Vrije Universiteit Brussel

Faculteit Wetenschappen
Department Computerwetenschappen
Web & Information Systems Engineering

Automated Gesture Candidates for Full-Body Gesture Recognition

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

Benjamin Peetermans

Promotor:	Prof. Dr. Beat Signer
Begeleiders:	Lode Hoste
	Brecht De Rooms

June 12, 2013

Vrije Universiteit Brussel

Faculty of Science
Department of Computer Science
Web & Information Systems Engineering

Automated Gesture Candidates for Full-Body Gesture Recognition

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master in Applied informatics, by

Benjamin Peetermans

Promotor: Prof. Dr. Beat Signer
Advisors: Lode Hoste
Brecht De Rooms

June 12, 2013

Abstract

The Microsoft Kinect[1], SoftKinetic DepthSense[2] and other 3D cameras allow to accurately track the movement of body limps. This enables developers to create a gesture recognition system without the need for additional gadgets (data glove or markers) to track human movements. Real-time gesture recognition is challenging since it requires us to detect patterns in the high-frequent streams generated from these cameras.

Complex gestures are hard to program manually due to the need for spatial and temporal constraints. Therefore, developers are required to use machine learning to obtain good recognition results. However, machine learning approaches often render an incomprehensible intermediate format and require a vast amount of learning data. Moreover, the resulting gesture definitions can not be programmatically extended or adjusted. On the other hand, gesture-oriented graphical toolkits such as iisu[3] or VolTra require a lot of manual operations to define a gesture which consumes development time. We argue that the development of full-body gesture definitions is still too difficult and consumes a lot of time.

We propose an automated gesture recognition system with a visual representation which allows for users to model a gesture by performing it once. Our approach models gestures as a sequence of poses. For each gesture a number of poses are generated which have to be matched in a specific order. A gesture is recognized when all poses are matched in the right order.
Our approach automatically generates a gesture model with an external representation which allows to visualise gestures in a 3D toolkit and enables users to refine the gesture model.

From our evaluation we conclude that automatically generating a gesture model reduces the development time compared to manually specifying the gesture model. Few manual refinements are required, lowering the development time of a gesture from minutes to seconds. Moreover, important benefits from programmatically specifying a gesture model are maintained such as extensibility and comprehensibility.

[1]http://www.microsoft.com/en-us/kinectforwindows/
[2]http://www.softkinetic.com/solutions/depthsensecameras.aspx
[3]http://www.softkinetic.com/products/iisumiddleware.aspx

Acknowledgements

Firstly, I would like to express my gratitude to my supervisors Lode Hoste and Brecht De Rooms for their constant guidance and time spent on helping me to achieve this thesis. The weekly meetings helped me to keep working forward and their insights were invaluable for the realization of this thesis.
I am grateful to Prof. Dr. Beat Signer for supporting this thesis and providing feedback during the presentations which were held during the course of the academic year.
I would like to thank my family and friends for supporting me in the realization of this thesis and for helping me to record several gestures throughout the year. A special thanks goes out to my girlfriend who supported me in every phase, from recording gestures to proof reading this dissertation.

Contents

List of Figures

List of Tables

1 Introduction

For many years, the standard for human computer interaction (HCI) was a WIMP style using mouse and keyboard to interact with a computer. With the rise of new technologies in the past few years, there has been a growing interest in using multimodal interfaces other than the standard mouse and keyboard. The biggest advantages of multimodal interfaces are fewer errors, and an advanced usability since multimodal interfaces are very intuitive.

Kendon[12] defines a gesture as follows:

> A gesture is a form of non-verbal communication in which visible bodily actions communicate particular messages, either in place of speech or together and in parallel with words. Gestures include movement of the hands, face, or other parts of the body. Gestures differ from physical non-verbal communication that does not communicate specific messages, such as purely expressive displays, proxemics, or displays of joint attention.

Gestures are one of the most frequently used modalities. In the past years gestures have become more and more prominent in our daily lives, especially since smart phones with touch screens have become increasingly popular. To use these gestures as input, we need to recognize them using a gesture recognition system.
Gestures can be captured using input devices such as a mouse or by using tracking cameras which allow using the whole body to perform gestures. Extensive research is conducted on how to detect these gestures, especially for gestures using a wiimote [20], pen-based gestures [6] and gestures performed with our own body [9]. The gestures used in this thesis are performed with our own body. The data is skeleton data, which consists of the coordinates of joints in a 3D space.

In this thesis, related work on gesture recognition based of skeleton data is discussed, including Dynamic Time Warping (DTW), Hidden Markov Models (HMM) and Neural Networks (NN). We find that current work lacks extensibility and feedback.
Therefore we propose a novel approach where gesture definitions are automatically generated in a comprehensible and extensible format. A gesture definition is automatically deducted from a single sample, and allow for manual adjustment if needed. The gesture definitions both have a visual 3D representation as a declarative intermediate format. The former allows to easily adapt generated definitions while the latter provides the possibility to extend the definitions with custom constraints.

1.1 Goal

Current automated gesture recognition techniques have limitations which are reflected in existing applications. Typically gesture definition shave no comprehensible representation and can therefore not be easily extended by a gesture programmer. The lack of feedback and debugging can be frustrating for users which results in a bad user experience. Existing research shows that declarative programming of full-body gestures has important advantages over imperative programming. According to Hoste [21]:

> A declarative rule-based language description of gestures in combination with a host language increases the extensibility and reusability of multi-touch gestures.

De Rooms [4] describes advantages of declarative gesture definitions over imperative languages such as comprehensibility, accuracy and feedback.

Developing gesture definitions requires experts to model the gesture and can consume a lot of development time for complex gestures. We argue this can be improved by automatically proposing a gesture model which can be refined by the gesture developer. We aim for the gesture recognition system to be able to recognize gestures without prior segmentation and in real-time.
Our approach can be used in a variety of domains but is best suited for gestures which can be divided into different steps that have to be followed, like dance gestures. Since gestures can be recognized in real-time and no prior segmentation is needed, our approach could also prove useful in activity recognition.

1.2 Microsoft Kinect

For tracking the human body, 3D sensors are generally preferred. There are various types of 3D sensors such as the Microsoft Kinect, Softkinetic's Depthsence and Asus Xtion[4], but we will only discuss the Microsoft Kinect since it is the most commonly available of all 3D sensors.

The Kinect is a input modality which was originally developed for gaming applications on the Microsoft Xbox 360[5], but quickly found its way to the desktop allowing developers to experiment with its possibilities.
The Kinect sensor uses technology invented by Primesense[6] to generate a depth image. It consists of a infra red (IR) laser emitter, an infra red camera and a RGB camera. The IR laser projects a semi-random pattern of speckles onto the scene, as shown in Figure 1.1, which is captured by the IR camera. The pattern is a 3x3 matrix of random dots with a centred registration dot in each square. This allows the Kinect to calculate the distance of the speckles to the camera and thus generating a depth image.
This depth image is analysed using the approach of Shotton et al. [22], which allows to generate skeleton data using a single depth image. Their approach consists of 2 different stages, as shown in Figure 1.2. First the single depth image is segmented into a probabilistic body part labelling, with each part to be spatially close to skeleton joints of interest. In a second stage these parts are transformed into skeleton data using a very large and varied set of training data (more than 100.000 samples).

[4]http://www.asus.com/Multimedia/Xtion_PRO_LIVE/
[5]http://www.xbox.com
[6]http://www.primesense.com/

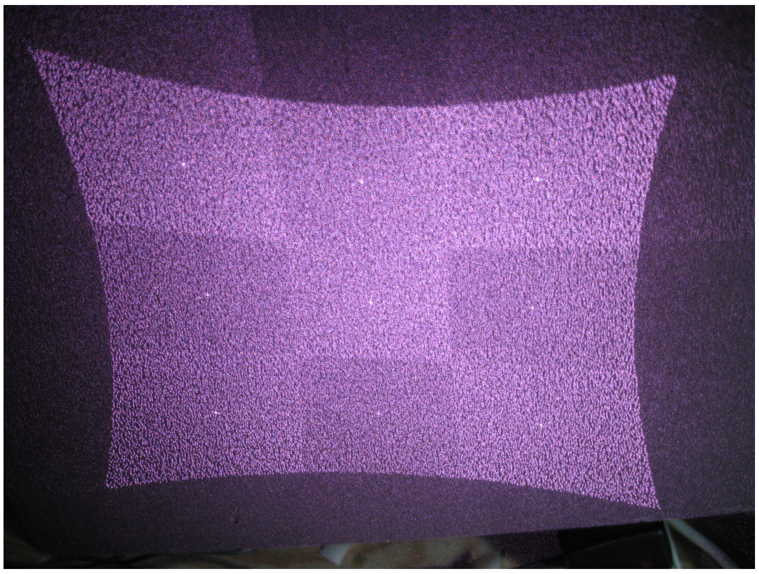


Figure 1.1: IR speckle pattern projected by the Kinect

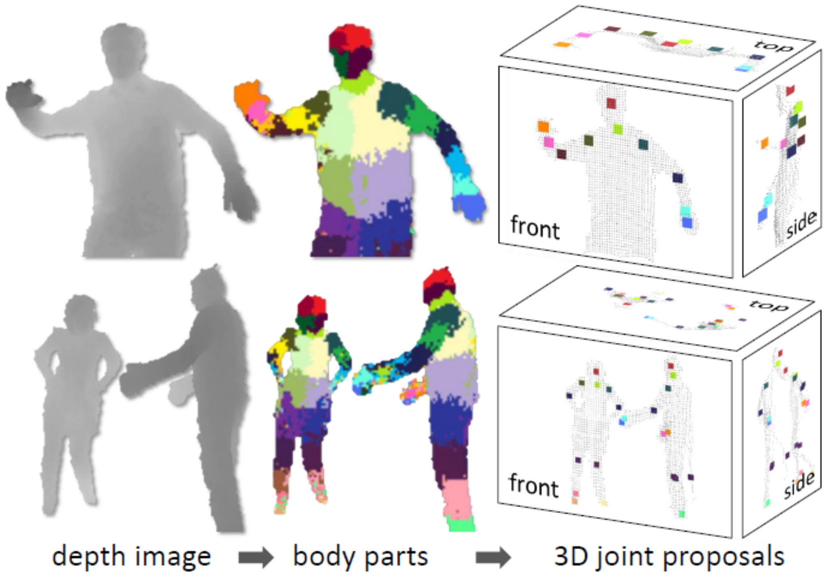


Figure 1.2: 2 stages described by Shotton et al. [22]

1.3 VolTra

Voltra [4] is a 3D editor which presents a novel concept to define gestures. The 3D gestures are described by using volumetric constructs and constraints between these volumes. A volume is always assigned to a certain joint which can be configured by the user. As a first step for the recognition, code is generated that registers when this joint enters the volume. Each volume can be moved, rotated and scaled as desired. Constraints such as an order and a time constraint can be added to these volumes to help define a gesture.

VolTra compiles these visual definitions to declarative rules which are send to Midas [21]. When a joint enters or leaves one of its volumes the event is registered as a fact in Midas. The gesture rules are matched against the enter and exit volume facts and when all constraints of a gesture are satisfied the gesture is recognized. VolTra also provides debugging by allowing to play samples. When playing samples, VolTra simulates the gesture stream by sending point events to Midas. By interweaving debugging code in the declarative rules, VolTra can ask Midas to send information back on the matching process which allows for visual feedback on the gesture recognition.

Figure 1.3 shows the GUI of VolTra with all major components denoted. The Samples area gives a list of the currently loaded gesture samples. By choosing a sample, the puppet in the Gesture scene can replay the selected sample. The Gesture sliders can be used to manually move the puppet to a point in the selected sample. The Menu allows to create new gestures, save them and let the puppet replay the selected sample, among other things. When a gesture is created it is shown in the Gestures list. To model a gesture the user should select a joint trajectory and a parent trajectory from the Joints menu. The joint trajectory will be drawn relative to the parent trajectory and new gesture volumes will snap to the curve to aid the user. To add new volumes or constraints to the gesture, the volume menu provides the option to create volumes for the selected joint, or put constraints between volumes.

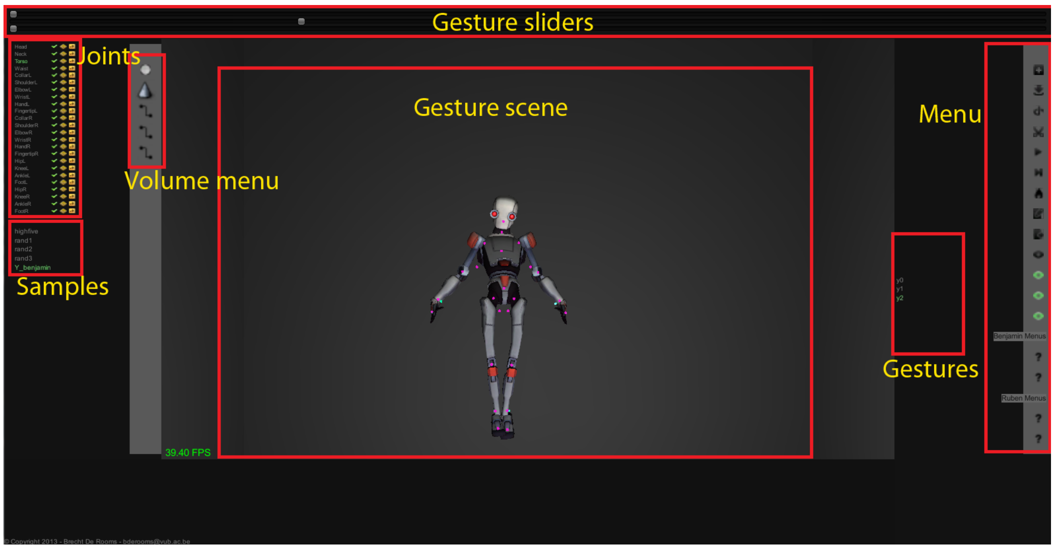


Figure 1.3: GUI of VolTra

1.4 Midas

Midas [21] is a declarative framework which was developed for multi-touch gestures. Midas receives gesture definitions from VolTra as rules and joint-volume interactions are registered by Midas and asserted as facts. The gesture definition rules are matched against facts by Midas and when a gesture is matched the rule is triggered. To extract information from a stream of events temporal and spatial operators are used. Midas automatically annotates all facts with timing information. This timing can be used to check the relationship between the timing attributes of different joints by using temporal operators. Spatial operators are used to check the distance between facts. Each fact is annotated with its position from which the distance can be calculated.

The syntax and rule-engine of Midas are inherited from the CLIPS[7] language, which provides continues incremental evaluation of rules by using the Rete algorithm. The key insight in the Rete algorithm is that rules can contain similar patterns in its left-hand-side (LHS), so we do not have to check the entire LHS of every rule when a new fact is asserted.
The Rete algorithm builds a tree where each node (except the root) corresponds to a pattern which occurs on the LHS of a rule. Each of these nodes keeps facts which satisfy their pattern in memory. Since every node contains only a part of the LHS of a rule, the path from the root to a leaf node contains the entire LHS. When facts are asserted they are propagated along the tree and added to the memory of matched nodes. When all nodes from the root to a leaf are satisfied, the entire LHS of the rule is matched and the rule can be triggered. This system reduces redundant processing to optimize for speed but sacrifices memory in return.

[7] `http://clipsrules.sourceforge.net/`

1.5 Thesis Structure

Chapter 2 describes the desired requirements in automated gesture recognition. The customizability, precision, extensibility, comprehensible representation, one-shot learning and feedback requirements are proposed and their need in an automated gesture recognizer is motivated.

In chapter 3, a literature survey is conducted on the most popular gesture recognition methods such as Dynamic Time Warping and Hidden Markov Models. Their issues are discussed and evaluated on the earlier proposed requirements for automated full-body gesture recognition.

Chapter 4 describes our approach to automate full-body gesture recognition. A divide-and-conquer strategy is applied for gesture recognition by dividing a gesture into a sequence of poses. When each pose is matched, the gesture is recognized. The calculation of gesture confidence values is explained and we describe how the sequence of poses is combined into a precise gesture definition.

Chapter 5 describes the implementation of the earlier proposed approach in VolTra. VolTra lets us create an external representation of gestures and enables us to generate declarative rules for gesture recognition. The implementation of confidence values is explained as well as the generated declarative rules for gesture recognition.

In chapter 6, our approach is evaluated with dance gestures. An overview of the used dance gestures is given and the choice for these gestures is motivated. Loose and strict gesture definitions are generated which are evaluated with dance gestures and random movements to test for false positives.

Possible future improvements for our approach are proposed in chapter 7. Methods on how to automate full-body gesture recognition even more are presented and suggestions on how to improve the confidence value of gestures are provided.

2 Requirements

By researching related work, 6 requirements were found to be essential for an automated gesture recognition system: *One-shot learning, Precision, Extensibility, Comprehensible representation, Customizability* and *Feedback*. These requirements are discussed and their need in an automated gesture recognition system is motivated.

2.1 One-shot learning

Most gesture recognition methods require a vast amount of training data when a new gesture has to be defined. This training data is hard to gather and a hassle for users who want to add new gestures to the system. Adding gestures should be easy and user-friendly, using only one training sample. For example Fei-Fei et al. [8] propose a method to categorize images using no more than 5 training samples. One-shot learning allows users to generate a gesture definition by using only one training sample.

2.2 Precision

A slight deviation in the execution of a gesture can influence whether or not the gesture is considered to be performed correctly. The user should be able to select a threshold which determines when an error is no longer acceptable.

2.3 Extensibility

Most gesture recognition systems do not allow a gesture definition to be modified after it has been created. If a small change has to be made or the definition contains a flaw, the entire definition has to be recreated. To enhance the usability, users should be able to modify generated gesture definitions to solve errors, or build upon the gesture to create a new, more complex gesture.

2.4 Customizability

Most gesture recognition systems track the full body or only the hands of a user, but some gestures only require specific joints to be matched. Users should be able to customize a gesture definition by selecting which joints are relevant for a gesture. Only these important joints should be used in the gesture definition. For example, a slight deviation of the elbow in a high-five gesture is still a high-five, but a deviation of the hand could become a different gesture. Figure 2.1 shows a high-five gesture performed in two different ways. On the right the arm is completely stretched but on the left the arm is partially bent. Even thought the execution of these two gestures is different they both are high-fives. In figure 2.2 the arm is stretched, but there is a small deviation of the hand which is directly above the head instead of to the side of the head. This should not be considered as a high-five gesture. By only selecting the right hand as an important joint we can make sure the high-five gesture is matched in figure 2.1 and not matched in figure 2.2.

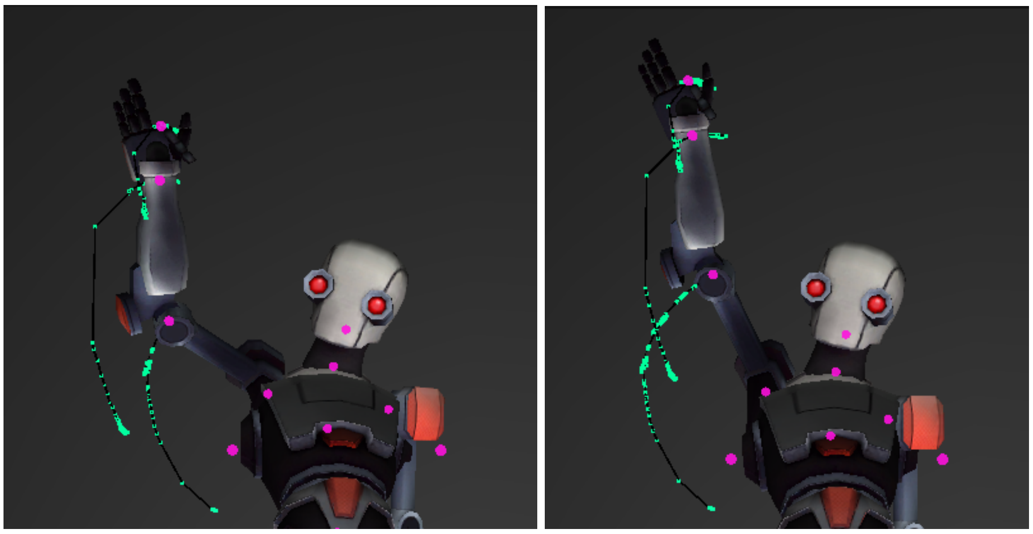


Figure 2.1: Two high-five gestures with a deviation in the right elbow

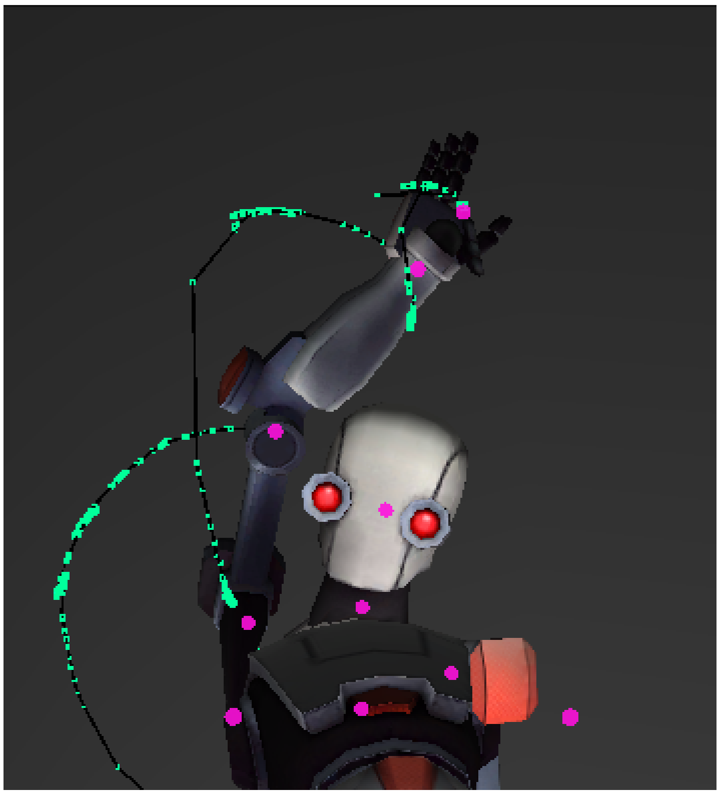


Figure 2.2: Two high-five gestures with a deviation in the right hand

2.5 Comprehensible representation

Some gesture recognition systems do not provide the user with information about the gesture definition. This means it is impossible to adjust the gesture, or see what could go wrong with the current gesture definition. By creating a comprehensible gesture representation we can provide the user with human-readable information about the gesture and the gesture definition. This allows the user to understand what is wrong with the generated gesture definition and to understand what has to change to fix flaws. An external representation can also be used to provide feedback to the user.

2.6 Feedback

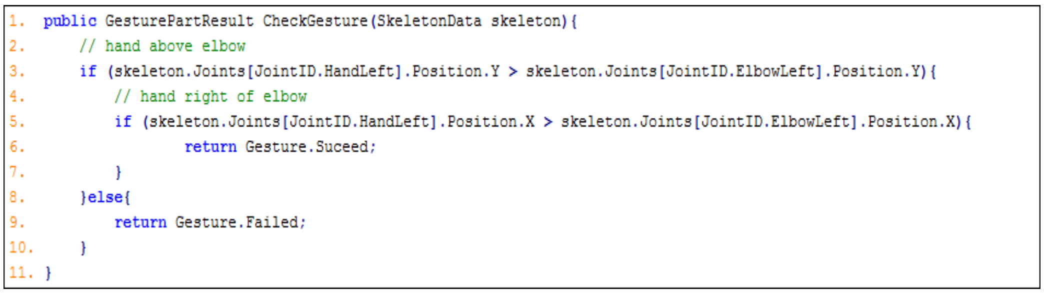
Feedback is important for developers to understand the generated gesture definitions. It helps the developer to see flaws in gesture definitions and to adjust them if necessary. Most gesture recognition system provide no feedback, or very limited (e.g. whether a gesture is matched). Feedback on which constraints fail and when they do so could help developers to quickly detect flaws in the gesture definition.
Karam [11] divided feedback for end users into 3 different stages: Reflexive feedback, recognition feedback and response feedback.
Reflexive feedback is the first stage of feedback and occurs while the gesture is performed. This stage provides users with information about the gesture they are performing. Karam's results showed false positive and system errors were significantly lower when reflexive feedback was used in tests.
Recognition feedback occurs when the gesture recognition process is over and gives feedback about which gesture was recognized. Response feedback provides the user with a notification that the task is complete. These stages of feedback are also beneficial for developers to aid with debugging.

3 Related Work

There have been different approaches to handle gesture recognition in the past 40 years. Most techniques rely on gadgets like a glove [14] or markers [13] but in the past decade interest has grown in gesture recognition using a 3D camera. Especially in the past few years gesture recognition has become a hot topic due to new technologies like the Microsoft Kinect[8]. As already mentioned in section 1 the context for this work is skeleton tracking, so we will elaborate on some techniques which also use skeleton data.

3.1 Imperative Coded Gestures

Imperatively coding gestures is the simplest form of gesture recognition. Gestures are manually coded by an expert and can not be easily changed by non-expert users. The gesture definitions typically consist of simple positional constraints between joints. The code in figure 3.1 is an example of how these constraints are coded in the Kinect for Windows SDK[9]. In this example we check if the left hand is above and to the right of the left elbow by comparing the coordinates of the left hand joint and the left elbow joint. When the left hand joint is above the left elbow joint (compare the Y-values) and to the right of the left elbow joint (compare the X-values), the method succeeds.



```
public GesturePartResult CheckGesture(SkeletonData skeleton){
    // hand above elbow
    if (skeleton.Joints[JointID.HandLeft].Position.Y > skeleton.Joints[JointID.ElbowLeft].Position.Y){
        // hand right of elbow
        if (skeleton.Joints[JointID.HandLeft].Position.X > skeleton.Joints[JointID.ElbowLeft].Position.X){
                return Gesture.Suceed;
        }
    }else{
        return Gesture.Failed;
    }
}
```

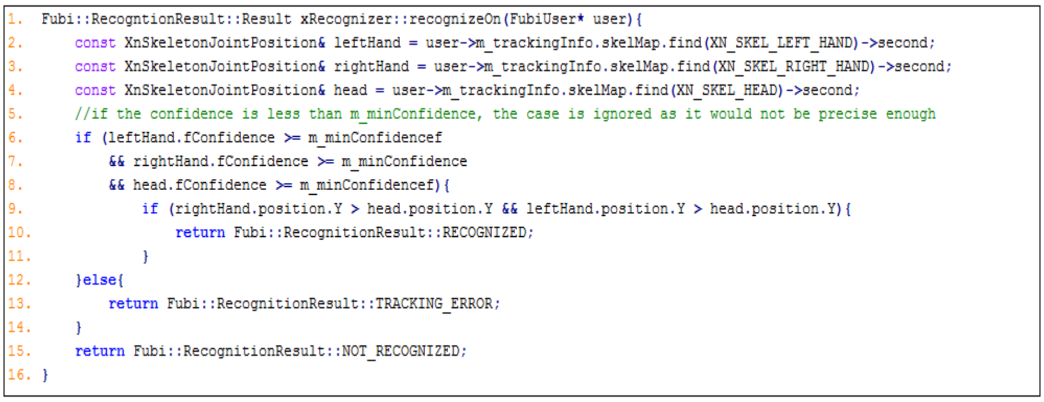
Figure 3.1: Hard coded gesture constraint for the Kinect for Windows SDK

[8]http://www.microsoft.com/en-us/kinectforwindows/

[9]http://www.microsoft.com/en-us/kinectforwindows/develop/overview.aspx

When detecting complex gestures there is a need for more complex constraints. For example, a gesture that requires the right hand joint to move relative to the right shoulder joint is already more difficult to code. The developer needs knowledge of measurements and orientation in a 3D space to develop such constraints. If temporal constraints have to be coded as well, the relative time between different states in a gesture have to be checked which is very time-consuming to code.

There are some frameworks that try to deal with the complexity of manually coding gestures, for example FUBI (Full Body Interaction Framework)[10]. The FUBI framework helps the developer to code more complex constraints by providing useful information such as a confidence value and the movement speed of joints. Figure 3.2 shows how an expert could define a constraint in FUBI where two hands need to be above the head. In addition to the comparison of the Y-values of hands and head, there is a control structure to check whether the joints reach a certain confidence level before trying to recognize the actual gesture. Another feature of FUBI is to keep states of gestures, which can be seen as poses. It is possible to put constraints on the duration of states or the interval between them. All constraints can be reused and a combination of linear movements and states is possible. However, no visual representation is provided and there is no support for an automated gesture definition.



```
Fubi::RecogntionResult::Result xRecognizer::recognizeOn(FubiUser* user){
    const XnSkeletonJointPosition& leftHand = user->m_trackingInfo.skelMap.find(XN_SKEL_LEFT_HAND)->second;
    const XnSkeletonJointPosition& rightHand = user->m_trackingInfo.skelMap.find(XN_SKEL_RIGHT_HAND)->second;
    const XnSkeletonJointPosition& head = user->m_trackingInfo.skelMap.find(XN_SKEL_HEAD)->second;
    //if the confidence is less than m_minConfidence, the case is ignored as it would not be precise enough
    if (leftHand.fConfidence >= m_minConfidencef
        && rightHand.fConfidence >= m_minConfidence
        && head.fConfidence >= m_minConfidencef){
            if (rightHand.position.Y > head.position.Y && leftHand.position.Y > head.position.Y){
                return Fubi::RecognitionResult::RECOGNIZED;
            }
    }else{
        return Fubi::RecognitionResult::TRACKING_ERROR;
    }
    return Fubi::RecognitionResult::NOT_RECOGNIZED;
}
```

Figure 3.2: Hard coded gesture in FUBI

[10]http://www.informatik.uni-augsburg.de/en/chairs/hcm/projects/fubi/

Another framework which tries to enhance the complexity of manually coding gestures is SoftKinetic's IISU Interaction designer[11]. Like FUBI, this framework allows us to directly access joint information. Figure 3.3 shows an example of how to code a Chinese bow gesture. A Chinese bow gesture consists of a bow in a certain angle with the left and right hand on the torso, as shown in Figure 3.4.

To code this gesture, we first define some variables to represent joints, since it makes the code more readable. The next part computes the current angle of the upper-body based on the current position of the pelvis and the head. Next we test the hands to be up. We should test whether they are up and in the same location, on the torso, but for simplicity we only check if the hands are higher than the pelvis with atleast a certain threshold. If the hands are in the right position and the upper-body has a big enough angle, we recognize the Chinese bow gesture.

```
function main()
 pelvis = USER1_SKELETON_PARTS_Pelvis
 left_hand = USER1_SKELETON_PARTS_LeftWrist
 right_hand = USER1_SKELETON_PARTS_RightWrist
 head = USER1_SKELETON_PARTS_Head
 //Compute angle based on pelvis and head
 torso_Joystick_Yaxis_Degrees = SK.axisAngle(head - pelvis, SK.Axis_UP,SK.Axis_FRONT)
 //Check if left and right hand are down
 if(left_hand_up) then
    if(left_hand:up() - pelvis:up() < (hand_up_threshold - hand_up_hysteresis) ) then
        left_hand_up = false
    end
 else
    if((left_hand:up() - pelvis:up()) > hand_up_threshold) then
        left_hand_up = true
    end
 end
 if(right_hand_up) then
    if(right_hand:up() - pelvis:up() < (hand_up_threshold - hand_up_hysteresis) ) then
        right_hand_up = false
    end
 else
    if(right_hand:up() - pelvis:up() > hand_up_threshold) then
        right_hand_up = true
    end
 end
 //Check if the body has the correct angle and both hands are up
 if(torso_Joystick_Yaxis > 0.55 && left_hand_up && right_hand_up) then
    is_bowing = true
 else
    is_bowing = false
 end
end
```

Figure 3.3: Chinese bow gesture in IISU

[11]http://www.softkinetic.com/en-us/products/iisumiddleware.aspx

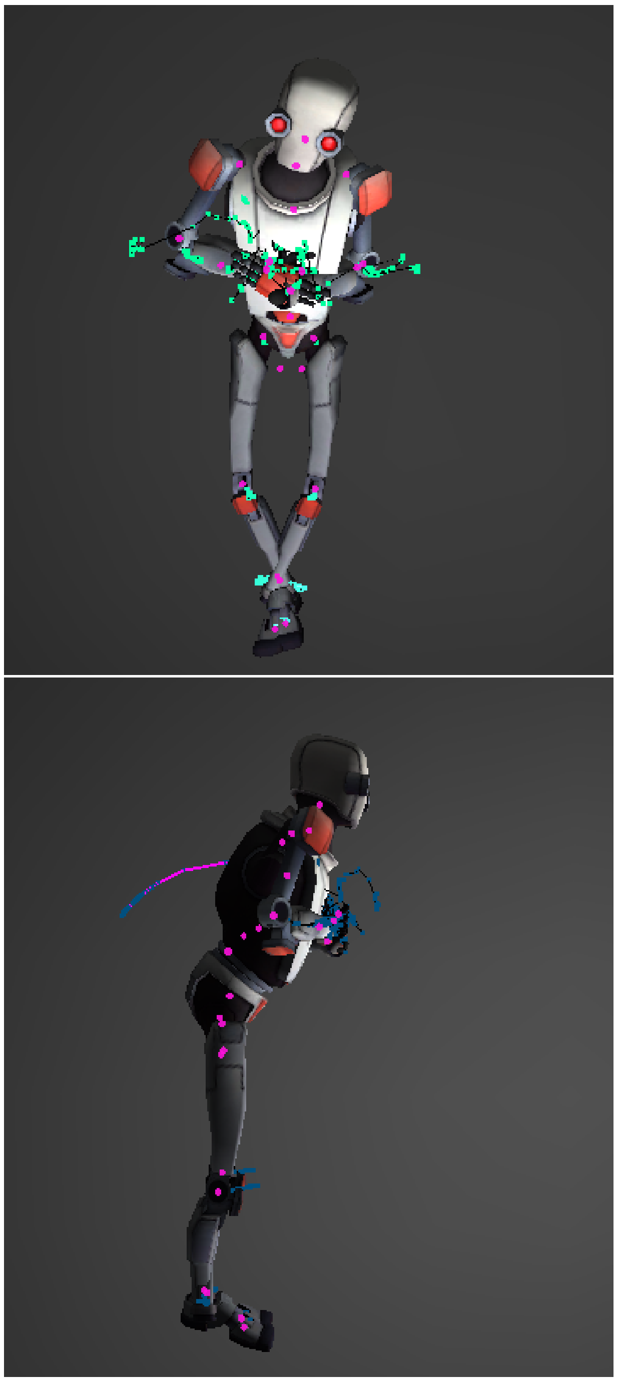


Figure 3.4: Chinese bow gesture

Frameworks make it easier for developers to code constraints on gestures. However, it still boils down to comparing coordinates of different joints which becomes very complex when multiple joints are involved or when more advanced gestures are required.
Furthermore, there is no possibility to automatically generate a gesture definition with a visual representation. SoftKinetic's IISU does provide a Unity3D[12] plugin where a simplified 3D avatar consisting of spheres can be used. Thanks to the volumetric avatar, volumes can be used together with the collision detection of Unity to program gestures. This provides a visual representation of the gesture definition, but it can not be automated.

[12]`http://unity3d.com/`

3.2 Template-based

To recognize a gesture from spatio-temporal data, template based techniques such as DTW (Dynamic Time Warping) or $1 recognizers can be used.
To recognize a gesture we first need a template which must be matched. A performed gesture is recognized if it is similar enough to the template. The problem is that two gestures can have a different timespan. The same gestures can be performed at different speeds. To accommodate this problem, DTW can be used.
DTW finds the optimal match between two sequences of trajectories and takes the possible difference in length of two sequences into account. The top part of Figure 3.5 represents two gestures with a distance measure between them. Any distance measure which aligns the i-th point of the first sample to the i-th point of the second sample (Euclidean, Manhattan,...) will give a poor similarity. The bottom half of Figure 3.5 depicts the same two gestures with an non-linear (elastic) distance measure. Here similar shapes will be matched, even if they are out of phase in the time axis, which gives a better similarity.

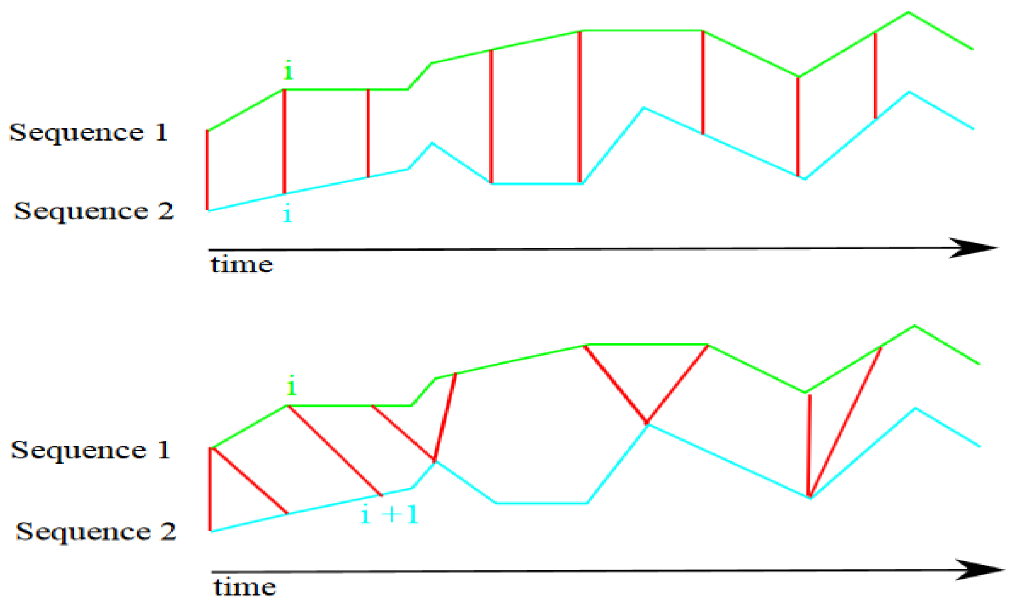


Figure 3.5: Linear and non-linear pattern matching

DTW was a technique used in speech recognition, but also has applications in gesture recognition. Corradini [3] uses DTW for off-line recognition of arm gestures, and Doliotis et al. [7] to recognize hand gestures.

Most DTW implementations require that the start and endpoint of a gesture are defined. In isolated samples we can define a special signal to mark when a gesture is performed, but this is impossible for continuous gesture recognition. Li et al. [16] propose a method to estimate the endpoint of gestures in a continuous video stream by comparing the stream to training data. When the endpoint of a gesture is estimated, the same point can be used as the starting point of the next gesture, upon which a new endpoint is estimated. This cycle repeats throughout the video stream.

Gesture recognition systems using DTW have been applied to dance gestures in the past with very promising results. Raptis et al. [19] presented a method to recognize dance gestures using a Microsoft Kinect camera with a recognition rate of 96.9%. In dance gestures it is possible to use the DTW algorithm since the gestures are typically synchronized with music. By applying beat-detection on the music and aligning beats with sub-gestures we exactly know the start- and endpoint of each gesture. This greatly simplifies the recognition process since they can segment the sample based on the beat-detection and assume that only a limited number of moves can span a predefined number of beats. Moreover, these moves are predefined and well-known to the system. There are some DTW approaches which do not need prior data segmentation. Bettens et al. [1] present a DTW implementation which uses multiple DTW grids. Each grid hypothesizes a different starting point, so no prior data segmentation is needed. To deal with the heavy computation this implies, they use an iterative implementation where only one column is evaluated in each grid at a given time.

DTW excels in comparing two sequences, but also has disadvantages. All points in 2 sequences are compared so the input data needs to be segmented. Hoste [10] presents a gesture spotting technique to automatically segment a trajectory which could be used to counter this disadvantage. Due to the incomprehensible internal matrix representation, a DTW gesture definition can not easily be extended or customized.

The $1 recognizer presented by Wobbrock et al. [24] is able to recognize gestures with the same accuracy as DTW, but does not use complex mathematical procedures. Their recognition algorithm re-samples the input sample to exclude problems with different movement speeds of gestures, next the sample is rotated to have an optimal alignment with the template gesture. After these steps, the sample is scaled and matched to recognize the gesture. A disadvantage of the $1 recognizer is that it has no concept of time, so it is impossible to use constraints such as time and speed to define a gesture. Furthermore, $ gesture recognizers are focussed on 2D gestures. Even though some claim they can be easily extended to 3D gestures [23], the extension to bring multiple joints in relation with each other is not trivial.

3.3 Declarative Approaches

A declarative approach brings all advantages of declarative programming to gesture recognition. Developers can reason about gesture definitions instead of coding the control flow of a gesture.
Hoste et al. [10] present a novel gesture spotting approach that offers automated reasoning over a complete motion trajectory. Major changes in a small section of the trajectory are stored as potential control points. When the complete trajectory has been parsed, the top *m* points are chosen. These points are visualized and can be manually refined by the developer. Spatio-temporal constraints can be relaxed by the developer to allow the matching of noisy gestures. Their approach inherently supports overlapping submatches since it searches for a combination of events which matches the gesture definition. The evaluation shows that their approach allows for a high recall rate even though the current implementation is not rotation invariant.

3.4 Machine Learning

3.4.1 Hidden Markov Model

Hidden Markov Models (HMM) are known for their applications in temporal pattern recognition, like speech, handwriting and gesture recognition. They use hidden states that correspond to different phases of an action. To keep the modelling of joints tractable two assumptions are made: the values of any state are only influenced by the values of the state that directly precedes it. This is known as the Markov assumption and greatly simplifies the model. The second assumption is that observations only influence the current state, so subsequent observations are independent. This means it is impossible to take long-range observations into account.

Yamato et al. [25] uses HMM to recognize different tennis strokes. For the training of the HMM, the Baum-Welch algorithm was used to find the unknown parameters. The Viterbi algorithm is used to find the most likely sequence of hidden states that generate the observed sequence.
Bevilacqua et al. [2] present a HMM based system for real time gesture analysis. Their approach continuously updates parameters of the performance of a gesture. The focus lies on the time progression, which is the current state of the gesture, and the likelihood, which gives a similarity value between the performed and the pre-recorded value. With an accurate estimation of the time progression and the likelihood it is possible to predict the evolution of the current gesture.
The system is developed with applications in performing arts in mind, where little training data is available. This means a statistical training algorithm like the Baum-Welch algorithm can not be used, in contrast with other HMM implementations. The learning procedure is simplified by using prior knowledge, which makes it possible to estimate parameters from a single training sample. The writers of this paper choose this method because they found that is in impractical to build a general gesture database, since the gesture data is dependent on the artistic context.
The algorithm which is used for the deduction of the time progression and the likelihood has a major disadvantage: long samples incur a large number of states. To reduce this computational overload a sliding window is used. The computation of the likelihood has to be computed on all the states, not just those in the sliding window. Whenever the window is moved, values that were not considered in the previous window are set to zero in the new window. This allows to compute the likelihood without increasing the CPU load.

3.4.2 Neural Networks

Lamar [15] defines a neural network as:

> A massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use.

Several types of neural network can be used for gesture recognition. Maraqa [17] compared two recurrent neural networks to recognize Arabic sign language for static hand gestures, namely a fully recurrent neural network and Elman's and Jordan's recurrent neural network. A digital camera and a coloured glove were used for input data. This data is segmented into 6 layers by using the HIS color model, 5 for the fingertips and 1 for the wrist. From these layers they extract 30 features, like angles and distances between fingers and wrist, which are grouped in one vector to represent a single gesture. This features vector serves as the input for the neural networks. Their results show that fully recurrent neural networks systems have a better recognition rate than Elman's and Jordan's recurrent neural networks.

Murakami [18] presented a Japanese sign language recognizer using a combination of two neural networks. Figure 3.6 shows the architecture of their system. A data glove is used to capture input data, which is then normalized. To detect the starting point of a gesture, each gesture has to start from a specific posture, which is detected by a neural network for posture recognition. When this posture is detected, it is considered as the starting point for a gesture, the data from the data glove is send to the second neural network (Elman's recurrent neural network) which recognizes sign language. To determine the endpoint of a sample, the history of the result of each sampling point is kept. The current sampling points are compared to the history, from which they determine if the sample is at an end.

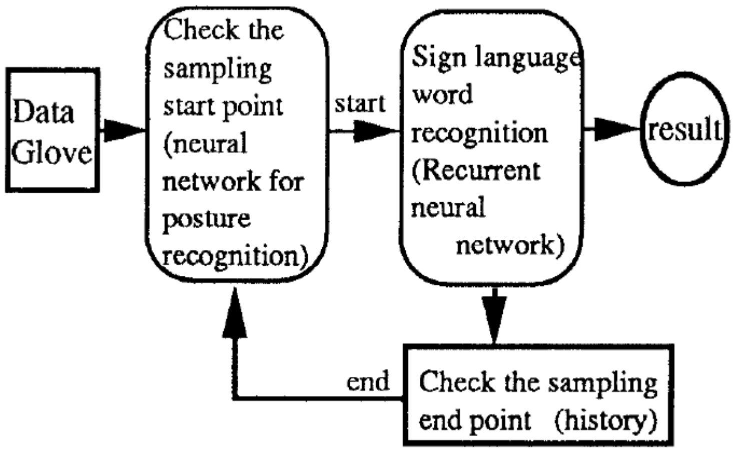

Data Glove
Check the sampling start point (neural network for posture recognition)
start
Sign language word recognition (Recurrent neural network)
result
end
Check the sampling end point (history)

Figure 3.6: sign language recognition system [18]

Even though the results of gesture recognition using neural networks are promising, they require a vast amount of training data which makes it impossible to easily add new gesture definitions. Maraqa [17] used 900 training samples to recognize 30 hand gestures.

Hidden Markov Models and neural networks are often referred to as a black box since they do not provide information about the gesture definition. No comprehensible representation is used and due to the amount of training data needed these techniques do not provide extensibility.

4 Automated Full-Body Gesture Recognition

4.1 Introduction

Graphical toolkits for gesture specification enable users to define gestures based on a recorded sample. However users still need to manually declare, scale an rotate regions which joints have to match. These transformations are necessary to let users choose which parts of the gesture are important and how precise or how loose each part should be defined. While the visual approach of graphical toolkits already help, manually transforming each region in a gesture can still take some time for gestures with a lot of constraints, especially long gestures.
We introduce an approach to automatically generate these regions in a graphical toolkit based on a single recorded sample. The user merely has to select the starting point and endpoint of a gesture in the recorded sample, upon which we generate a visual definition for the selected gesture. Only one sample is needed as training data and the generated gesture definition can be adjusted by users. This ensures the extensibility of our gesture and allows users to manipulate the gesture definition if desired.

4.2 Poses

Gestures can be defined as a sequence of poses, so recognizing a gesture boils down to recognizing each individual pose and check if these poses were performed in the right order. To determine the poses that compose to a gesture, the trajectory of the gesture has to be known. This means one sample of the performed gesture is needed and the user has to select the the starting point and the ending point of the gesture in that sample. Poses are generated on a time interval between these two points, each pose matching a certain moment in time between the starting point and ending point of the gesture as is explained later. For each joint in a pose, a matching region is determined. This joint region defines how close the joint has to be to the original curve to match the pose and is represented as an ellipsoid, as shown in figure 4.1.

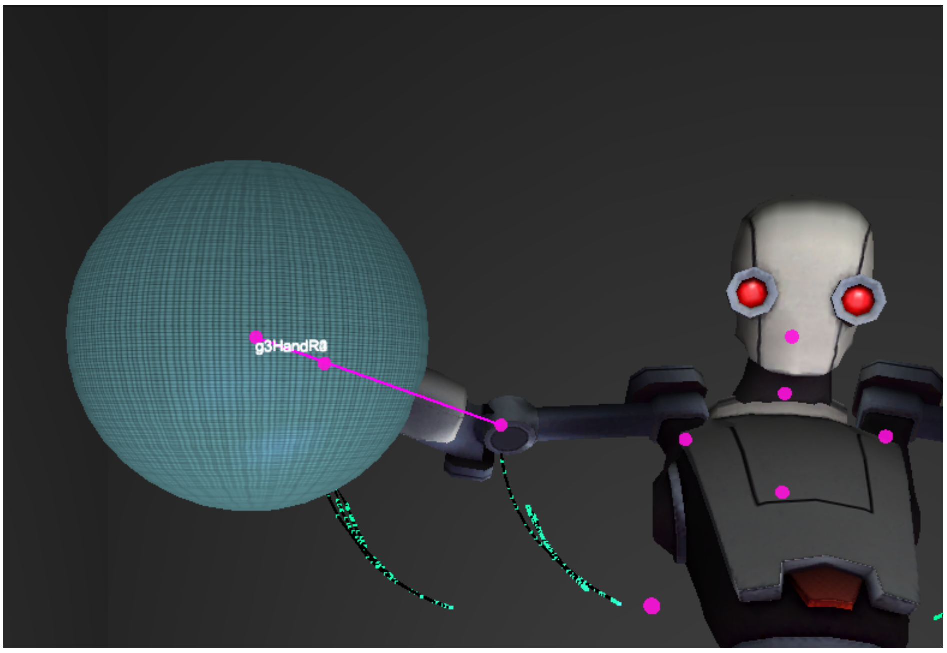


Figure 4.1: Joint region of the right hand joint represented by an ellipsoid

It is important that there is a significant overlap between the joint regions of joints in subsequent poses, since the precision of the gesture definition depends on it. For each joint the joint region of a pose should start before the joint region of the previous pose ended. An example is shown in Figure 4.2, where a gesture to raise both hands is depicted. The gesture takes 4 joints into account (right hand, right elbow, left hand and left elbow) and defines 2 poses which are represented by blue and yellow ellipsoids. Each ellipsoid represents the joint region of a joint per pose. The time interval used between these poses is chosen in a way so that joint regions partially overlap between poses for each joint. For explanation purposes, the overlapping areas are denoted by red circles. Since the elbow joints in pose 2 remain in the proximity of their location in pose 1, the overlap between these joint regions is much bigger than the overlap of the hand joints.
The movements performed outside of joint regions are not monitored by the gesture recognition system, so a deviation which occurs outside of these regions is not registered and will not affect the gesture recognition. This could lead to false positives, so there should be as little space as possible between the joint regions of subsequent poses. For each gesture a time interval to generate the poses should be specified by the user. This time interval should be chosen in a way that there is always a small overlap between joint regions.

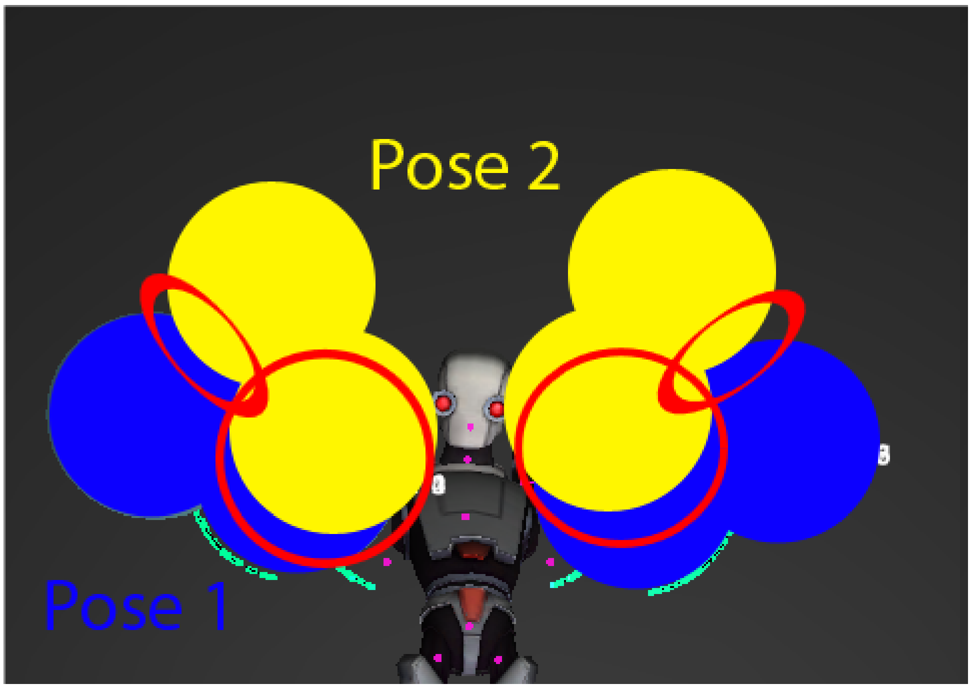
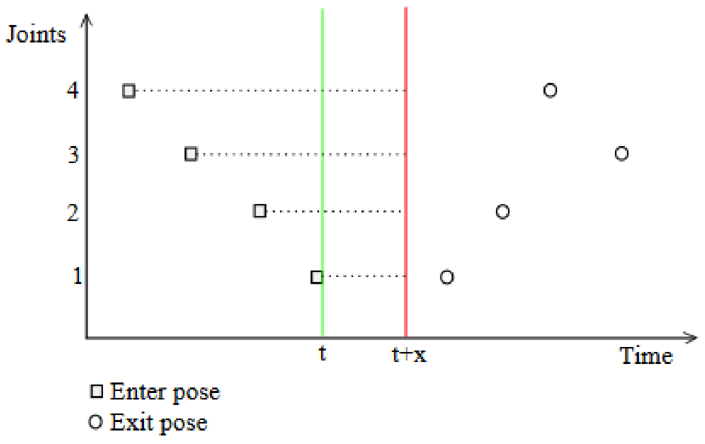


Figure 4.2: Two subsequent poses with overlap of joint regions of 2 different poses

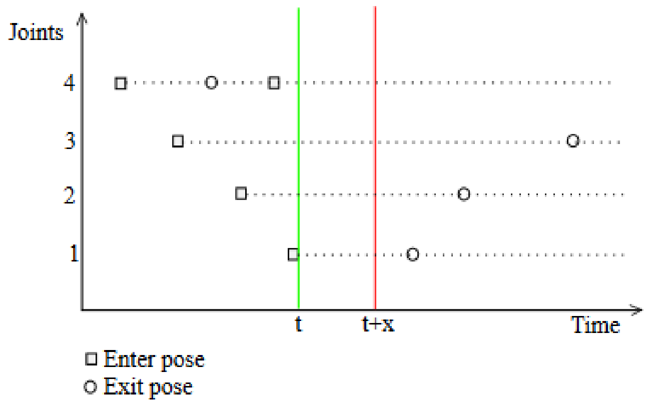
4.2.1 Pose Recognition

To match a pose all joints have to be near a certain position at the same time. By checking if a joint is inside the joint region which was generated we know if the joint is close enough to be matched. All joints have to be in their joint region for at least a small amount of time, for example 100 milliseconds, for the pose to be matched. Matching a pose is not as simple as just keeping track of when all joints entered their joint regions. When a pose matching is in progress it can be invalidated by a joint that enters its region but goes out of the region before the other joints were in their regions for the given time interval. Several situations can cause the pose to be invalidated, Figure 4.3 shows two possible situations where the pose is matched and Figure 4.3 shows two situations where the pose is not matched.

The ideal case is demonstrated in figure 4.3a : To match a pose each joint has to enter its region and all joints have to be in that region for some time. This can be controlled by keeping track of all points in time where a joint enters its region. When the last joint enters its region (t) a timer is started to make sure all joints stay in their regions for a small amount of time (x). When that time has passed (t+x) we check if the joints did not leave their respective regions and recognize the pose. If one or more joints left their region before the time has passed (t+x), the pose was not performed and the matching will restart when the joint enters its region again. This means a joint can enter and exit several times and the pose can still be matched afterwards which makes it more robust to jitter. Such an example is explained in figure 4.3b, where joint 4 enters its region, leaves it and enters again. This is allowed since at time t all joints are in their respective region at the same time. To accommodate for this exception we have to keep track of the time on which a joint entered its region last, instead of keeping track of all times for each joint. This means the situation depicted in 4.3b will be recognized as performing the pose, since all joints are in their region from time t until time t+x.



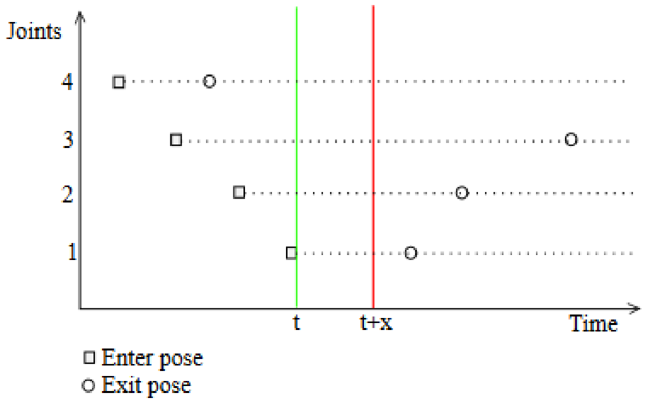
(a) All joints enter their regions and stay there



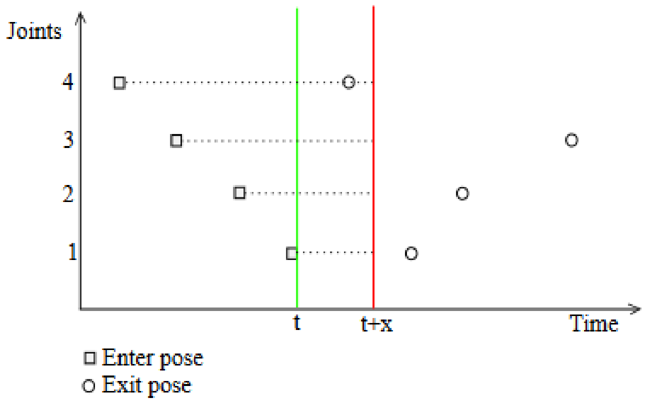
(b) Joint 4 enters its region, exits and enters again

Figure 4.3: Pose recognition with matched poses

Two situations where the pose is not matched are shown in figure 4.4. Both are similar cases, since they involve a joint leaving its position. Figure 4.4a shows a situation where joint 4 enters a position but leaves it before the other joints are in place. Since at no point in time all joints are in position the pose is not matched. Figure 4.4b shows a similar case, only the joint exits its position after all joints are in place, but before the time limit (t+x) has passed.



(a) Joint 4 exits its region before other joints are in place



(b) Joint 4 exits its region before the time limit has passed

Figure 4.4: Pose recognition with not matched poses

4.2.2 Pose Confidence

As explained earlier, a pose is recognized when all joints are within a certain region for a certain time interval. These regions are determined by an ellipsoid shape that is automatically generated. However, checking whether a pose matches in an ellipsoid shape only provides a binary value. Either the pose is in or not. In order to provide a pose confidence level we divide the joint regions into different sections according to how close a joint is to the center of the region. Each region is divided into a number of sections which all have a joint confidence value. The closer the section is to the center of the joint region, the better the joint confidence. The joint confidence value is a number between 0 and 1 which changes each time a section with a better confidence value is entered. The pose confidence value is calculated by taking the mean of all joint confidences. This gives us a pose confidence value between 0 and 1 which indicates how well a pose was matched.

4.3 Combining Poses

To combine these poses into a gesture we have to keep track of the time a pose was matched. By comparing these times we can easily see if the poses were matched in the right order, thus if the gesture was performed correctly. If one pose of the pose sequence is not matched, or the poses were matched in the wrong order, the gesture was not performed correctly and it is not recognized. When a gesture is recognized we calculate a gesture confidence value by calculating the mean of all pose confidence values of the pose sequence. This gesture confidence value gives an indication on how good the gesture was matched overall.

5 Automated Gesture Candidates

5.1 Poses

Before we generate the sequence of poses needed to recognize a gesture, the user can select which joints are important and should be monitored by the gesture recognition system. These joints are added to the set of important joints and are the only joints which will have joint regions to be matched. This allows users to customize gesture definitions. For each pose and for each joint in the important joints set, a group of ellipsoids is generated. This group of ellipsoids consists of a number of ellipsoids with different sizes. The ellipsoid with the biggest size defines the joint region, while the other ellipsoids define the sections of the joint region which give better confidence values.
As shown in figure 5.1, the smaller the size of the ellipsoid the better the confidence. A pose is then defined as the collection of the ellipsoids from each joint in the important joints set. To generate the sequence of poses we run through the training sample and generate a pose on the time interval specified by the user. Each pose has an visual representation so users know what the poses look like and can adjust them by transforming the ellipsoids. Since positions are used as the main feature, using absolute positions for the ellipsoids would make the poses position dependent. Therefore we opted to use relative-positions as a feature by placing the ellipsoids relatively towards a parent joint.
Every time a joint enters an ellipsoid of its group of ellipsoids an *EnterEllipsoid* fact is asserted into Midas. This fact enables developers to construct rules which depend on whether a joint entered an ellipsoid. The *EnterEllipsoid* fact keeps track of useful information, such as the time a joint entered an ellipsoid and the confidence value of the entered ellipsoid. When a joint leaves an ellipsoid of its group of ellipsoids, an *ExitEllipsoid* fact is asserted which indicates a joint left an ellipsoid.

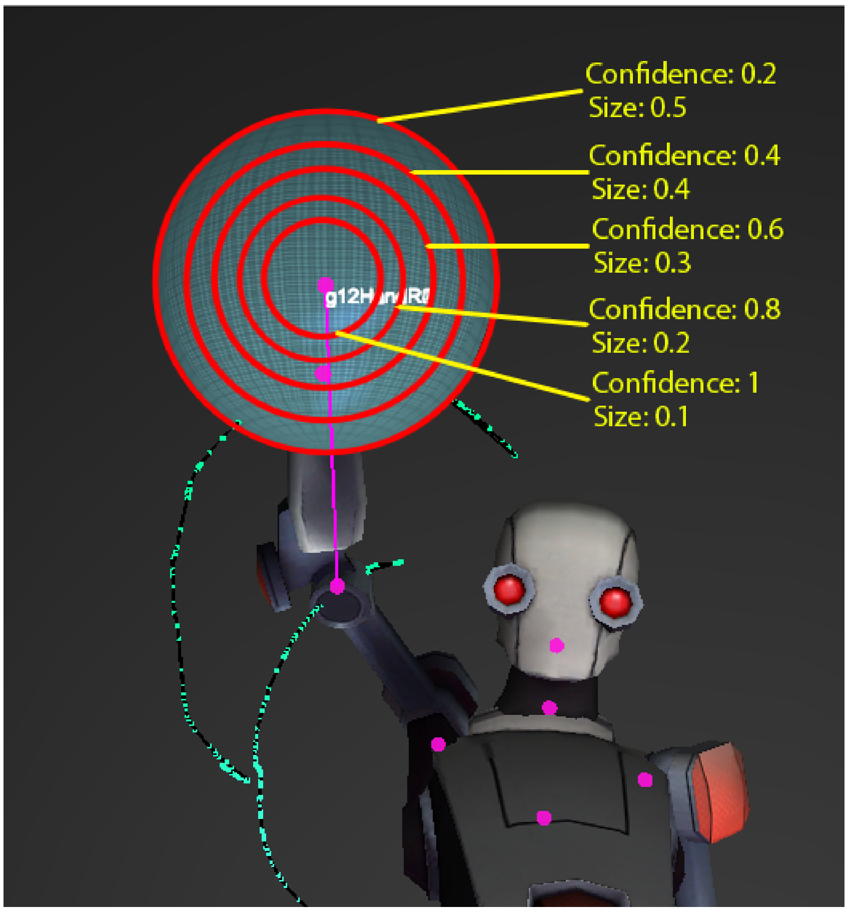


Figure 5.1: 5 ellipsoids on the right hand with different scales and confidence values

5.1.1 Pose Confidence

As mentioned before we generate a group of ellipsoids for each joint to indicate a confidence value. The amount of different ellipsoids with different scales that is generated can be specified by the user. By default each group of ellipsoids contains 5 ellipsoids since this seems to give a good indication about the accuracy of a pose. Smaller ellipsoids have a higher confidence value since they mark a smaller region and are therefore only matched when the pose is performed accurately. Every time a joint enters an ellipsoid from its group of ellipsoids, the confidence level of the newly entered ellipsoid is compared to the confidence level of the previously entered ellipsoid. If the new confidence level is higher, the *EnterEllipsoid* fact is modified with the new information.
These confidence values are always between 0 and 1 and are dependent on the number of ellipsoids in a group of ellipsoids. The smallest ellipsoid always has a confidence value of 1, while the confidence values of the other ellipsoids are calculated by dividing 1 by the number of ellipsoids in a group of ellipsoids. For example a user can opt for 10 ellipsoids which will yield a set A of confidence values where

$$A = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$$

The user can adjust the size of these ellipsoids by setting a maximum scale size. The scale size for all ellipsoids will then be calculated by dividing the maximum scale size by the number of ellipsoids in a group of ellipsoids. This ensures the distance between all ellipsoids is equal, as shown in figure 5.2 where the distance is denoted by x. For example a user can select a maximum scale size of 0.5 and choose to generate 5 ellipsoids which will yield a set B of ellipsoid sizes where

$$B = \{0.1, 0.2, 0.3, 0.4, 0.5\}$$

According to De Roover [5], fuzzy reasoning allows to model our world in a more accurate way. By generating gesture definition with multiple ellipsoids, the gesture definitions allow some fuzziness. Instead of declaring a gesture matched or not matched, we allow a gesture to be matched with a certain degree of truth, which is a more intuitive method since two gestures are almost never performed exactly the same. This allows to create strict or loose gestures with a confidence value. Strict gestures can be defined by choosing a small ellipsoid size, since small ellipsoids require the gesture to be accurately performed. Choosing a big ellipsoid size generates loose gestures, since a certain error in the performed gesture is tolerated. This can be useful in different domains, for example to set the difficulty level of a game.

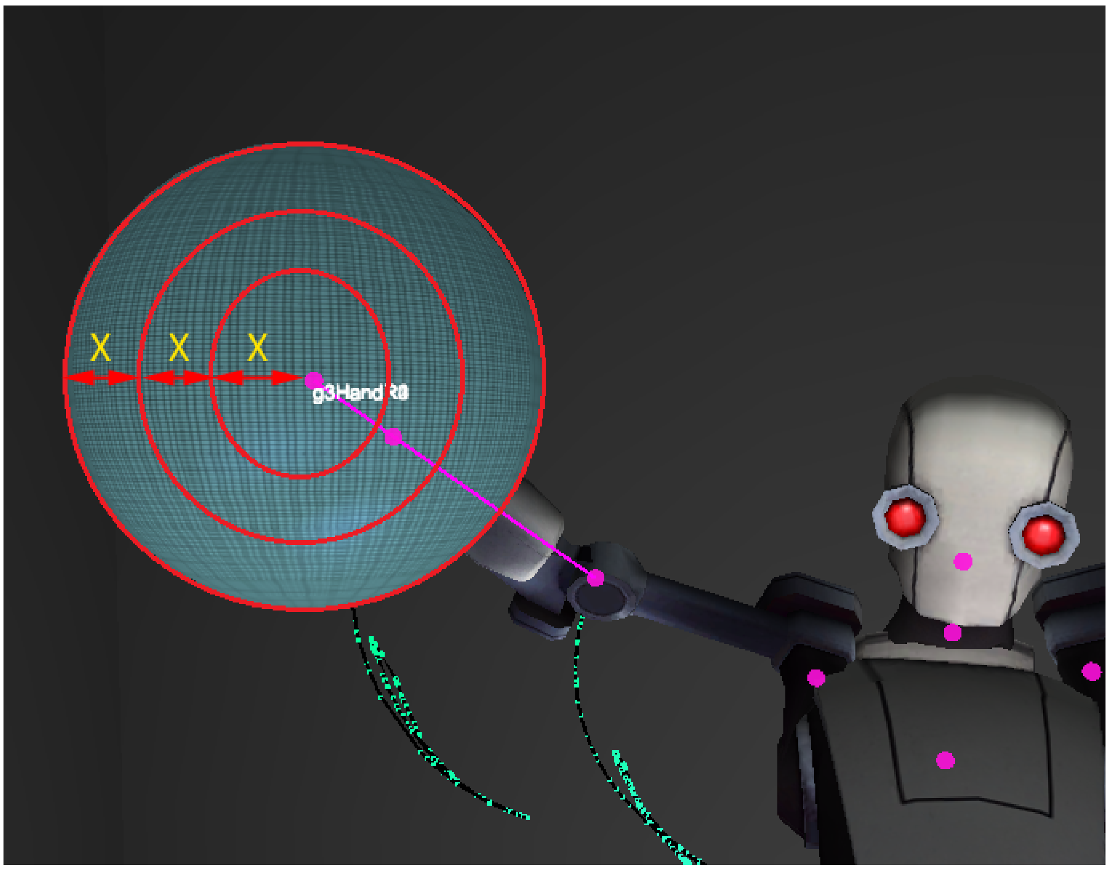


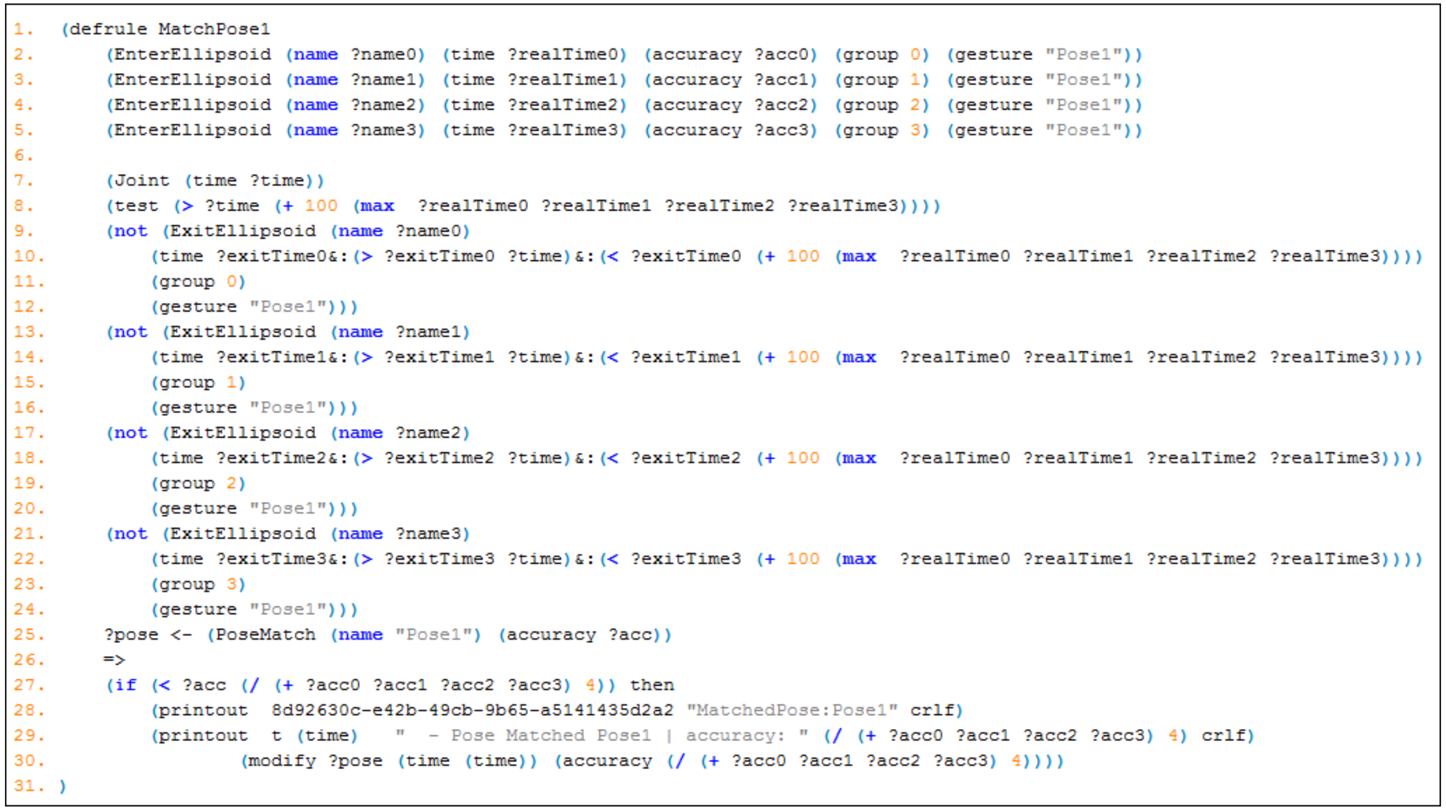
Figure 5.2: Three ellipsoids on the right hand joint with an equal distance between the ellipsoids

5.1.2 Pose Recognition

To recognize a pose we generate declarative rules which implement our strategy as described in section 4. For each pose, we generate a rule that requires the presence of an *EnterEllipsoid* fact for each joint that is part of the important joints set defined by the user. When a joint enters an ellipsoid, an *EnterEllipsoid* fact for that joint and ellipsoid is asserted. If there is an *EnterEllipsoid* fact for each joint, all joints have entered An ellipsoid. When an *EnterEllipsoid* fact is present for are joints, we wait 100 milliseconds. If a joint leaves an ellipsoid, an *ExitEllipsoid* fact is asserted. If there are no *ExitEllipsoid* facts for the joints used in the gesture, all joints still reside within their ellipsoids so the pose is matched.

Figure 5.3 shows the generated declarative rule for recognizing a pose with 4 joints. We need an *EnterEllipsoid* fact for each joint to recognize the pose, so line 2-5 checks if there are 4 *EnterEllipsoid* facts in Midas for the specified pose. Moreover, each of these facts can only be matched by an *EnterEllipsoid* fact which corresponds to an ellipsoid from the group of ellipsoids of a specific joint. For example, the *EnterEllipsoid* fact on line 2 can only be matched by an *EnterEllipsoid* fact which corresponds to an ellipsoid of the group of ellipsoids with id = 0, from pose 'Pose1'. By checking the id of the group of ellipsoids an ellipsoid belongs to along with the name of the gesture, we can make sure each *EnterEllipsoid* fact on lines 2-4 is matched by an ellipsoid from different joints.

When each joint has an *EnterEllipsoid* fact we wait 100 milliseconds, starting from the time the last *EnterEllipsoid* fact was asserted. This wait is enforced by the test at line 8. When these 100 milliseconds have passed we have to make sure no joint has left the ellipsoid it entered since we started waiting. This is enforced by lines 9-24 where we check that there is no *ExitEllipsoid* fact which was asserted after the *EnterEllipsoid* fact for the same joint was asserted and before the 100 milliseconds were over. We ask Midas to check that there is no *ExitEllipsoid* fact of the ellipsoids which have the same name as the *EnterEllipsoid* facts. If there is an *ExitEllipsoid* fact with the same name we check if the time the *ExitEllipsoid* took place was at least 100 milliseconds after the last *EnterEllipsoid* fact was registered or before the *EnterEllipsoid* was asserted. If these conditions are satisfied the pose is matched and the code below the '=>' on line 26 is executed. Since we only need the pose which has the best confidence we check whether the just matched pose has a better confidence than the previously matched pose on line 27. If so feedback is given to the user in the form of a text message on lines 28 - 29 and the *PoseMatch* fact is modified with a new recognition time and confidence value on line 30.



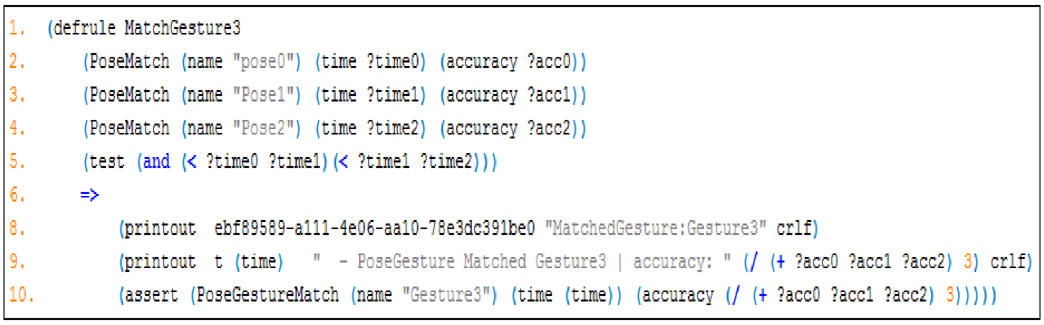
```
(defrule MatchPose1
    (EnterEllipsoid (name ?name0) (time ?realTime0) (accuracy ?acc0) (group 0) (gesture "Pose1"))
    (EnterEllipsoid (name ?name1) (time ?realTime1) (accuracy ?acc1) (group 1) (gesture "Pose1"))
    (EnterEllipsoid (name ?name2) (time ?realTime2) (accuracy ?acc2) (group 2) (gesture "Pose1"))
    (EnterEllipsoid (name ?name3) (time ?realTime3) (accuracy ?acc3) (group 3) (gesture "Pose1"))

    (Joint (time ?time))
    (test (> ?time (+ 100 (max  ?realTime0 ?realTime1 ?realTime2 ?realTime3))))
    (not (ExitEllipsoid (name ?name0)
        (time ?exitTime0&:(> ?exitTime0 ?time)&:(< ?exitTime0 (+ 100 (max  ?realTime0 ?realTime1 ?realTime2 ?realTime3))))
        (group 0)
        (gesture "Pose1")))
    (not (ExitEllipsoid (name ?name1)
        (time ?exitTime1&:(> ?exitTime1 ?time)&:(< ?exitTime1 (+ 100 (max  ?realTime0 ?realTime1 ?realTime2 ?realTime3))))
        (group 1)
        (gesture "Pose1")))
    (not (ExitEllipsoid (name ?name2)
        (time ?exitTime2&:(> ?exitTime2 ?time)&:(< ?exitTime2 (+ 100 (max  ?realTime0 ?realTime1 ?realTime2 ?realTime3))))
        (group 2)
        (gesture "Pose1")))
    (not (ExitEllipsoid (name ?name3)
        (time ?exitTime3&:(> ?exitTime3 ?time)&:(< ?exitTime3 (+ 100 (max  ?realTime0 ?realTime1 ?realTime2 ?realTime3))))
        (group 3)
        (gesture "Pose1")))
    ?pose <- (PoseMatch (name "Pose1") (accuracy ?acc))
    =>
    (if (< ?acc (/ (+ ?acc0 ?acc1 ?acc2 ?acc3) 4)) then
        (printout  8d92630c-e42b-49cb-9b65-a5141435d2a2 "MatchedPose:Pose1" crlf)
        (printout  t (time)    "  - Pose Matched Pose1 | accuracy: " (/ (+ ?acc0 ?acc1 ?acc2 ?acc3) 4) crlf)
            (modify ?pose (time (time)) (accuracy (/ (+ ?acc0 ?acc1 ?acc2 ?acc3) 4))))
)
```

Figure 5.3: Generated rule to recognize a pose

5.2 Gesture Recognition

For recognizing the complete gestures, we generate a rule that combines the *Pose-Match* facts which are asserted by the code snippets that match the poses. In this rule, we also test whether the poses were performed in the right order. Figure 5.4 shows an example of a declarative rule for a gesture which consists of 3 poses. Line 2-4 check whether a *PoseMatch* fact was asserted for each pose. If all poses are matched we can easily know if they were performed in the right order by comparing the time at which they were matched, as is done on line 5. When the poses are matched in the right order, the gesture is matched and on line 10 a *PoseGestureMatch* fact is asserted in Midas with a confidence value which is the mean of the pose confidence values of the 3 poses used in the gesture.



```
1.  (defrule MatchGesture3
2.      (PoseMatch (name "pose0") (time ?time0) (accuracy ?acc0))
3.      (PoseMatch (name "Pose1") (time ?time1) (accuracy ?acc1))
4.      (PoseMatch (name "Pose2") (time ?time2) (accuracy ?acc2))
5.      (test (and (< ?time0 ?time1)(< ?time1 ?time2)))
6.      =>
8.          (printout  ebf89589-a111-4e06-aa10-78e3dc391be0 "MatchedGesture:Gesture3" crlf)
9.          (printout  t (time)   "  - PoseGesture Matched Gesture3 | accuracy: " (/ (+ ?acc0 ?acc1 ?acc2) 3) crlf)
10.         (assert (PoseGestureMatch (name "Gesture3") (time (time)) (accuracy (/ (+ ?acc0 ?acc1 ?acc2) 3)))))
```

Figure 5.4: Generated rule to recognize a gesture with 3 poses

6 Evaluation

To evaluate the robustness of our approach, we implemented several gestures with different ellipsoid sizes. We need gestures which have some similarity to test our approach for false positives. We used dance gestures since they can be quite similar and technical, which is ideal for our implementation since we can model each technical stance separately by a pose.

The precision requirement is implemented by letting the user choose an ellipsoid size. These sizes define a threshold at which errors while performing a gesture are acceptable. The ellipsoids of each pose can be moved, scaled and rotated if desired by the user. This allows to fix errors in the generated definition or to define a new gesture definition which can be completely different from the generated one. Constraints can also be refined by changing the generated declarative code. This provides the extensibility needed for an automated gesture recognition system.
Each pose has a visual and a declarative representation. The visual representation consists of the ellipsoids in VolTra and aids the developer with debugging and to understand the gesture definition, while the declarative representation represented by pose and gesture rules ensures the extensibility of the generated gesture definition. Since users have to choose which joints are important for a gesture, customizability is implemented. Feedback is given to the developer using the visual representation and text messages.

6.1 Method

In order to test our gesture recognition approach, we created a data set of 6 gestures each performed by 9 different subjects, resulting in 54 samples. The subject group consists of 5 women and 4 men, the youngest subject is 17 years while the oldest is 46. The subjects also vary in height and weight. The samples were recorded using a Microsoft Kinect camera, Kinect SDK version 1.0.3.191 and OSCeleton[13]. One expert set an example for the gesture, upon which the 9 subjects copied the gesture performed by the expert. All samples were recorded at the same time in a room with sufficient lightning. These samples are loaded into VolTra where we select the starting and endpoint of the gesture using the reference sample (of the expert) to generate the gesture definition.
The performed dance gestures are:

1. Y
2. M
3. C
4. A
5. Disco
6. Egyptian

[13] `https://github.com/Zillode/OSCeleton-KinectSDK`

The first four gestures: Y,M,C and A are dance moves of the famous song YMCA (1978), from the American disco group Village People. The fifth gesture, Disco, is a famous disco dance move and the sixth gesture , Egyptian, is a dance move from the famous song Walk like an Egyptian by The Bangles.
For the YMCA gestures the hands and arms have to be hold in such a way that they depict the letter the move stands for, as shown in 1 - 4 in Figure 6.1. For the Disco gesture the subject has to put his left hand on his right hip, after which the left hand is moved up in a diagonal fashion to the left until it points to the sky. This move is also known as ”The Point”. For the Egyptian dance move the arms have to be bent in a specific way before moving the hands forward and backwards. Figure 6.1 presents the 6 gestures in VolTra, as performed by the expert. We choose these dance moves since they have a lot of similarity (for example the A and M move) and most people are familiar with the dances, which should make is easier to perform these gestures.

We used our data set of 54 samples to test each gesture definition with 3 different ellipsoid sizes: 0.3 , 0.4 and 0.5. We also evaluated other ellipsoid sizes but found these values to be optimal. A smaller ellipsoid size than 0.3 makes the ellipsoids very small and almost impossible to match, while an ellipsoid size larger than 0.5 makes the ellipsoids very big, matching every gesture in our dataset. We asked 3 different subjects than the aforementioned 9 subjects to perform a sequence of movements which resembles some of the gestures, but do not match hem. As shown in table 6.1, this random data shows that an ellipsoid size of 0.5 already poses a small risk for false positives. A bigger ellipsoid size would only increase this risk, while no false positives were found when conducting tests with the same random samples and ellipsoid sizes of 0.3 and 0.4.

To generate our gesture definition we used a timespan of 1 second between each generated pose for ellipsoid sizes of 0.5, 900 milliseconds for ellipsoid sizes of 0.4 and 800 milliseconds for ellipsoid sizes of 0.3. There is a difference in time for each ellipsoid size since poses with smaller ellipsoids have to be generated closer together in order to minimize the space between joint regions of subsequent poses. Our evaluation gestures consist of 2 or 3 poses, depending on the gesture. For the Egyptian gesture 2 poses are sufficient since only the hands need to move while the elbows stay in the same place, however for the Disco gesture we need 3 poses to rule out false positives.

To calculate a confidence level of the performed gestures, we calculate the mean of the best matched ellipsoids of each joint. Per joint we used 5 ellipsoids, each with a different confidence level. We choose to use 5 ellipsoids since using more ellipsoids makes the smallest ellipsoid very small and almost impossible to match. Using less ellipsoids will relax the confidence value, resulting in a high confidence level for badly performed gestures.

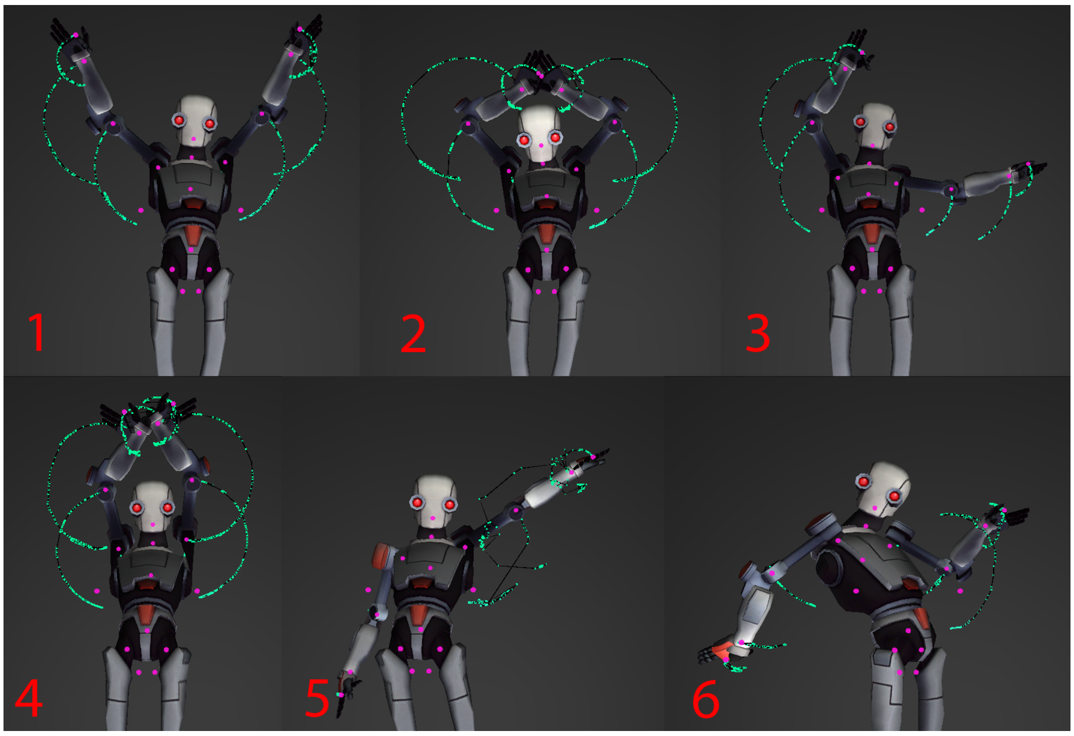


Figure 6.1: Overview of 6 dance gestures

Gesture	Sample	Confidence
A	Random 1	0.75
Disco	Random 3	0.667
Total false positives		3.70%

Table 6.1: False positives for ellipsoid size 0.5

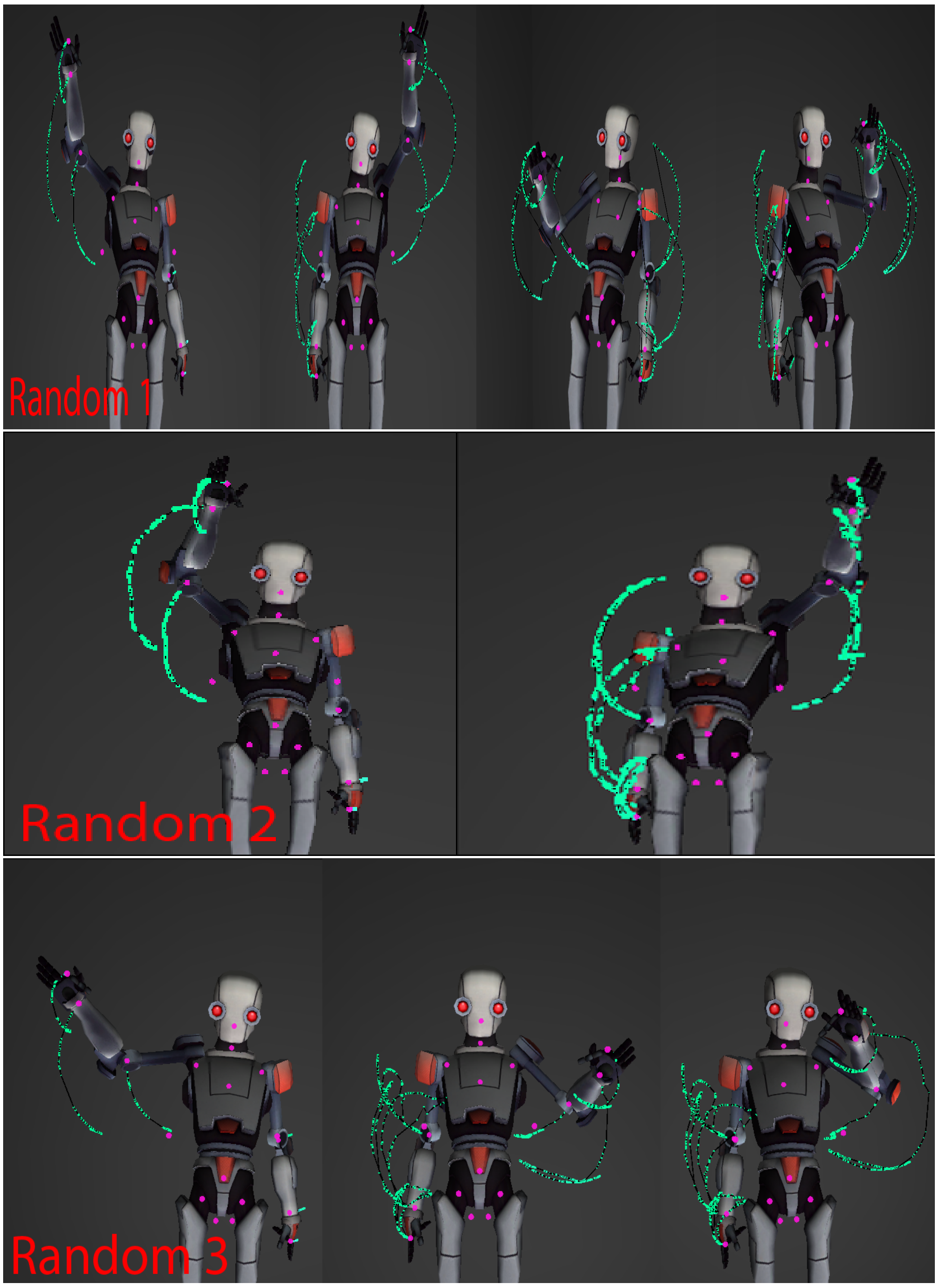


Figure 6.2: Overview of 3 random samples

6.2 Results

An overview of the used gesture definitions is given in figures 6.4 and 6.5. For each gesture definition, the generated poses are displayed.
These gesture definitions are the result of our approach used on the reference samples. Whether a certain gesture definition is too strict or too general depends largely on the size of the gesture ellipsoids. To determine the ideal size, we generated each gesture 3 times, each time with different ellipsoid sizes. This also allows us to see how the scale of the ellipsoids influences the gesture accuracy. A bigger ellipsoid size means it is more likely for a joint to match a ellipsoid, whereas gestures with a smaller ellipsoid size are more difficult to match. This implies we can easily adopt the strictness of our gestures to the needs of the user. By merely changing the ellipsoid size we can make the gesture definitions very strict or very loose, which allows our approach to be used in different situations. For a fun game, the gesture definitions can be very loose with a big ellipsoid size, whereas for a technical application the gesture definitions can become very strict with a small ellipsoid size.
Our evaluation gave us 162 confidence levels (54 samples evaluated with 3 different gestures) which are shown in table 6.4. These confidence levels give us a measure of how accurate the gesture performed by a subject resembles the original gesture performed by the expert. Sometimes a gesture is not recognized, for these misses we fill the table with a zero. These confidence values represent the matched gestures without manual refinement of the gesture definition, since the confidence of a refined definition depend on how it is refined (rescaled or moved).

Table 6.2 shows the amount of misses and matches of our evaluation. We can clearly see that a strict gesture (ellipsoid size 0.3) is harder to match than a loose gesture (ellipsoid size 0.5). Since our subjects are not expert dancers, the recognition rate for strict gestures is lower than for loose gestures.

	Recognition rate		
ellipsoid size	0.3	0.4	0.5
Y	77.78%	100%	100%
M	55.56%	77.78%	88.88%
C	44.44%	88.88%	100%
A	55.56%	77.78%	100%
Disco	77.78%	100%	100%
Egyptian	100%	100%	100%

Table 6.2: Recognition rate of each gesture per ellipsoid size

When a gesture is missed we manually adjust the gesture definition to match the missed sample and record the time it takes us to do so. To adjust the definition we simply need to move the ellipsoids to a different location in the 3D plane. Table 6.3 shows the approximate time in seconds it takes us to manually alter a gesture definition to match all samples and the time it took us to manually create the same gesture definitions. When manually creating these definitions the default ellipsoid size was 0.5. For other ellipsoid sizes the size had to be manually adjusted. In some cases, like the Y gesture for a gesture definition with ellipsoid sizes of 0.3, all misses have the same cause. This happens when there is a difference between how the expert performs a gesture and how other subjects perform the same gesture. For example, Figure 6.3 shows a gesture which is performed correctly, but the right hand joint misses an ellipsoid of the generated gesture definition. These misses are easily fixed, since we can just translate the corresponding ellipsoids to match the joint for one sample, and all misses with the same cause are recognized. The time investment to correct these gestures is rather small since one ellipsoid manipulation solves multiple misses.
Misses can also have very specific causes, which are impossible to solve by simply adjusting the gesture definition for one sample. These misses have to be fixed sample by sample, which increases the time to manually correct these gestures.

	Automated with refinement			Manual		
Ellipsoid size	0.3	0.4	0.5	0.3	0.4	0.5
Y	22	0	0	123	132	108
M	122	42	15	136	142	96
C	23	10	0	102	113	87
A	107	50	0	146	135	117
Disco	69	0	0	97	106	65
Egyptian	0	0	0	95	101	89

Table 6.3: Manual and automated development time in seconds per ellipsoid size and gesture for 9 samples

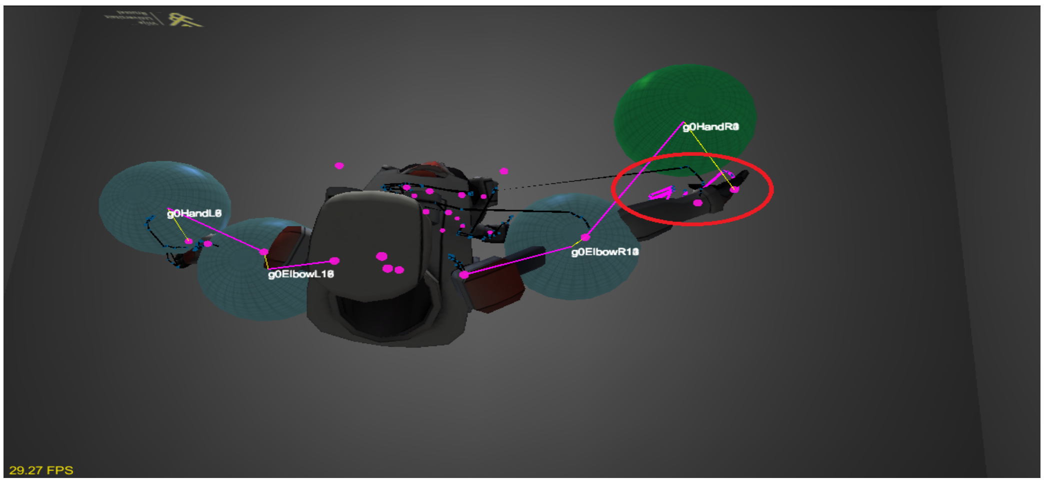


Figure 6.3: Right hand joint misses a ellipsoid of the gesture definition

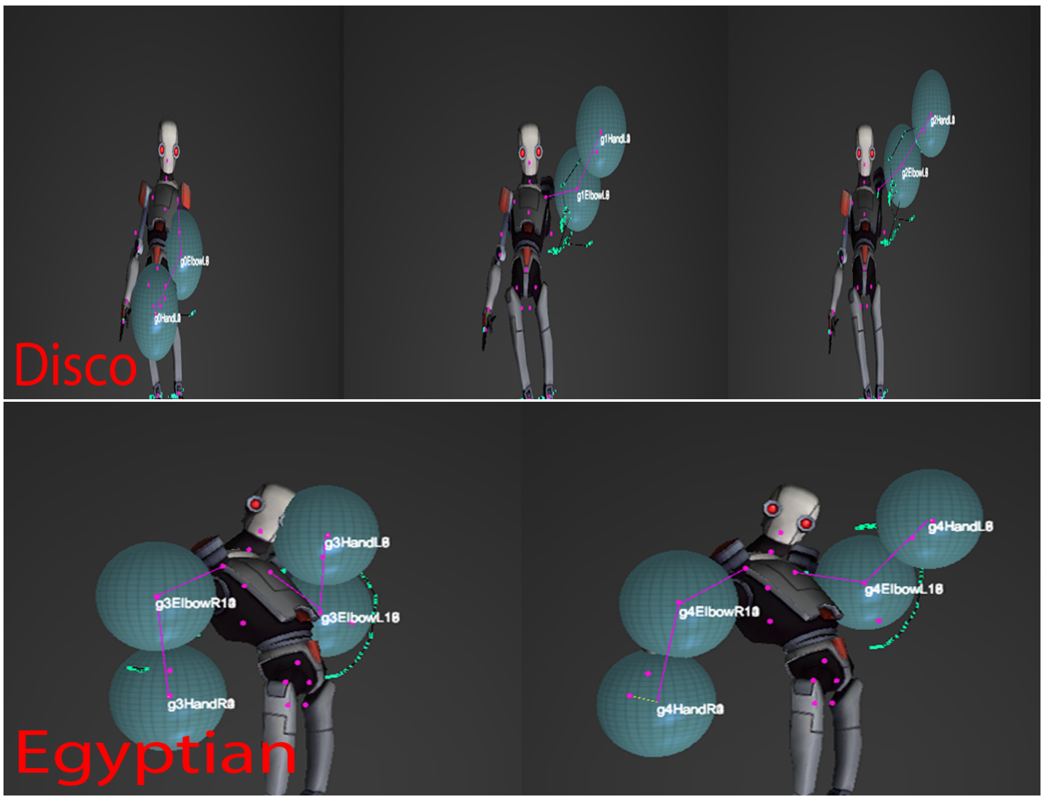


Figure 6.4: Disco and Egyptian gesture definitions with ellipsoid size 0.4. Each row corresponds to a gesture definition.

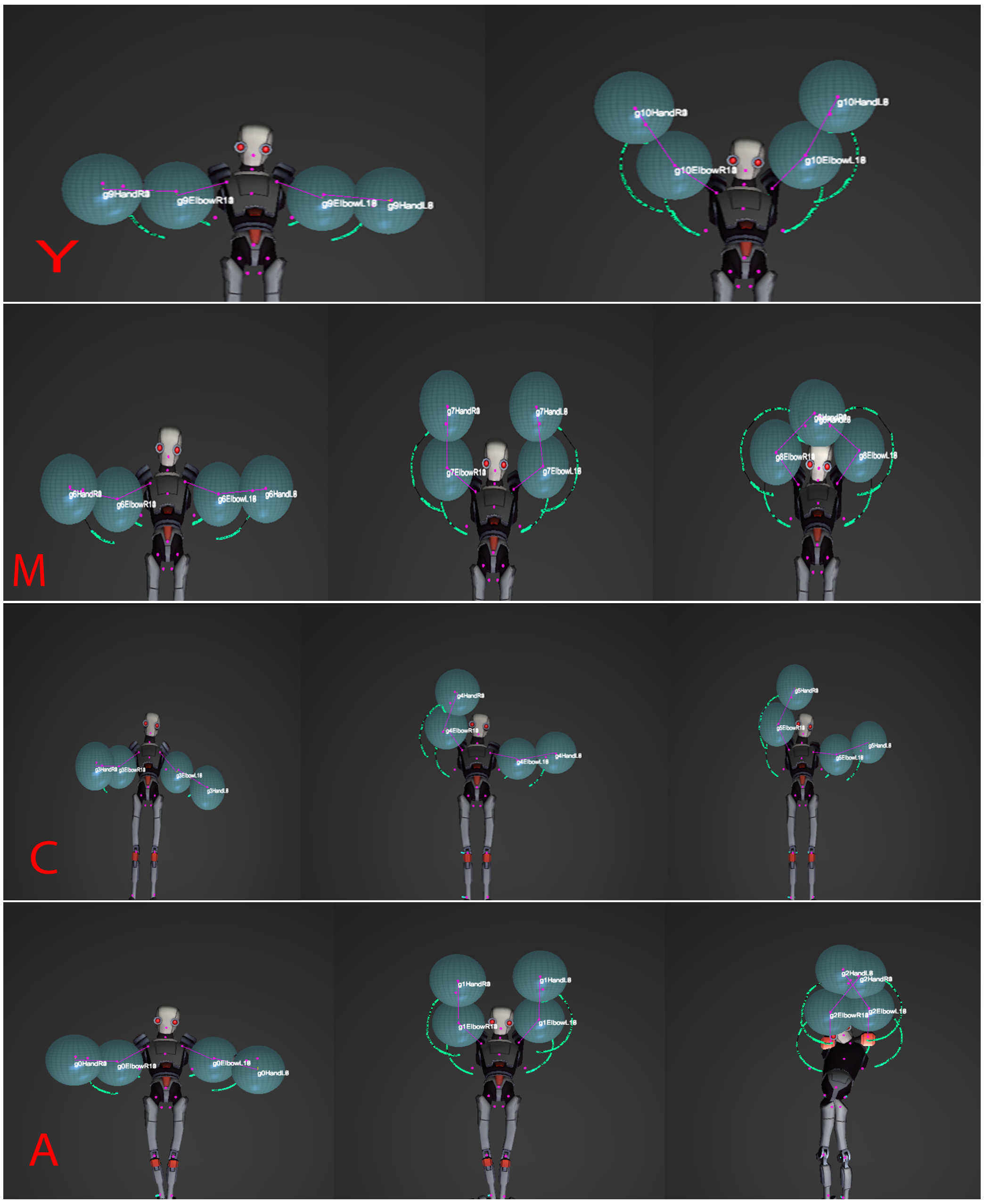


Figure 6.5: YMCA gesture definitions with ellipsoid size 0.4. Each row corresponds to a gesture definition.

	Confidence		
Scale	0.3	0.4	0.5
Subject 1	0.625	0.75	0.775
Subject 2	0.85	0.875	0.95
Subject 3	0.75	0.875	0.875
Subject 4	0.85	0.925	0.9
Subject 5	0.7	0.8	0.85
Subject 6	0	0.7	0.7
Subject 7	0.675	0.8	0.9
Subject 8	0	0.7	0.725
Subject 9	0.85	0.8	0.85
Mean	0.59	0.80	0.84

(a) Y Gesture

	Confidence		
Scale	0.3	0.4	.0.5
Subject 1	0	0.734	0.8
Subject 2	0.667	0.767	0.817
Subject 3	0	0.833	0.867
Subject 4	0.833	0.95	0.967
Subject 5	0.734	0.85	0.9
Subject 6	0.85	0.9	0.93
Subject 7	0	0	0.783
Subject 8	0.667	0.783	0.833
Subject 9	0	0	0
Mean	0.417	0.646	0.766

(b) M Gesture

	Confidence		
Scale	0.3	0.4	.0.5
Subject 1	0.7	0.733	0.817
Subject 2	0	0.6	0.8
Subject 3	0	0.767	0.867
Subject 4	0.93	0.95	1
Subject 5	0	0.717	0.8
Subject 6	0	0	0.83
Subject 7	0.7	0.767	0.883
Subject 8	0.683	0.8	0.85
Subject 9	0	0.75	0.767
Mean	0.335	0.676	0.846

(c) C Gesture

	Confidence		
Scale	0.3	0.4	.0.5
Subject 1	0.6	0.7	0.75
Subject 2	0.667	0.783	0.867
Subject 3	0	0.667	0.75
Subject 4	0	0.783	0.817
Subject 5	0	0	0.75
Subject 6	0	0	0.633
Subject 7	0.783	0.883	0.9
Subject 8	0.733	0.767	0.85
Subject 9	0.767	0.717	0.8
Mean	0.393	0.583	0.697

(d) A Gesture

	Confidence		
Scale	0.3	0.4	.0.5
Subject 1	0	0.767	0.9
Subject 2	0.8	0.8	0.867
Subject 3	0.9	0.967	0.933
Subject 4	0.8	0.7	0.833
Subject 5	0.767	0.8	0.867
Subject 6	0.567	0.733	0.833
Subject 7	0.8	0.7	0.8
Subject 8	0	0.9	0.7
Subject 9	0.967	0.93	1
Mean	0.622	0.81	0.86

(e) Disco Gesture

	Confidence		
Scale	0.3	0.4	.0.5
Subject 1	0.625	0.8	0.825
Subject 2	0.8	0.925	1
Subject 3	0.9	0.95	1
Subject 4	0.85	0.925	0.925
Subject 5	0.8	0.85	0.925
Subject 6	0.875	0.925	0.975
Subject 7	0.65	0.825	0.85
Subject 8	0.875	1	1
Subject 9	0.95	1	1
Mean	0.81	0.911	0.944

(f) Egyptian Gesture

Table 6.4: Confidence levels

7 Conclusion and Future Work

7.1 Conclusion

We started with a overview of requirements which are beneficial for automated gesture recognition systems while motivating why these requirements are important. Related work was discussed with the requirements for automated gesture recognition in mind. Our approach is presented which allows for automated full-body gesture recognition while maintaining the specified requirements which enhance automated gesture recognition systems. An implementation of this approach is given as an extension of VolTra.
Our approach counters shortcomings of existing automated gesture recognition methods:

- The generated gestures allow a great amount of customization by the user after they have been generated. This improves the usability since a gesture definition with flaws can simply be adjusted to fix the errors, while other methods require the gesture definition to be recreated.

- Generated gestures have a comprehensible representation which helps users to understand the generated gesture definition and debug them. Users are also provided with a confidence value which gives an indication of how well a gesture was matched.

- Only a single training sample is needed to generate a gesture definition. This ensures users can add gestures in an easy and quick way.

- Control over the gesture definition stays with the user, which can use several parameters to adjust the gesture definition like ellipsoid size, time interval between poses and joint selection. This allows users to choose how strict or loose a gesture should be.

7.2 Future Work

7.2.1 Generate time between poses

Instead of letting the user select a time interval for pose generation, it could be deducted from the gesture and the chosen ellipsoid size. As mentioned in chapter 4 the time interval on which poses are generated should make sure the joint regions of each joint in subsequent poses are very close or partially overlap. This means we could deduct when a pose should be created by calculating the euclidean distance between 2 positions of a joint. Since ellipsoids are generated with the joint at the center, the ellipsoids should partially overlap when the euclidean distance between 2 joint positions is smaller than the diameter of the ellipsoids.
The first pose could be generated at the starting point of the gesture, which is indicated by the user. The next pose should then be generated when the euclidean distance between the position of a joint in the last generated pose and the current position of that joint is somewhat smaller than the diameter of the generated ellipsoid.

7.2.2 Improve Confidence Value

The confidence of pose and gesture matches can be calculated in a smarter way than calculating the mean of joint confidence values. Here are some suggestions on how to improve the confidence values:

- Users could add weights to more important joints so a weighted mean can be calculated for the confidence value. This would allow users to select which joints are very important for the gesture and produces a confidence value which reflects how well the gesture was matched in a more accurate way then a normal mean.
- The pose confidence value could be influenced by whether joints enter their joint regions at the same time. If all joints enter their joint region within a small time interval, the confidence value could be higher. If a joint enters its joint region some seconds after the other joints entered their regions the confidence could be smaller, depending on what the user desires.
- If desired by the user, the confidence value could also take the performance speed of gestures into account. The time at which poses are matched is already registered so the performance speed of the gesture could be derived from this data.

7.2.3 Automated joint selection

Instead of letting users manually select which joints are important and should be included in the gesture definition, a suggestion for these joints could be derived from the training sample. By analysing the training sample we could derive which joints are important for the gesture. Joints which barely move are probably not included in the gesture, while joints which cover a large area of the 3D space are probably very important in the gesture. The joints which are derived from the sample should only be a suggestion to the user, who should be able to add or remove joints from the important joints set.

References

[1] Frederic Bettens and Todor Todoroff. Real-Time DTW-based Gesture Recognition External Object for MAX/MSP and Puredata. *Proc. SMC*, 9:30–35, 2009.

[2] Frederic Bevilacqua, Bruno Zamborlin, Anthony Sypniewski, Norbert Schnell, Fabrice Guedy, and Nicolas Rasamimanana. Continuous Real-Time Gesture Following and Recognition. In *Gesture in Embodied Communication and Human-Computer Interaction*, pages 73–84. Springer, 2010.

[3] Andrea Corradini. Dynamic Time Warping for off-line Recognition of a Small gesture vocabulary. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 82–89. ICCV, 2001

[4] Brecht De Rooms. A Development Environment for Extracting and Defining 3d Full-Body Gestures. Master's thesis, VUB, August 2012.

[5] Coen De Roover. Incorporating Dynamic Analysis and Approximate Reasoning in Declarative Meta-programming to Support Software Reengineering. Master's thesis, VUB, May 2004.

[6] Adrien Delaye, Rafik Sekkal, and Eric Anquetil. Continuous Marking Menus for Learning Cursive Pen-based Gestures. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, pages 319–322. ACM, 2011.

[7] Paul Doliotis, Alexandra Stefan, Christopher McMurrough, David Eckhard, and Vassilis Athitsos. Comparing Gesture Recognition Accuracy using Color and Depth Information. In *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*, page 20. ACM, 2011.

[8] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot Learning of Object Categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[9] Pragati Garg, Naveen Aggarwal, and Sanjeev Sofat. Vision Based Hand Gesture Recognition. *World Academy of Science, Engineering and Technology*, 49(1):972–977, 2009.

[10] Lode Hoste, Brecht De Rooms, and Beat Signer. Declarative Gesture Spotting Using Inferred and Refined Control Points. In *Proceedings of ICPRAM 2013, 2nd International Conference on Pattern Recognition Applications and Methods*, Barcelona, Spain, Februari 2013.

[11] Maria Karam. *PhD Thesis: A Framework for Research and Design of Gesture-based Human-Computer Interactions*. PhD thesis, University of Southampton, 2006.

[12] Adam Kendon. *Gesture: Visible Action as Utterance*. Cambridge University Press, 2004.

[13] C Keskin, A Erkan, and L Akarun. Real-Time Hand Tracking and 3D Gesture Recognition for Interactive Interfaces using HMM. *ICANN/ICONIPP*, 2003:26–29, 2003.

[14] T Kuroda, Y Tabata, A Goto, H Ikuta, and M Murakami. Consumer Price Data-glove for Sign Language Recognition. In *Proceedings of the 5th International Conference on Disability, Virtual Reality Assoc. Tech., Oxford, UK*, pages 253–258, 2004.

[15] Marcus Vinicius Lamar. *Hand Gesture Recognition using T-CombNET-A Neural Network Model Dedicated to Temporal Information Processing*. PhD thesis, Nagoya Institute of Technology, 2001.

[16] Hong Li and Michael Greenspan. Continuous Time-varying Gesture Segmentation by Dynamic Time Warping of Compound Gesture Models. In *International Workshop on Human Activity Recognition and Modelling (HARAM)*, pages 35–42, 2005.

[17] Manar Maraqa and Raed Abu-Zaiter. Recognition of Arabic Sign Language (arsl) using Recurrent Neural Networks. In *First International Conference on the Applications of Digital Information and Web Technologies*, pages 478–481. IEEE, 2008.

[18] Kouichi Murakami and Hitomi Taguchi. Gesture Recognition using Recurrent Neural Networks. In *Proceedings of the Special Interest Group on Computer-Human Interaction Conference on Human Factors in Computing Systems: Reaching Through Technology*, pages 237–242. ACM, 1991.

[19] Michalis Raptis, Darko Kirovski, and Hugues Hoppe. Real-time Classification of Dance Gestures from Skeleton Animation. In *Proceedings of the 2011 Association for Computing Machinery SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 147–156. ACM, 2011.

[20] Thomas Schlomer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture Recognition with a wii Controller. In *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, pages 11–14. ACM, 2008.

[21] Christophe Scholliers, Lode Hoste, Beat Signer, and Wolfgang 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.

[22] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-Time Human Pose Recognition in Parts from Single Depth Images. *Communications of the ACM*, 56(1):116–124, 2013.

[23] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. Gestures as Point Clouds: a $ p Recognizer for User Interface Prototypes. In *Proceedings of the 14th Association for Computing Machinery International Conference on Multimodal Interaction*, pages 273–280. ACM, 2012.

[24] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. Gestures without Libraries, Toolkits or Training: a $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th annual Association for Computing Machinery, Symposium on User Interface Software and Technology*, pages 159–168. ACM, 2007.

[25] Junji Yamato, Jun Ohya, and Kenichiro Ishii. Recognizing Human Action in Time-sequential Images using Hidden Markov Model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1992.