sent to a PC via a USB connection where a three dimensional image is generated. With this image, skeleton tracking can be applied.

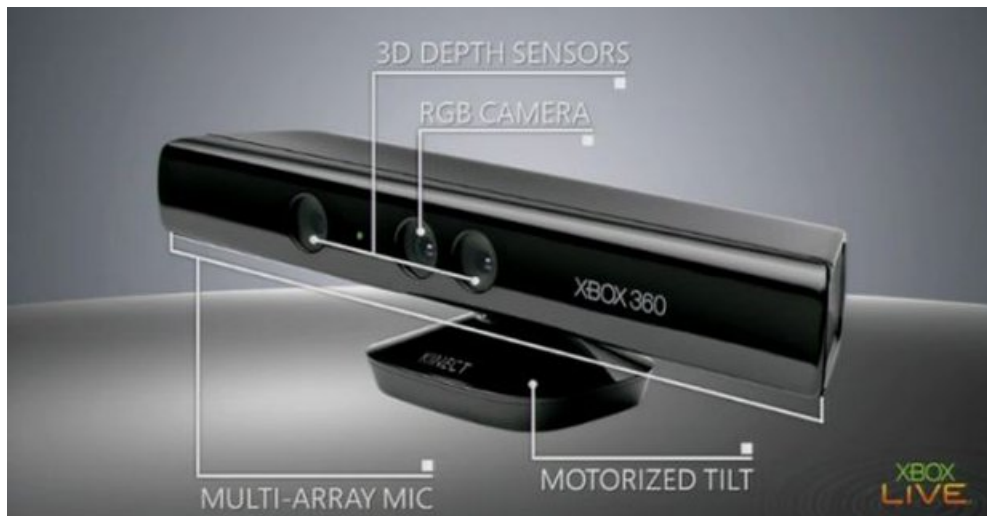


Figure 2.1: Properties of Kinect

There are a few disadvantages when using the Kinect in a home environment. The first disadvantage of using the Kinect in home automation is its limited range, as shown in Figure 2.2. There are two camera modes: default and near mode. Near mode gives better results at short distance but worse results on a further distance, compared with the default mode¹. This difference is realized by using different firmware for the camera. In both cases, a Kinect camera will only detect people within a depth range of a few meters. This can be problematic in larger rooms where one probably needs more than one Kinect to cover the entire room.

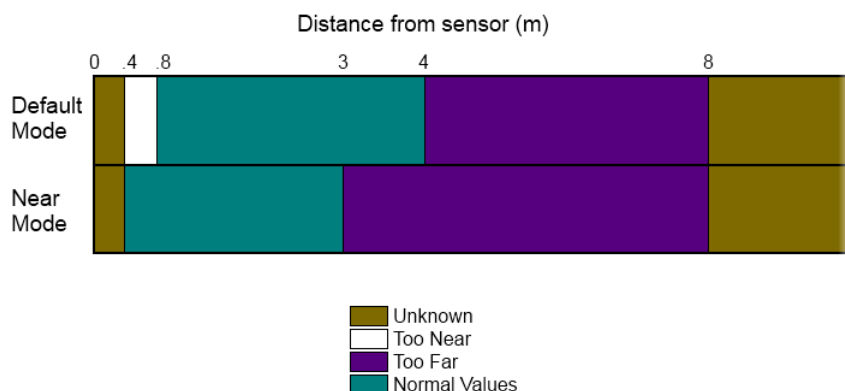


Figure 2.2: Measurable distance with a Kinect in normal and near mode

¹Kinect Near-mode: <http://blogs.msdn.com/b/kinectforwindows/archive/2012/01/20/near-mode-what-it-is-and-isn-t.aspx>, [24-May-2012]

A second disadvantage of using the Kinect is its sensitivity for light sources. The infrared VGA camera of the Kinect has difficulties with reading the infrared dots when there is direct sunlight. Especially in the context of home automation this can be problematic. A possible solution for this problem is to use another type of camera which does not have this dependency.

2.2 Gesture recognition algorithms

2.2.1 Rules

Most popular solutions for gesture recognition are based on template or machine learning algorithms. Generally they require a clear definition of the begin and end point of the gesture (i.e. segmentation). This is often enforced by letting the user stand still before and after the gesture [13]. In daily use, this is not a practical solution. What a user wants is that he can do his gesture without pausing. Therefore, the software must be able to spot the gestures from a continuous stream of data. This process is called *gesture spotting*.

Ideally, one wants to have a clear separation of concern between the one developing the general application and the one developing the different gestures. Therefore, a good solution should have the following properties:

- **Modularization**
It should be possible to implement an additional gesture without the necessity to have a deep knowledge about the already implemented gestures.
- **Composition**
It should be possible to create a complex gesture by composing it from simpler gestures. It not only makes the code of complex gestures easier to read but it also reduces the duplication of code.
- **Temporal and spatial operators**
When one tries to extract meaningful gesture data from a stream of data, the use of spatial and temporal operators comes in handy. By offering this functionality to the developer, he can more easily define the gestures.

A possible solution for good gesture spotting is the use of declarative rules. With these rules, one can define control points with their corresponding spatial and temporal constraints. A gesture is recognized if it goes through all control points while respecting all spatial and temporal constraints.

These rules provide a number of powerful properties, in addition to the ones defined above:

- **Non-subsequent event matching**
Events that do not match the constraints are skipped but can be reused as a starting point of another gesture.

- Negation
One can for instance specify that there are no points allowed with relative y-values (compared to a control point) larger than a certain threshold
- Overlapping sub-matches are allowed

A possible implementation of this approach is Mudra [7]. We will extend the implementation in the context of our work.

Mudra

There are two types of complexity in coding: accidental and essential complexity. Accidental complexity is caused by using inappropriate software engineering tools. By selecting a more appropriate software language or framework, the accidental complexity can be reduced. Essential complexity is the complexity inherent to the problem one tries to solve and it cannot be reduced. Mudra was developed because current frameworks for multi-touch gesture recognition are often not adequate [10] and therefore generate a lot of accidental complexity. By introducing a new programming language, Mudra tries to reduce the accidental complexity [7], compared to existing implementations currently available for programmers.

To obtain a separation of concerns between the one developing the general application and the one developing the different gestures, Mudra is based on a three-layered architecture, as shown in Figure 2.3.

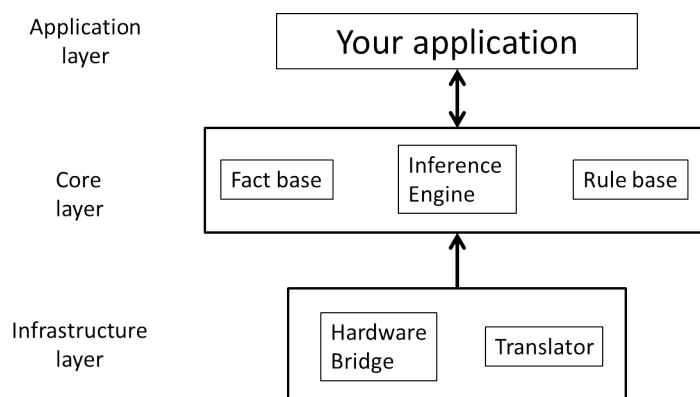


Figure 2.3: Mudra architecture

The infrastructure layer is responsible for the communication between the hardware and Mudra. It has two parts: a hardware bridge and a translator. The hardware bridge is responsible for extracting the data from the device. If one has multiple devices, then it will also have multiple hardware

bridges. In many cases, the hardware bridge is the SDK provided with the hardware. All these devices will probably have a different way for representing their data. Therefore, the extracted data needs to be sent to the translator which translates the data in a uniform format that is understandable by Mudra. After the data is translated, it is sent to the Mudra core layer.

Mudra stores the data from the infrastructure layer in its fact base. This is a collection of all the facts that it can use to check compliance with the rules. Possible facts can be coordinates but also triggered gestured from the past. For the sake of performance, in a rule one can define the maximum timespan for a gesture. Mudra will automatically remove all facts from the fact base that are older than the maximum timespan allowed. These rules are defined in the application layer and sent to Mudra, which will store them in the rule base. The inference engine will combine the fact base with the rule base to check if there are any matches. If a match is found, Mudra will notify the application about it.

2.2.2 Template matchers

Template matchers are one mean to recognise gestures. The idea is to calculate similarity values between a gesture and several stored samples. This section will discuss the algorithm “Dynamic Time Warping” [5] algorithm which is an example of a template matcher.

Dynamic Time Warping

This algorithm is used to calculate the similarity between two time-series, not necessarily with the same length. Distance measuring (such as Euclidean distance) cannot be used for measuring the similarity between two time-series. As an example, if there are two time series which are identical but out of phase, the Euclidean distance will give a large distance. DTW overcomes this problem because it ignores global and local shifts in time.

One-dimensional DTW The algorithm will try to match two one-dimensional time series, called x and y . Every possible matching between x and y is called a warping path. A matrix with size $|x| * |y|$ is created to represent the matching of individual points. Afterwards, a warping path can be drawn on the matrix, as shown in Figure 2.4. Each cell in the matrix represents the accumulated minimum warping cost so far.

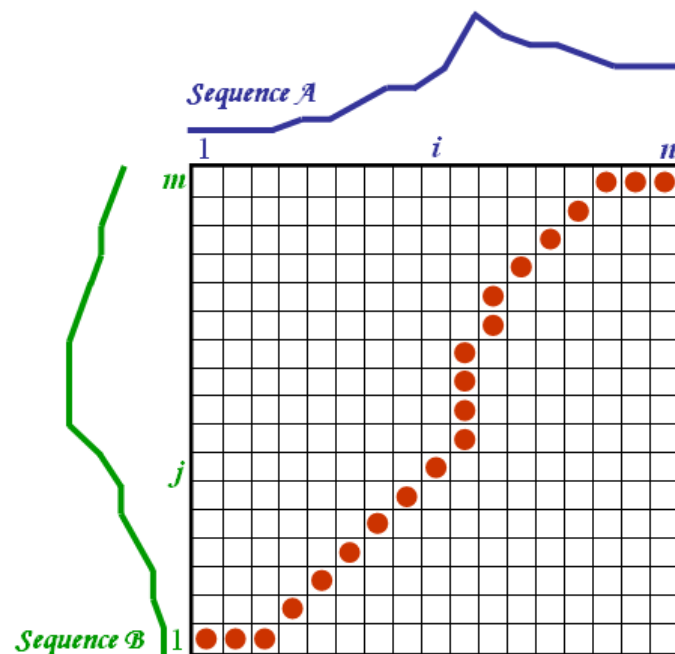


Figure 2.4: matrix with warping path

To reduce the number of different warping paths, some constraints are applied:

- the warping path must start at (1, 1)
- the warping path must end at ($|x|$, $|y|$)
- the warping path must be continuous, i.e. if the algorithm is in (i, j) , the next element in the path must either be (i, j) , $(i + 1, j)$, $(i, j + 1)$ or $(i + 1, j + 1)$.
- the warping path must have a monotonic behaviour, i.e. it cannot move backwards

In general, one is only interested in the path that gives the “best” match. This means that we are looking for the path with the lowest normalised total wrapping costs, given by:

$$\min \frac{1}{|w|} \sum_{k=1}^{|w|} DIST(w_{ki}, w_{kj})$$

$DIST(w_{ki}, w_{kj})$ represents the distance between point i in time serie x and point j in time serie y . The best path is calculated by first calculating each cell of the previously defined matrix with the following formula:

$$C_{(i,j)} = DIST(i, j) + \min\{C_{(i-1,j)}, C_{(i,j-1)}, C_{(i-1,j-1)}\}$$

Afterwards, the wrapping path is found by back-tracking from position ($|x|$, $|y|$) to position (1, 1).

Optimisation DTW is a computationally quite costly algorithm because each cell in the matrix needs to be calculated. This makes DTW less suitable for real-time recognition. However, the original method can be optimised. When having a (reasonable) good match, the warping path will lie near the diagonal, as can be seen in Figure 2.4. By constraining the warping path so that it cannot drift too far from the diagonal, the algorithm does not need to calculate the entire matrix anymore.

N-dimensional DTW To go from two one-dimensional vectors to two N-dimensional vectors, only the distance function needs to be changed in order that it can handle N-dimensional vectors. The new distance function is:

$$DIST(i, j) = \sqrt{\sum_{n=1}^N (i_n - j_n)^2}$$

Templates

As seen before, DTW is an algorithm that compares an input sequence with a sequence stored on the computer. The stored sequence is called a template. There is a separate template for each gesture that needs to be recognised. In general, users give more than one sample for a template. There are several ways of handling more than one sample. One can return the smallest distance of the samples, or the average distance.

Classification

Classification means that the algorithm will try to match the input time series with all the templates. The template with the lowest distance is returned. The matching with the different templates can happen in parallel.

In some cases, the best match can still be quite poor. This is the case when the input-data does not have an obvious match with one of the templates. In this case, one expects that the algorithm will return a message to say that no good match has been found. To realise this, a simple threshold is added.

2.2.3 Machine learning

Machine learning algorithms perform gesture classification by learning the underlying model of a gesture. This section will discuss the technique called Hidden Markov Models(HMMs) [14]. It is used for the recognition of gestures that one can not model but where closely related data is available.

Markov model

When there is direct access to the data (reality) that needs to be recognised, the modeling of the data can happen in two ways: by using a deterministic system or a non-deterministic system. In the case of a deterministic system, the reality is modelled by a finite state machine where each state has only one successor and one predecessor. In contrast, each state in a non-deterministic system can have multiple predecessors and successors. Every possible transition from one state to one of its successors has a probability which can be different, depending on the current state and its successors. All these transition probabilities are stored in a time-independent matrix which is called a state transition matrix.

A Markov assumption means that one assumes that when modeling the reality, the current state of the model only depends on the previous states of the model and not on any other data. Furthermore, if the current state only depends on the n previous states, this is called an order n model. For instance, the current state of a first order model only depends on the previous state. Each system that can be modelled in this way, is called a Markov process. The corresponding model is called a Markov model.

Hidden markov model

A Markov process can sometimes not be powerful enough to model the reality. This is the case when there is no access to the data one wants to model but rather to closely related data. For instance, when applying Markov modeling to speech recognition problems, one cannot access the vocal chord or the position of the tongue themselves, but rather data coming from various sensors (e.g. microphone) that allows to capture the produced sound.

Therefore, to model this the Markov model is extended with an extra pair of states. As a result, a Markov model now has two pair of states: the states one can observe (e.g. sound of a person) and the “hidden” states one can not observe (e.g. vocal chords). Subsequently, the notion of the probability of being in a certain observable state, given a hidden state is introduced. These probabilities are stored in a time-independent matrix, called a *confusion matrix*.

A hidden Markov model is a Markov model which has the following properties:

- It has a vector which indicates the probability of starting in a certain hidden state at time 0.
- It has a state transition matrix which indicates the probability of going to a certain hidden state, given the current hidden state.
- It has a confusion matrix which indicates the probability of being in a specific observable state, given a hidden state.

Problems

In general, HMMs are used to solve the following three types of problems.

Problem 1: evaluation This section discusses the problem of calculating the probability of an observed sequence, given a HMM. For instance with speech recognition: when one has a separate HMM for each word, which HMM has the highest probability given a certain speech sample? The easiest way to calculate the probability of an HMM is by taking every possible permutation of hidden states and calculate for each permutation the probability for generating the observed sequence. However, in a real-life example, this becomes infeasible due to the large amount of permutations.

Therefore, a forward algorithm is used. First, the notion of partial probability is introduced:

$$\alpha_t(j) = \Pr(\text{observation} \mid \text{hidden state is } j) * \Pr(\text{all paths to state } j \text{ at time } t)$$

The chance of seeing the observation at time t in state j is equal to the chance of having the observation in state j , multiplied with the chance of any path to j in time t . The probability for all paths to state j at time t can be calculated with the following recursive algorithm:

time $t = 1$: $\Pr(\text{all paths to state } j \text{ at time } t)$ is equal to the initial probabilities.

time $t + 1$: $\Pr(\text{all paths to state } j \text{ at time } t + 1)$ is equal to:

$$\sum_{x=1}^{\text{all states}} \Pr(\text{all paths to state } x \text{ at time } t) + \Pr(\text{paths from } x \text{ to } j)$$

The total probability of a given HMM is calculated by adding up the probabilities of the states at the last time unit. This algorithm is much faster than the previously discussed method because it eliminates unnecessary duplicate calculations.

Problem 2: decoding To improve the modeling of the reality, one often wants to know which hidden states are visited and in which sequence they are visited. This can be done by using the Viterbi algorithm [14]. It will keep track of the most probable path for getting into a certain intermediate state. To achieve this, the algorithm uses the notion of partial probability like defined in the previous problem. Calculating the most probable path is done by using the following recursive algorithm:

$\delta(i, t)$ is the path with the highest probability to get to state i in time ts .

$$\delta(j, t + 1) = \max_{\text{all } i} [\delta(i, t) * \text{Pr}(\text{paths from } i \text{ to } j)]$$

Calculating the best path to j in time $t + 1$ is done by taking the best paths for every state at time t and multiplying them with the chance of going from that state to state j . The path with the highest probability will be a new path to j in time $t + 1$. The back tracing is done by using pointers to the previous state.

Problem 3: learning The training of an HMM is typically done using the Viterbi or BaumWelch [14] algorithms. Given a sequence of observations, these algorithms will incorporate knowledge. These algorithms are a case of the expectation maximization algorithm. The algorithm will start with some random chosen initial values for the parameters (initial distribution, transition probabilities and emission probabilities) and try to optimize them.

Remarks Hidden Markov models have already been successfully applied to numerous domains such as speech and gesture recognition. However, the main disadvantage of these types of learning techniques is that they require a large number of samples before they are able to grasp the model of the gesture, which is perfectly normal. However, for our use case, we would like to experiment with a number of gestures and perform rapid prototyping as we are unaware which gesture interaction descriptions are more natural than others. Additionally, the learning of a robust garbage state that allows for high precision is also hard to achieve with few training data.

2.3 Home automation

In this section, we argue why gestures are a good candidate to be applied in home automation. Furthermore, we will also discuss the important properties one needs to take into account when defining gestures in the context of home automation.

2.3.1 Why gestures?

Traditional remote controls have many problems [9]. They are often covered with too many small buttons. The text labels on or next to the buttons are cryptic and often so small that they are difficult to read. Furthermore, one needs to have the remote always near in order to use it.

Portable touchscreens (e.g. Tablet or mobile phones) are emerging to be used as a remote control. However, they often have the same problem as traditional remote controls. Moreover, they have a dynamic interface which makes them potentially harder to learn for elderly persons.

Although *voice control* seems the ideal solution, there are also problems with this modality. The first problem is inferior speech recognition with surrounding noise. This noise can come from other people having a conversation, or music from the TV or music installation. The second problem is that speech is not always the most graceful interface. One can imagine somebody is giving a party and the host wants to dim the lights in the room during a conversation. First, he needs to ask his visitors to be quiet. Then, he needs to loudly state a sentence like “computer, lower lights to level 2”. This is clearly not an ideal solution. If one would have gestures, the lights could be dimmed discretely via a gesture.

The amount of *gestures* one can easily remember is limited. This amount can be reduced by using it in conjunction with various types of contextual awareness. Possible options are:

- Only gestures: this means that one has a different gesture for each function. There is no awareness of location so if one has multiple televisions in the house, he needs to have a different gesture for each television.
- Gestures and location awareness: this means that gestures within one room/space need to be different. However, gestures can be reused in different rooms. When one has two televisions, one in the living room and one in the bedroom, he can use the same gesture for both televisions.

- Gestures and direction awareness: the subtle difference with the previous option is that with direction, we do not only know in which room one is, but also in which direction he is pointing. One can imagine a living room with multiple lights. If one wants to turn on a specific light, he can do the gesture for turning on the light while pointing in the direction of the corresponding light. If there would be no awareness of direction, one would have a different gesture for each light in the room.
- Gestures and speech: in this case, speech recognition can be used to reduce the amount of possible gestures. For instance, one can have a gesture for higher. By first pronouncing the correct appliance (e.g. television, thermostat or light), followed by the up gesture, one can easily control different appliances in his home with a small amount of gestures.

2.3.2 Important properties

A very important property that one needs to take into account is the probability that one will be seated while executing the gestures. One can imagine that this probability is quite high for controlling the television. User tests showed that downward motions cannot be effectively executed by seated users [12]. Therefore, one should avoid designing gestures with downward movements when there is a high probability that they will be executed in a seated position.

Another property that is important to consider is the available space. When one is sitting on a couch with other people, one does not want to hit or disturb the others while performing a gesture. This has mostly implications on the available horizontal space.

What also needs to be taken into consideration is the accuracy of the camera and the tracking of persons. When one has poor tracking capabilities, the gestures need to be large enough so that the tracking errors remain marginal. This can sometimes be conflicting with the previous property which limits the available horizontal space.

2.4 Comparison

In this section, we will compare the different algorithms and explain why we have chosen for a combination of declarative rules with a template matcher.

	Template matcher	Machine learning	Declarative rules
Need for start and end points	Yes	Usually	No
Large sample data	No	Yes	No
Precision	High	High	Low
Decidability	Medium	Medium	High
Modularization	High	Medium	High

It is clear that a template matcher needs a start and end point, as explained earlier. Declarative rules are designed to work without start and end points. Machine learning is the only of the three categories listed above that needs training. For precision, declarative rules scores notably worse than the two other contenders. Decidability expresses how easy it is to decide if we have a valid gesture or not. This is medium for template matchers because they mostly return a score value with an unbounded range. This implies that one needs to find an individual threshold for each gesture. With machine learning, it returns a probability value between 0 and 1 which makes it much easier to decide. However, one still needs some kind of threshold. With declarative rules, it is very simple: it matches the rules or it does not, there is no in-between.

When one builds a modularized system, adding a single gesture does not cause many changes to the system. This is clearly the case with template matchers as one just needs to add a template for this gesture without changing the templates of any other gesture. For machine learning algorithms, one can argue that this is harder to achieve since access to the complete training data is required. This data is typically not provided since only the “learned model” is required to classify gestures, which makes it impossible to extend the system. Adding a gesture to a system with declarative rules is as simple as just adding the rules for this gesture, without changing anything to the other rules.

In the ideal case, one wants a system where one has no need for start and end points, as little training as possible, obtain high precision, high decidability and high modularization. As one can see, there is no single algorithm that satisfies all our requirements. However, we think that by combining declarative rules with a template matcher, we can get very close to achieving this goal.

The declarative rules ensure that we do not need fixed start and end points. Neither of them needs extensive training data which is one of the requirements. We estimate that the precision will be high as the template matcher will check the possible matches generated by the declarative rules. The modularization will be high since both subsystems support this property. We estimate that the decidability will not be as high as with declarative rules, but we hope that this will be a small price to pay for the enhanced precision.

Chapter 3

Proposed solution

This chapter explains our proposed solution in a general way. First we will show our gesture recognition process (sections 3.1 to 3.4) and then discuss our tools for gestures for home automation (sections 3.5 to 3.5.2). Implementation details are discussed in chapter 4.

3.1 General architecture

We used a Microsoft Kinect camera which is connected to a PC via a USB connection. Using the Microsoft Kinect SDK, we obtain the coordinates of the different persons and their body parts. This gives us a continuous stream of coordinates of body parts (with an identifier of the corresponding person). What we want to achieve is that we can accurately recognize gestures from this continuous stream of coordinates. To achieve this, we propose a two-phase filtering where we do a first filtering with a declarative rule system. This system on its own is not accurate enough. Therefore, we have a verification by introducing a second filtering based on a template matcher. A gesture is only correctly identified if the template matcher confirms the gesture recognized by the declarative rule system.

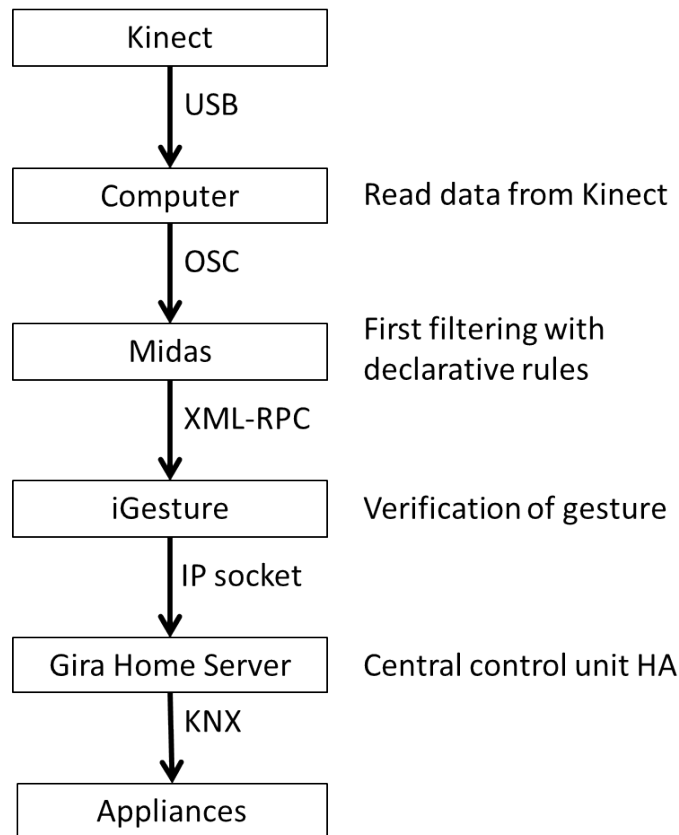


Figure 3.1: Architecture of our solution

Figure 3.1 provides an overview of the structure of our solution. The data received from the camera is sent to a first system where it is filtered by declarative rules. Afterwards, these potential matches are verified by a second system using a template matcher. If the gesture is correctly identified, we will send the corresponding command to a central control unit which communicates with the different appliances.

3.2 Skeleton tracking

There are different cameras out there that can be connected to a PC. When a camera is connected to a computer, one mostly receives a video stream. To perform gesture recognition, we need to be able to recognize the persons and their body parts from this video stream. This is called skeleton tracking. Some cameras offer this functionality as part of their SDK. When this is not the case, one can always implement an existing algorithm for the recognition of people and their body parts from a continuous video stream. As implementing such an algorithm falls out of the scope of this thesis, we

decided to go for a camera that already offers skeleton tracking with their SDK.



Figure 3.2: kinect skeleton tracking example

As this is a research thesis and not an industry project, the price of the camera also plays an important role. Therefore, we decided to use a camera that combines an affordable price with an SDK that offers skeleton tracking which lead us to choose Microsoft's Kinect camera. The camera is connected to our computer via a USB connection. Every time the SDK has processed a new image frame, a function is called where we can extract the coordinates of all the body parts. An example of the skeleton tracking of the Microsoft Kinect can be seen in Figure 3.2 ¹. On the left hand side, one can see the green lines with dots which represent the joints of the skeleton. On the right hand side, one can see a depth view.

3.3 Segmentation using declarative rules

What we get as input is a continuous stream of data. If one wants to implement gesture recognition for real use cases, the first thing a developer needs to deal with is gesture spotting. This is the extraction of potential gesture matches from a continuous stream of data. We achieve this by describing gestures in declarative rules. These rules express temporal or spatial constraints.

¹Kinect Explorer: <http://blogs.msdn.com/b/csharpfaq/archive/2012/02/06/start-coding-for-the-kinect.aspx>, [24-May-2012]

To make this more concrete, we will illustrate this with the example of a Z gesture. In Figure 3.3, one can see a plot of a part of a continuous stream of coordinates, expressed in mm. The full blue line represents the x coordinates and the dotted red line represents the y coordinates. The stream contains the Z gesture where one starts at the upper left corner of Z and ends at the lower right corner of Z.

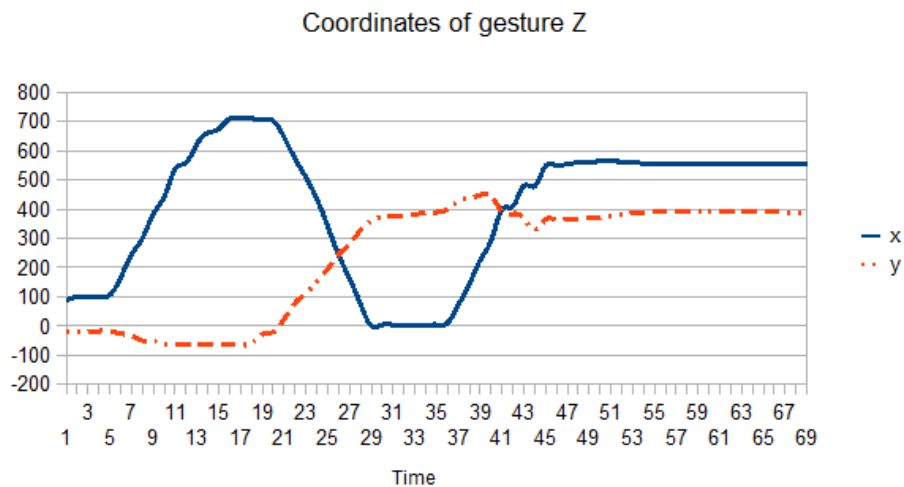


Figure 3.3: Chart showing coordinates of a Z gesture.

As one can see, our gesture starts around time 6 (expressed in frames). We move our hand to the right while remaining more or less leveled. This can be seen on the chart as the x coordinate rises from 100 to 700 while the y coordinate remains more or less the same. The change in position starts around time 6 and ends around time 17. This is a first characteristic property of the gesture that must be encoded in a declarative rule. The rule will encompass that the x coordinate must raise with around 600 mm while the y coordinate must stay more or less leveled. It can be extended with a time constraint that this part of the gesture should have a certain minimum and/or maximum duration in ms.

This first movement is followed by a movement from the upper right of Z to the lower left of Z. One can see this on the chart as the x coordinate declines while at the same time, the y coordinate increases. Note that for technical reasons, going down implicates an increase in the y coordinate. This will be explained later on in this thesis. The movement starts around frame 20 and ends around frame 30. This is a second characteristic property that needs to be encoded in a declarative rule. It must encompass that the

x coordinate must drop around 700 mm while the y coordinate must incline with around 400 mm.

The last movement is a horizontal one where the x coordinate increases while the y coordinate remains more or less leveled. It starts around frame 36 and ends around frame 45. This movement can be encoded in a similar way as the first movement of this gesture. In total, one needs three spatial constraints to describe the Z gesture. It can be further optimized by adding time constraints. These time constraints may apply on the duration of the entire gesture or on the duration of a subpart. A matching engine will analyze the continuous stream of data and search for patterns that match with one of the defined sets of declarative rules.

A potential implementation of declarative rules with matching engine is Mudra. It has been developed at the VUB and written in the programming language C. As the SDK of the camera has an interface in C#, one needs to find a way to send the continuous stream of data from the SDK in C# to the complex event processing engine in C. We have achieved this by using the OSC-protocol. OSC stands for open sound control ². It is a protocol developed to transmit continuous streams of data, which makes it suitable for our case. The matching engine will process the incoming stream of data and do gesture spotting. However, one needs to understand that these potential matches do not offer a high level of accuracy. Therefore, in addition a verification step is needed.

3.4 Verification: template matcher

Potential matches found by the gesture spotting engine, need to be verified because they have a low accuracy. This verification can be done by using another algorithm to check if we get the same result. As these potential matches have a finite length, and therefore a clear begin and end point, we can choose from a wide range of algorithms to perform this verification. Possibilities include template matching algorithms or machine learning algorithms.

To keep our solution as general as possible, we decided to extend the open source gesture recognition framework iGesture [17]³. Therefore, the user can extend the framework with the gesture recognition algorithm of his choice. As we preferred an algorithm that does not require training, we opted for a template matching algorithm. We choose to implement one of the most well-known template matching algorithms, Dynamic Time Warping (DTW) [5].

²OSC: <http://opensoundcontrol.org/>, [24-May-2012]

³iGesture: <http://www.igesture.org/>, [24-May-2012]

By choosing for a template matching algorithm, we must be able to define templates in the gesture recognition framework. Therefore, we also needed to extend it with the input modality of the Kinect Camera. To keep our solution as general as possible, we opted to use OSC as protocol for transferring the coordinates. This is the same protocol that we use for transferring the stream of coordinates from the Kinect SDK to Mudra, our segmentation tool. If one wants to use a different camera than the Microsoft Kinect, he would only need to create an OSC server to send the coordinates.

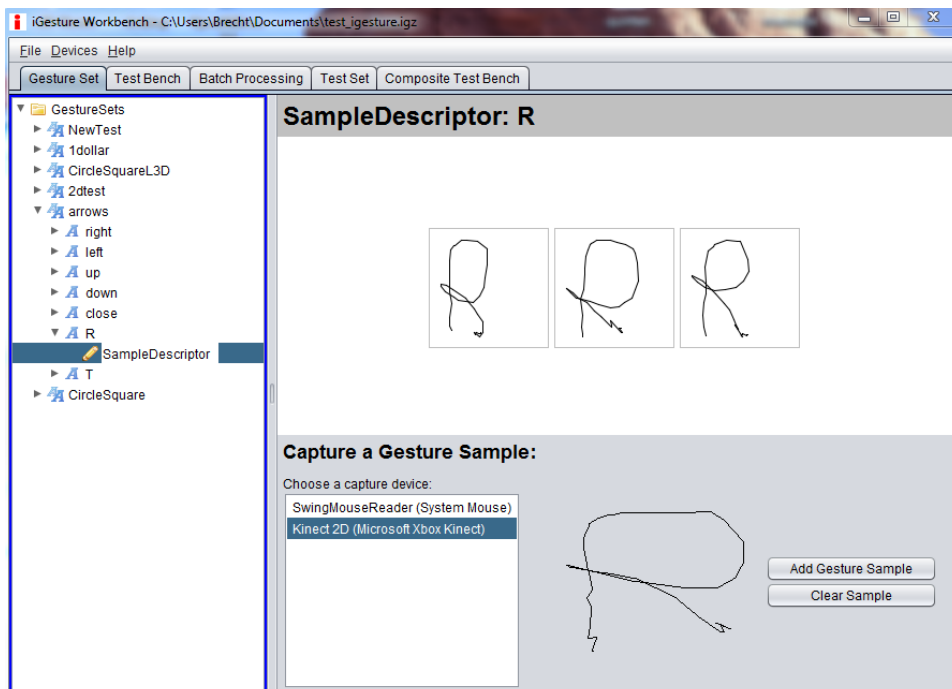


Figure 3.4: Adding sample to gesture

If one wants to add samples to implement a new gesture, the first thing one needs to do in iGesture is to create a gesture set. We called it arrows in the example one can see in Figure 3.4. In this gesture set, one can define all the gestures he wants, like R in our example. Each gesture consists of one or more samples. In the Figure 3.4, we have already added 3 samples to the gesture R and we are adding a fourth sample. An overview of all the gestures in a gesture set is generated by selecting the gesture set, as seen in Figure 3.5.

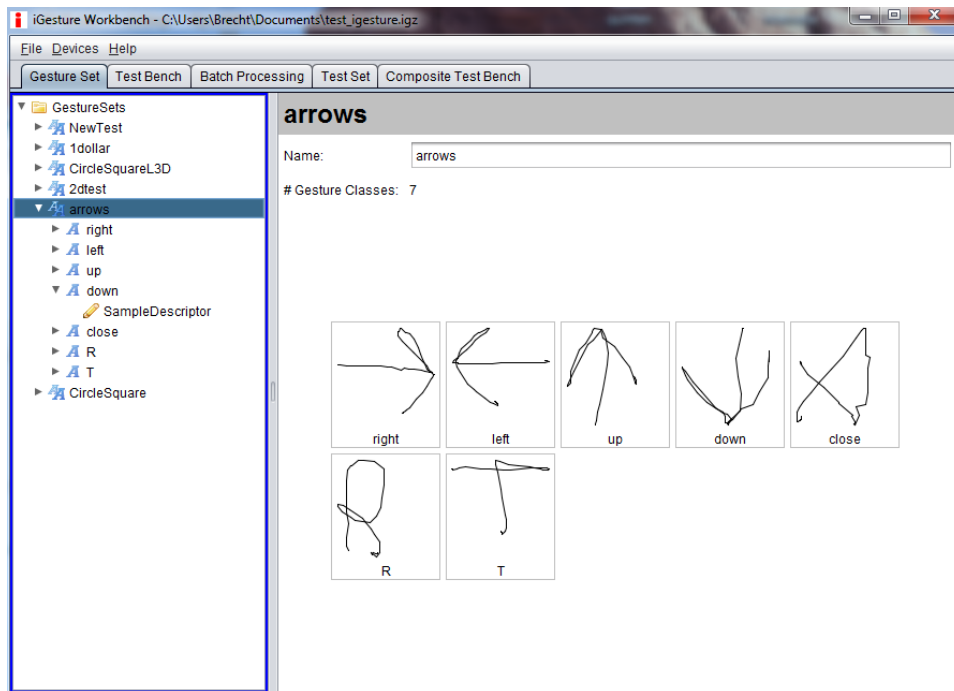


Figure 3.5: Overview of the gestures

The gesture recognition framework (iGesture) is written in Java while our declarative rules system (Mudra) is written in C. Mudra needs to send potential matches to iGesture. This is clearly not a continuous stream of data. Therefore, we decided to use XML-RPC for the communication between these two entities. By using XML-RPC, we also make it possible to easily switch to another gesture recognition framework if someone wants to do this. One just has to make sure that the new gesture recognition framework has an XML-RPC library that is listening on a certain port.

Mudra was extended for this thesis to export the following procedures: initialise, recognise and remove, as shown below.

```
//Initialises a recogniser in the iGesture framework via
//XML-RPC.
```

```
//Format: (igesture:initialise <name> <algorithm>
           <gesture-set >)
```

```
//Processes a number of MFacts through the registered
//recogniser in the iGesture framework via XML-RPC.
```

```
//Format: (igesture:recognise <name> <facts> <threshold>
           <max-matches >)
```

```
//Removes a recogniser in the iGesture framework via
//XML-RPC.
//Format: (igesture:remove <name>)
```

The first procedure will initialize the algorithm one wants to use. The second procedure will compare a sample with a defined set of gestures and do the recognition. Finally, remove will remove an instance of the algorithm. We use this initialise-recognise-remove cycle as some algorithms might need training. This training can be done once we are in the `initialiseAlgorithm` procedure and one can from then on use the trained instance for all the samples.

We added functionality so one can automatically generate the declarative rules that correspond with a given gesture. First, one does a gesture as shown in Figure 3.6.

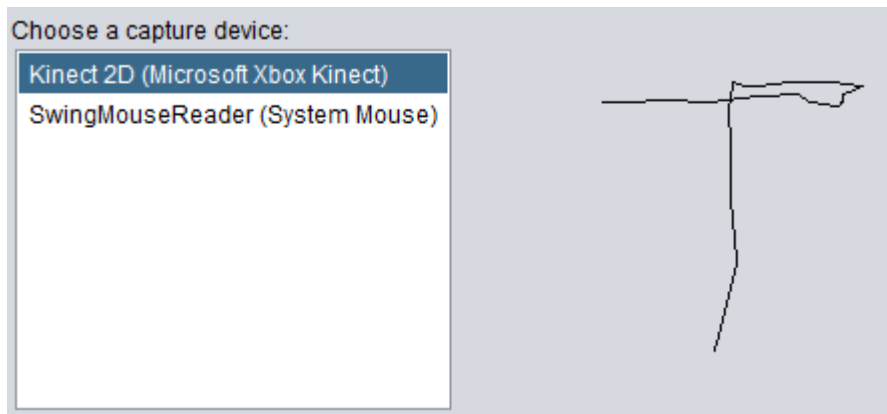


Figure 3.6: Initial gesture

Afterwards, one can simple select the different characteristic points of the gesture using the left mouse button. The characteristic points are indicated with a circle. For our example, we have four characteristic points which can be seen in Figure 3.7.

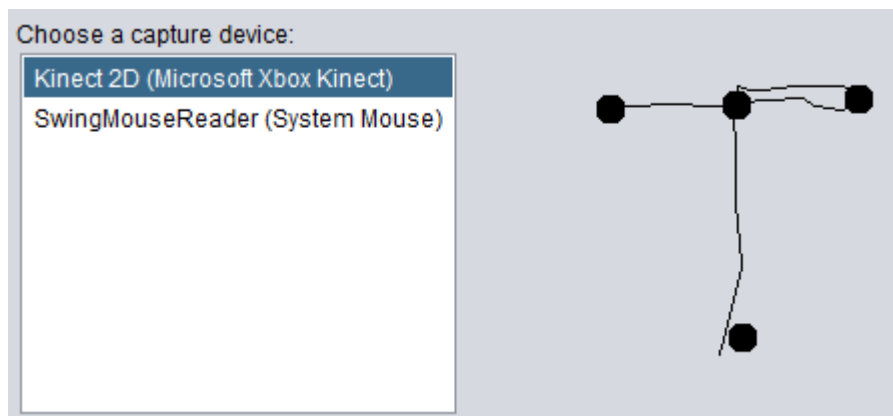


Figure 3.7: Select characteristic points

To generate the declarative rules, one simply clicks with the right mouse button. A frame will appear with the corresponding code. The frame with code for our example can be seen at Figure 3.8.

```

Midas Code
{set-default-timespan "Joint" 5000)
(deftemplate zGesture (slot joint))
(set-default-timespan "zGesture" 5000)
(defrule zGesture
  (Joint (joint 15) (x ?x1) (y ?y1) (on ?on1))
  (not (and (zGesture (on ?onz)) (test (time:within ?on1 ?onz 2000))))
  (Joint (joint 15) (x ?x2) (y ?y2) (on ?on2))
  (test (space:within-radius-translate ?x1 ?y1 ?x2 ?y2 433 -16 150))
  (test (time:before ?on1 ?on2 1000))
  (Joint (joint 15) (x ?x3) (y ?y3) (on ?on3))
  (test (space:within-radius-translate ?x2 ?y2 ?x3 ?y3 -213 10 150))
  (test (time:before ?on2 ?on3 1000))
  (Joint (joint 15) (x ?x4) (y ?y4) (on ?on4))
  (test (space:within-radius-translate ?x3 ?y3 ?x4 ?y4 11 369 150))
  (test (time:before ?on3 ?on4 1000))
=>
  (printout t "zGesture" crlf)
  (igesture:recognise "dtw" (select-facts-between "Joint" ?on1 ?on4) 0.4 7)
  (assert (zGesture (joint 15) (on ?on1))))

```

Figure 3.8: Frame with declarative rules

At the moment, one can do only single handed gestures. This limitation has been introduced for complexity reasons: the support of multiple hands is technically possible but it makes the automatic generation of declarative

rules substantially more complex and can therefore be seen as a potential future extension.

3.5 Home automation

Once a gesture has been recognized, a corresponding action needs to be executed. As mentioned before, we target the context of home automation.

In general, a home automation system has a central control unit which is responsible for the communication with the different controllable appliances. A possible central control unit is the Gira Homeserver⁴. We can send commands to this control unit via an IP socket. These commands are just strings with no mandatory format. For the sake of readability, we encourage to use an OSC-format for the strings. This means that a string has the following pattern:

```
/room/appliance/status
```

When one wants to turn on a light in the living room, he will send a string like `/living/light1/on` to the central control unit. Based on the incoming string, the central control unit will send the necessary commands to the different appliances using the KNX protocol. This protocol is a standard and widely used in home automation. To make it more concrete with an example: one can for imagine a multimedia room where the user does a specific gesture. The gesture is recognized and the corresponding command is sent to the central control unit. This unit will send commands to dim the lights, close the curtains and turn on the beamer, like shown in Figure 3.9. This protocol specification was done in collaboration with a Flemish company called Pattyn Domotica⁵.

⁴Gira Homeserver, http://www.gira.com/nl_BE/produkte/homeserver.html, [25-May-2012]

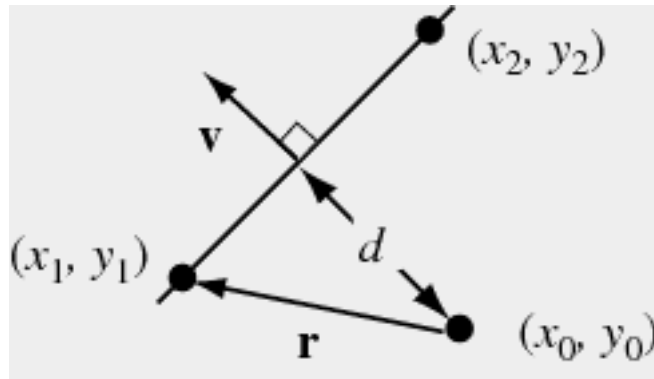
⁵Pattyn Domotica, <http://www.pattyn-domotica.be>, [25-May-2012]



Figure 3.9: An example of a multimediaroom

3.5.1 Modeling room

Location awareness is achieved by the limited range of the Kinect camera. This allows us to infer whether a user is in the living room or in the kitchen. However, there are still multiple appliances in the room that require interaction. Imagine that one has multiple lights in the living room. If we want to control them independently from each other with gestures, we have two possibilities. We can define different gestures for each light unit or we need to combine our gesture with pointing into the direction of a specific light unit. To know in which direction a person is pointing, we can define the line that goes through the person's shoulder and wrist. With the equation of this line, we can calculate the distance of all objects to this line.



Take (x_1, y_1) as the coordinates of the shoulder and (x_2, y_2) as the coordinates of the wrist. We can draw a unique line through these two points. Define vector v as the vector perpendicular of this line and vector r as the vector between (x_0, y_0) and the beginning of the line. Calculating the distance d from any point (x_0, y_0) to this line, is done by projecting the vector r onto vector v . As a result, we get the following formula:

$$d = |v \cdot r| = \frac{|(x_2 - x_1) \cdot (y_1 - y_0) - (x_1 - x_0) \cdot (y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

By ordering all relevant objects on distance, we can find the object with the nearest distance. In our example where we want to control different light units, we can take all the light units and calculate their distance to the imaginary line from our shoulder to our wrist. The light unit with the smallest distance to this line is the correct light unit.

To know where all the controllable elements are located in a room, we need to have a model of the room. Each controllable element, for example a light unit or a television, is represented by an object. It contains a location and a dictionary with a mapping from the gestures to the corresponding commands. To improve the usability, we also created a visualization tool in order that one can see the different elements, as shown in Figure 3.10.

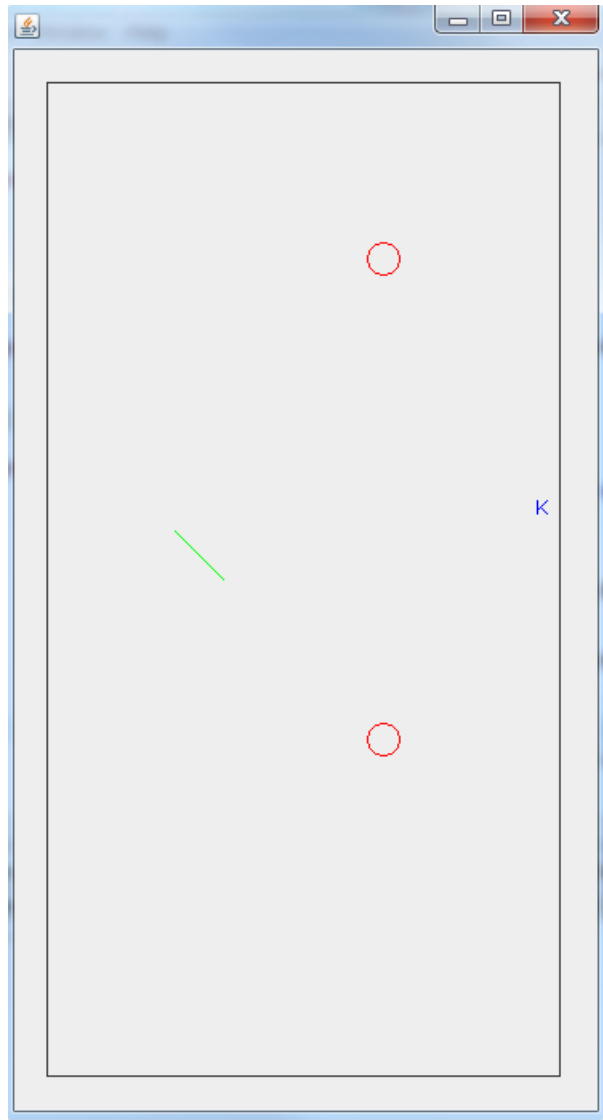


Figure 3.10: A screenshot of our pointing application

The rectangle represents the walls of the room. The location of the Kinect is represented by the letter K. Each controllable appliance is represented by a circle at its corresponding location. When a person stretches his arm, a line appears to indicate the direction to which the person is pointing. As one can see in the example of Figure 3.10, the person is pointing to the lowest appliance that can be controlled.

Target estimation Using this room modeling tool, we can now estimate which appliance is targeted by the user. Using the results from Mudra, the iGesture service and the KNX protocol definition, we have a feature complete

software stack to achieve gesture-based home automation as illustrated in Figure 3.1.

3.5.2 Gesture set

When defining gestures, one often needs to find a balance between a gesture that is large enough so the camera can detect it but small enough so one does not disturb other people. When one defines a simple gesture like moving a hand upwards, one needs to bring into account that this gesture will likely be a sub-gesture of other gestures. To make it more concrete with an example, one can clearly see that drawing a vertical upwards line is a sub-gesture of drawing a P in the air. This makes the definition of gestures harder as one has to manually encode all the possible sub-gesture combinations.

This problem becomes even more substantial when we start talking about hovering. Imagine one is sitting in his multimedia room and has an overview of all available films on a screen. Most intuitively, one selects a film by moving the arm left/right/up/down until he selected the wanted movie. This is called hovering. To start the film, one obviously has to perform a gesture. Performing this gesture will inevitably lead to changing the selected movie as one is moving his arm in a direction. This is not workable as one will always start a movie he did not want to see.

There are multiple solutions for this problem. The first solution would be that one uses voice commands to activate and deactivate the hovering. When one is in a hovering state, the gesture recognition will be disabled. If the right film is selected, one deactivates the hovering and start the movie with a gesture. A second solution would be to use the left arm for hovering and the right arm for gestures.

Due to time constraints, we limit ourself to a basic set of deictic and iconic gestures that allow us to evaluate the feasibility of our approach. It also allows us to precisely define problematic scenarios and offer room for software-based improvements as mentioned in section 6.

3.5.3 Conclusion

We created a feature complete software stack to achieve gesture-based home automation. The movements are captured by a 3D camera which is in our case Microsoft's Kinect. To do the gesture spotting, we use a declarative rule approach. A possible implementation of this approach is Mudra which is what we used in this setup. As the Kinect SDK has a C# interface and Mudra has a C interface, we transferred the continuous stream of coordinates from the Kinect SDK to Mudra via the OSC protocol. The gestures spotted by Mudra lack accuracy, therefore we foresee a verification phase.

The verification is achieved by checking the spotted gesture of Mudra with a template matcher. For the sake of flexibility, we opted to use an open source gesture recognition framework so one can add his favorite gesture recognition algorithm. Our choice fell on the iGesture framework. To proof the extensibility of the framework, we extended it with one of the most well-known template matching algorithms: Dynamic Time Warping. As iGesture is written in Java while Mudra is written in C, we used XML-RPC to transfer the spotted gesture from Mudra to iGesture. When the template matcher confirms the spotted gesture, we have correctly identified a gesture.

Afterwards, one needs to execute the actions that correspond to the recognized gesture. If one has multiple light units in one room and one wants to control them individually with the same gesture, the software needs to know in which direction one is pointing to find the nearest relevant light unit. Therefore, we created an application where one can model the room and add the different units with their corresponding location. When the nearest unit is found, the corresponding command is sent to the central control unit which will do the communication with the necessary appliances using the KNX protocol.

For an enhanced user experience, we also extended iGesture with the possibility to automatically generate the declarative rules. One just needs to do the gesture and indicate the characteristic points with the mouse.

Chapter 4

Implementation

This chapter will discuss in detail our implementation of the proposed solution.

4.1 Skeleton tracking

To read data from the Kinect camera, there are two possibilities: the official Kinect SDK from Microsoft¹ and the semi-open source projects OpenNI, NITE and OSCelton². The official SDK has interfaces in C# and Visual Basic. However, it supports only Windows as operating system. The official SDK was only released in autumn 2012. Before that, there was only the OpenNI solution.

OSCelton reads the data from the Kinect and sends it to a specified port of an IP address. The protocol used is OSC as one could have guessed from the name of the application. Because OSCelton exists already for a few months, most applications use this library. However, testing revealed that the Kinect SDK of Microsoft has a higher precision. Therefore, we decided to use the official SDK. For ease of use, a OSCelton compatible solution was made for the Microsoft Kinect SDK. We implemented our own OSC server in C# and made it open source at <https://github.com/Zillode/OSCeleton-KinectSDK>. Because of this, the rest of our setup does not know whether the data comes from the official SDK or OSCelton. Developers can now easily switch between these two options without making any changes to the rest of their code. This possibility for switching is needed because the official SDK of Microsoft only supports the Windows operating system. If one has another operating system, OSCelton will be the only option.

¹<http://www.microsoft.com/en-us/kinectforwindows/>

²<https://github.com/Sensebloom/OSCeleton>

An OSC message starts with “/Joint” and has the following fields: the id of the skeleton joint, the id of the corresponding user, three floats representing the coordinate in mm and an integer representing the time from start in milliseconds. A possible example of joint message would be:

```
/Joint ,1 ,15 ,256.56 ,95.53 ,867.71 ,16080
```

The id of the person is 1 and we track a joint with id 15. The coordinates of this joint are (256.56 mm, 95.53 mm, 867.71 mm) and the corresponding time is 16.08 seconds after we started recording. In the table below, one can see the different joint points with the corresponding id.

Id	Microsoft SDK	OSCCelton	Id	Microsoft SDK	OSCCelton
1	Head	head	13	ElbowRight	r_elbow
2	ShoulderCenter	neck	14	WristRight	r_wrist
3	Spine	torso	15	HandRight	r_hand
4	HipCenter	waist	16	/	r_finger
5	/	r_collar	17	HipLeft	l_hip
6	ShoulderLeft	l_shoulder	18	KneeLeft	l_knee
7	ElbowLeft	l_elbow	19	AnkleLeft	l_ankle
8	WristLeft	l_wrist	20	FootLeft	l_foot
9	HandLeft	l_hand	21	HipRight	r_hip
10	/	l_finger	22	KneeRight	r_knee
11	/	r_collar	23	AnkleRight	r_ankle
12	ShoulderRight	r_shoulder	24	FootRight	r_foot

To enhance debugging, we extended the Microsoft SDK with the recording of the coordinates. The coordinates are stored in a CSV file. This CSV output was implemented to be compatible with the Mudra CSV system, which allows developers to easily test recorded gestures without having to perform the movement in front of the camera.

4.2 Segmentation: declarative rules

We use a declarative rules approach, namely Mudra, to perform gesture spotting. Mudra takes a stream of OSC messages as input. The message contains the id of the joint point, the id of the corresponding user, a coordinate and a number representing the time passed since one started recording.

To make the reader familiar with Mudra, we will discuss its syntax with an example. Mudra uses declarative rules for expressing the control points and the corresponding constraints. This is the same gesture as we discussed in our proposed solution.


```

1 (deftemplate zGesture (slot joint))
2 (set-default-timespan "zGesture" 5000)
3 (defrule zGesture
4 (Joint (joint 15) (x ?x1) (y ?y1) (on ?on1))
5 (Joint (joint 15) (x ?x2) (y ?y2) (on ?on2))
6 (test (time:before ?on1 ?on2 2000))
7 (test (space:within-radius-translate ?x1 ?y1 ?x2 ?y2
8                                         600 0 150))
9 (Joint (joint 15) (x ?x3) (y ?y3) (on ?on3))
10 (test (time:before ?on2 ?on3 2000))
11 (test (space:within-radius-translate ?x1 ?y1 ?x3 ?y3
12                                         -100 400 150))
13 (Joint (joint 15) (x ?x4) (y ?y4) (on ?on4))
14 (test (time:before ?on3 ?on4 2000))
15 (test (space:within-radius-translate ?x1 ?y1 ?x4 ?y4
16                                         550 400 150))
17 =>
18 (assert (zGesture (joint 15) (on ?on1))))

```

In the first line, we define the name of our gesture. Afterwards, we define the maximum time-span of our gesture, in this case 5000 milliseconds (5 seconds). One needs to find a balance between a long enough timespan to be able to complete the gesture, but on the other hand, keep the timespan as short as possible for better performance. The timespan is linear with the amount of data that Mudra needs to take in consideration. Now, we can start with the definition of our control points.

In line 4 and 5, we define our first two control points. These control points have a joint, a location and a time-indication. The joint is linked to a joint of the skeleton of the person. In this case, joint 15 corresponds with the right hand. A joint has also a coordinate (with x and y) which we bind to the variables x1 and y1. Next to this, a joint also has a time in milliseconds, which is bound to the variable on1.

In line 6, we define a time constraint that the first control point should happen before the second control point and the maximum time difference between these control points is 2000 milliseconds (= 2 seconds). In the next line, we define a spatial constraint between these two control points. We define that the difference in x-coordinates should be 600mm and the difference in y-coordinates should be 0mm. However, in reality, gestures done by the user are not that exact. Therefore, we foresee a margin so that the user should pass this second control point within a radius of 150mm. This sequence of defining control point, add time constraint and add spatial constraint is repeated for the rest of the control points. This happens in line 7

to 16.

In rule 17 and 18, we declare that if all the conditions are met, the gesture should be asserted.

4.3 Second filtering: template matcher

We integrated iGesture within Mudra to apply a verification step. The verification happens with the use of a template matching algorithm. Therefore, we need to define templates in iGesture with the Kinect as input modality. As iGesture did not offer support for the Kinect camera, we had to add support for it ourselves.

4.3.1 Extending iGesture for camera-based sample recording

Spatial scaling This is realized in iGesture by creating a new type of input, OSC input. We have added an OSC listener which listens on a predefined port for OSC messages. Once received, these messages will be processed and the coordinates will be displayed on the screen. A problem faced during the implementation was the scaling of the display. When one does a gesture, we cannot predict if the person will do a small gesture or a large gesture. Therefore, a fixed scale is not appropriate. If one does a large gesture, it will not fit on the screen. On the other end, if one does a small gesture, one will barely be able to see the gesture on the screen. As a solution, we have chosen to use an automatic scaling. We start with an initial scale, ideal for small gestures, and automatically zoom out whenever needed.

Segmenting samples from continuous input Another problem that arises is when recording samples of an endless stream of coordinates is to define the a start and end point. To solve this problem, we added voice (de)activation on the Microsoft Kinect SDK. By using voice commands like “camera on” or “camera off”, one can control the streaming of coordinates. This is achieved with the help of the Microsoft Speech SDK. In Figure 4.1, one can see a state machine explaining how to control the Kinect camera with voice commands.

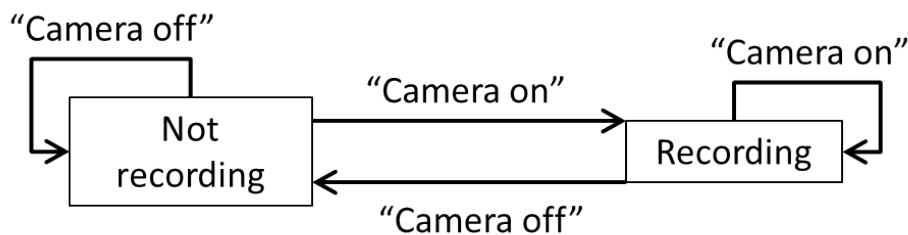


Figure 4.1: State machine for voice control Kinect

4.3.2 Semi-automated rule generation

These two additions to iGesture allows average users to easily add new samples to the system using a camera-based input modality. Next, we show how basic rules are generated from these newly defined samples. First, the user needs to select the characteristic points of the gesture. This is done by selecting them with the left mouse button. Every time the user presses the left mouse button, we retrieve the coordinate and add it to an array of points. When the user has selected all the characteristic points, he can simply generate the code by pressing the right mouse button. An algorithm will loop over the array of points and generate a string which contains the declarative rules. This string is showed in a new frame that pops up. Figure 4.2 shows a state diagram of this procedure.

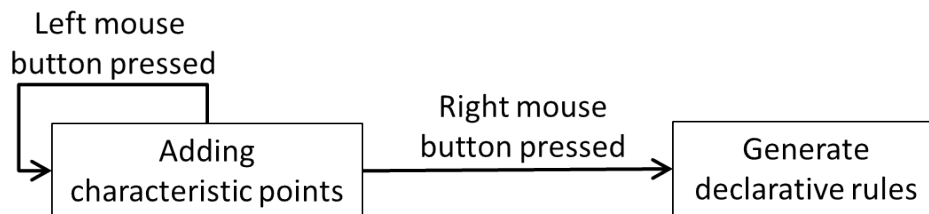


Figure 4.2: State machine for generating declarative rules

The algorithm is quite straight forward. First, we have the following lines of fixed text:

```
1 (deftemplate zGesture (slot joint))
2 (set-default-timespan "zGesture" 5000)
3 (defrule zGesture
4 (Joint (joint 15) (x ?x1) (y ?y1) (on ?on1))
5 (not (and (zGesture (on ?onz))
6 (test (time:within ?on1 ?onz 2000))))
```

These lines mean that we define a gesture with the name zGesture and this gesture has a default maximum timespan of 5 seconds. We define a first point of the gesture and code that there must be a default minimum timespan of 2 seconds between two occurrences of this gesture.

Then, we use a for-loop to loop over the characteristic points. For each point, we calculate the distances (in x and y) between this point and the previous point in the array. The distances are encoded into the following temporal constraints (for each point):

```
1 (Joint (joint 15) (x ?x(i+1)) (y ?y(i+1)) (on ?on(i+1)))
2 (test (space:within-radius-translate ?x(i) ?y(i) ?x(i+1)
3 ?y(i+1) diffX diffY 150))
```

```
4 (test (time:before ?on(i) ?on(i+1) 1000))
```

In the first line, we define a new point (the (i) and (i+1) must be replaced with the index in the for-loop). In the second line, we say that the distance between the previous point and the current point should be the distance that we calculated with a maximum allowed variation of 150 mm. Finally, we code that the time difference between these two points should be 1 second or less.

After the for-loop is completed, we need to add the following final statements:

```
1 =>
2 (igesture:recognise "dtw" (select-facts-between "Joint"
3                               ?on1 ?on(number-of-points)) 0.4 3)
4 (assert (zGesture (joint 15) (on ?on1)))
```

We define that found gestures needs to be sent to iGesture. The algorithm used is DTW and we need to send all data between the first characteristic point and the last characteristic point. The results of the template matcher needs to be filtered with a minimum accuracy of 0.4 (40%) and we want to have the best 3 results. Finally, the gesture should be stored as a fact in the fact base of Mudra.

4.3.3 DTW for hand trajectory classification in iGesture

As iGesture did not provide the classifier we required, we implemented Dynamic Time Warping. Section 2.2.2 already explains how this algorithm works and we did a straight forward implementation for iGesture.

4.3.4 Service oriented iGesture

When Mudra has spotted a gesture, it will send the gesture to a gesture recognition framework for verification. This communication with the framework happens with XML-RPC so that we are independent of framework used.

Mudra will make the following calls to the iGesture recognition service:

1. Instantiate the algorithm one wants to use
Mudra will call the function `initialiseAlgorithm` with an XML-RPC struct as parameter. This struct contains the fields `algorithm` (a string containing the name of the algorithm one wants to use), `gestureset` (the name of the gestureset that needs to be used for the recognition), `threshold` (a double representing the minimum accuracy required) and `name` (the name one wants to give to this instance).

2. Perform the recognition

Every time Mudra finds a potential match, it will call the `recognise2D` or `recognise3D` function. This will do the recognition with a previous instantiated algorithm of step 1. The recognize function takes a struct and an array as parameters. The struct has two fields: name (the name of the instantiated algorithm one wants to use) and threshold (if one wants a different accuracy then the default one specified in step 1). The array contains the points of the gesture. Each point is a struct with four fields: X, Y, Z (floats representing the coordinate) and ON (which represents the time corresponding with this point).

3. Delete the instance of the algorithm

When one wants to close the application, we need to delete our instantiated algorithms. This is done by calling the function `removeAlgorithm` with the name of the instantiated algorithm as parameter.

Each template matching system that supports these functions can be used in combination with Mudra. We have extended iGesture with the functionality described above. It has an XML-RPC listener that will listen on a certain port for incoming calls for one of these three functions. If the `initialiseAlgorithm` function is called, it will initiate the algorithm in iGesture and store it in a dictionary with the corresponding name. When the `recognise2D` function is called, it will take the corresponding algorithm from the dictionary with initiated gesture recognition algorithms and do the recognition with the input data. Afterwards, the results are filtered on accuracy and the maximum amount of results allowed. Finally, when Mudra calls the `removeAlgorithm` function, the initiated algorithm is removed from the dictionary of initiated algorithms.

4.4 Home automation

Once a template matcher has confirmed the gesture, a corresponding action needs to be taken. Some gestures are unique and therefore, one does not need to know in which direction a person is pointing. If this is the case, we can simply execute the corresponding action by sending the right commands to the central control unit. This is achieved by sending the command to a buffer. A separate thread is responsible for popping the messages from the buffer and sending them to the central home automation control unit. This control unit is accessible via an IP connection.

However, it might be the case that your gesture is not unique. This is for instance the case when one has multiple light units that one can control separately but when there is only one gesture available to turn on a light. By checking in which direction the person is pointing while performing the gesture, the application can know which light the person wants to turn on. One can achieve this by taking the coordinates of the shoulder and wrist of the arm and calculate the line that goes through it. The distance from each object to this line can easily be calculated with the formula explained in the previous chapter. This assumes that you know where all your controllable units are located. To achieve this, we have created a model to represent a room.

This model consists of the following objects:

- **Element**
This represents a controllable element like a light unit. Each element has a location (in x, y) and a hashtable which contains the names of all relevant gestures for this appliance with their corresponding commands.
- **Person**
This represents a person. It has an OSC listener to receive the coordinates of the person. It has one function which is called `stretchedArm`. It will look which arm is stretched and return a hash set with the coordinates of the shoulder / hand of the stretched arm. If neither of the two arms is stretched, an exception is thrown.
- **Room**
This represents the room. A room has a width and length, expressed in cm. A room also has a Person and an arraylist of Elements. A room also holds the position of the Kinect camera. We need to know this because the coordinates we receive from the Kinect are relative to the Kinect. If one knows the exact coordinate of the Kinect, one can easily calculate the absolute position a person. Finally, the most import

function is `findPoint`. It will look in which direction the person is pointing and return the corresponding element. If no element is found near, an exception is thrown.

When one has done a gesture and the corresponding element has been found, the element is queried for the command which corresponds with the given gesture. This command is then added to the buffer with commands that need to be executed.

Chapter 5

Validation

We tested our setup with eight different gestures. It is a mix of commonly used gesture and gestures that are hard to detect with a rules approach only. The commonly used gestures are swipe left and swipe right. These do not need further explanation as everyone knows how they look like.

A first set gestures that are harder to detect are the letter T and arrow up. In Figure 5.1 one can see these two gestures on the left, with their characteristic points on the right.

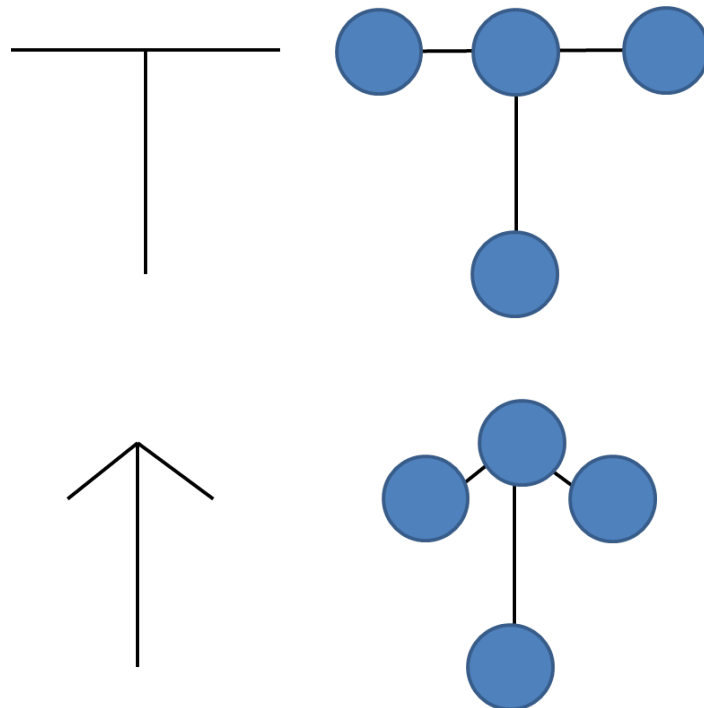


Figure 5.1: Characteristic points

As one can see, when one is doing a T or an arrow up, one will likely also trigger the other gesture. Using smaller circles is not a good solution as this creates the problem that it will not trigger when the gesture performed by a user is not exactly the same, although it is conceived by the user to be the same. Therefore, we use larger circles and a validation step with a template matcher to define which one of those two gestures it is. A second set of gestures that we use faces similar problems. It contains square, circle and triangle gestures.

We used 5 persons to test our setup. They all have no experience with doing gestures for a camera. Neither of them has a mobile phone with a touch interface. Their experience with computers varies from almost nothing (just doing some surfing) to using it all day long in a professional context. The results of the test are shown below:

	Left swipe	Right swipe	T	Arrow up	Square	Circle	Triangle
Recognized by Mudra (in %)	100	100	100	80	80	60	80
Number false positives per gesture by Mudra	0	0	0.8	0.6	0.2	1.4	1
Recognized by Mudra + DTW (in %)	100	100	100	80	80	40	40
Number false positives per gesture by Mudra + DTW	0	0	0	0	0	0.2	0.4

As one can see, we get a significant decline in the amount of wrongly identified gestures by doing verification with DTW. Like discussed earlier, when one does the gesture T, he will likely also trigger the gesture arrow up which is confirmed by our testing. By doing verification with a template matcher, we were able to filter out the unintended triggering of the arrow up.

However, this approach is not completely flawless. When doing a circle or triangle, we achieve a decline in wrongly identified gesture but we were not able to completely eliminate it. This is caused because template matching algorithms do not always give perfect results. To test this further, we also tested the template matcher with another input modality: a traditional mouse. We created a new gesture set with the mouse and tested it afterwards.

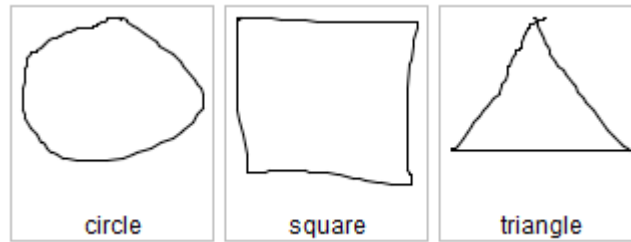


Figure 5.2: Gesture set created with mouse

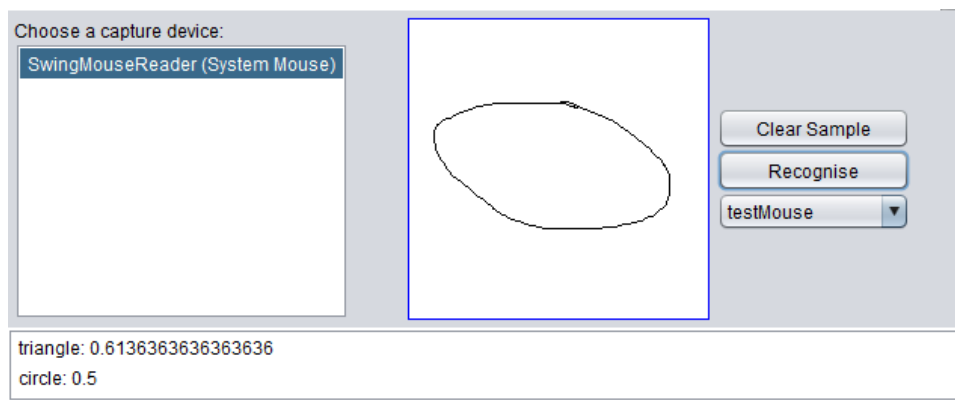


Figure 5.3: Testing with mouse of gesture set

As one can see in Figure 5.3, this is not the result one likes to see. We also tested it with another gesture recognition algorithm (signature [17] ¹) but it gave similar results. We think this problem can be solved by using advanced machine learning algorithm for the verification or improving the rules by an expert developer or more advanced tooling.

¹Signature, algorithm, http://www.igesture.org/algo_signature.html, [25-May-2012]

Chapter 6

Future work

6.1 Multiple cameras

As discussed earlier, the Kinect camera only has a limited range of a few meters. When one has a large room, this range will be insufficient. A possible solution for this problem could be to use multiple Kinect cameras. However, a few problems arise when one tries to do this.

A first problem that arises is that there is only support for connecting one camera at the time to a computer. This means that one needs to have a separate computer for each camera which makes it an expensive solution.

A second problem that might arise is the overlap in ranges. When two or more cameras project an infrared grid on the same place, this might have an effect on the accuracy. This needs to be further investigated. If multiple cameras recognize the same gesture at the same time, we obviously want that the corresponding action of the gesture is executed only once. This can be achieved with a low level fusion of the different cameras. Furthermore, there also needs to be a strategy if multiple cameras detect different gestures at the same time.

6.2 Scanning rooms

At the moment, one has to manually encode the different controllable objects with their location. This can be quite time consuming as one needs to manually measure the width/length of the room and the location of the different objects.

A possible solution can be that one is able to use the Kinect to “scan” the room. This can be done with a SLAM (Simultaneous Localization and Mapping) algorithm. The result is 3D map where the user can indicate the

location of the controllable objects in the room. This can be further extended with a user interface where one also can annotate the objects with the commands they understand.

6.3 Other extensions

Another extension might be to add person recognition. If this can be achieved, one can have person-specific gestures.

As discussed earlier, doing verification with template matchers does not always give the expected result. Therefore, future work might be to test with different algorithms like machine learning and different template matchers to make a comparison which algorithm gives the best result.

Chapter 7

Conclusion

To perform gesture recognition with a 3D camera, an algorithm is required that is capable of handling a continuous stream of coordinates of multiple body segments of a person. However, the current state of the art gesture recognition algorithms for continuous streams (declarative rules, gesture spotting approaches) focus on high recall. Therefore, they will often generate false positives. This represents a major problem since we try to apply gesture recognition in the context of home automation. Home appliances should not be triggered unintentionally. For instance, one would not like to have the stereo equipment suddenly start playing at maximum volume because the gesture recognition algorithm falsely identified a gesture.

To tackle this problem, we came up with a hybrid rule- and template-based gesture recognition solution. First, we spot the gestures based on declarative rules. This gives us potential matches. Because of the high recall and low accuracy, a verification step is needed. Therefore, we verify the potential matches with a template matching algorithm. When the template matcher confirms the gesture, we have correctly identified the gesture and the corresponding action must be executed. In our thesis, this would be the control of different appliances in a house.

To show a proof of concept, we created a full software stack to provide abstractions for the recognition of gestures and the controlling of appliances. We have chosen Mudra for the gesture spotting with declarative rules and iGesture as a gesture recognition framework to do the verification. The choice for a gesture recognition framework was done based on the fact that one could easily extend it with the algorithm of his preference. To proof that this can easily be done, we added the algorithm of our choice: Dynamic Time Warping.

The first thing that needed to be done was connecting our input device,

the Microsoft Kinect camera, to Mudra for the gesture spotting. As the Kinect SDK (C#) and Mudra (C) did not have an interface in a shared programming language, we opted to send the stream of coordinates from the Kinect SDK to Mudra via the OSC protocol. OSC is a protocol designed to cope with continuous streams of data. Secondly, the verification is done by sending potential matches found by Mudra to iGesture where they will be checked with a template matcher. As Mudra (C) and iGesture (Java) should also be decoupled using a networking layer, we opted to use XML-RPC for communication between these two entities. To make it easier for developers, we also extended iGesture so one can easily automatically generate the necessary declarative rules by allowing to select the characteristic points. Once this is done, an algorithm will automatically generate the corresponding declarative rule.

If a gesture is identified and verified, the corresponding action needs to be executed. If one can only control one appliance in the room with a gesture, we can send the necessary commands to the home automation central control unit. This control unit will communicate with the different appliances using the KNX protocol. However, if one can control multiple appliances in the room using the same gesture (e.g. activate), we need to estimate the direction one is pointing to in order to infer which appliance he wants to control. Therefore, we need to have a model of the room to know where all the appliances are located. To achieve this, we created a separate tool where one can easily define the different elements in the room with their corresponding location. Once we calculated which is the nearest relevant element in the room, we can send the corresponding command to the central control unit.

Finally, to demonstrate our approach, we did a small user test with five participants. Each participant was asked to do a number of gestures. The results show that Mudra is able to correctly spot most gestures in the continuous stream of information and that the number of false positives was reduced by using additional classification techniques from iGesture. However, the false positive gestures were not totally excluded. For future work we therefore advice to improve the declarative rule generation based on more advanced features, and to experiment further with other gesture sets. Nevertheless, our architectural contributions were shown to be fully functional and pushed gestural interfaces for home automation one step closer to reality.

Bibliography

- [1] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff. A Unified Framework for Gesture Recognition and Spatiotemporal Gesture Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31, Sep 2009.
- [2] R. A. Bolt. “Put-That-There”: Voice and Gesture at the Graphics Interface. In *Proceedings of SIGGRAPH 1980, 7th Annual Conference on Computer Graphics and Interactive Techniques*, pages 262–270, Seattle, USA, Jan 1980.
- [3] M. Elbouz, A. Alfalou, and C. Brosseau. Fuzzy logic and optical correlation-based face recognition method for patient monitoring application in home video surveillance. *Optical Engineering*, 50, Jun 2011.
- [4] K. Eva, P. Jochen, H. Joachim, and B. Alexander. Gesture recognition with a time-offlight camera. *International Journal of Intelligent Systems Technologies and Applications*, 5(3/4):334–343, 2008.
- [5] N. Gillian, R. B. Knapp, and S. O Modhrain. Recognition of multivariate temporal musical gestures. using n-dimensional dynamic time warping. In *Proceedings of NIME 2011, 12th International Conference on New Interfaces for Musical Expression*, pages 337–342, May 2011.
- [6] T. Hammond and R. Davis. LADDER, A Sketching Language for User Interface Developers. *Computers and Graphics*, 29(4), Aug 2005.
- [7] L. Hoste, B. Dumas, and B. Signer. Mudra: A Unified Multimodal Interaction Framework. In *Proceedings of ICMI 2011, 13th International Conference on Multimodal Interaction*, Alicante, Spain, Nov 2011.
- [8] L. Hyeon-Kyu and K. Jin-Hyung. Gesture Spotting From Continuous Hand Motion. *Pattern Recognition Letters*, 19(5-6), April 1998.
- [9] T. Starner Jake, J. Auxier, and D. Ashbrook. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring. In *International Symposium on Wearable Computing*, pages 87–94, 2000.

- [10] D. Kammer, G. Freitag, M. Keck, and M. Wacker. Taxonomy and overview of multi-touch frameworks: Architecture, scope and features. In *Workshop on Engineering Patterns for Multitouch Interfaces*, 2010.
- [11] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala. Proton: Multi-touch Gestures as Regular Expressions. In *Proceedings of CHI 2012, 30th International Conference on Human Factors in Computing Systems*, Texas, USA, Nov 2012.
- [12] C. Ming-yu, M. Lily, P. Padmanabhan, H. Alexander, and S. Rahul. Controlling your tv with gestures. In *Proceedings of MIR, 11th International Conference on Multimedia information retrieval*, pages 405–408, March 2010.
- [13] S. Nagaya, S. Seki, and R. Oka. A Theoretical Consideration of Pattern Space Trajectory for Gesture Spotting Recognition. In *Proc. of FG '96*, Killington, USA, Oct 1996.
- [14] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, Jan 1986.
- [15] E. Sanchez-nielsen, L. Laguna, L. Antun-canalís, and M. Hernandez-tejera. Hand gesture recognition for human-machine interaction. *Image Rochester NY*, 12(1):91–96, 2004.
- [16] C. Scholliers, L. Hoste, B. Signer, and W. De Meuter. Midas: A Declarative Multi-Touch Interaction Framework. In *Proceedings of TEI 2011, 5th International Conference on Tangible, Embedded, and Embodied Interaction*, Funchal, Portugal, Jan 2011.
- [17] B. Signer, U. Kurmann, and M. Norrie. igesture: A general gesture recognition framework. In *Proceedings of ICDAR 2007, 9th International Conference on Document Analysis and Recognition - Volume 02*, pages 954–958, Sep 2007.
- [18] A. Wilson, A. Bobick, and J. Cassell. Recovering the temporal structure of natural gesture. In *Proceedings of FG 1996, 2nd International Conference on Automatic Face and Gesture Recognition*, pages 66–71, Oct 1996.