Gesture recognition in streaming motion data using offline training with a limited training set

Tim K.C. Franssen

Februari 26th, 2013

Abstract

This Master's thesis is about the analysis of motion capture data, focussing on quickly and accurately recognizing arm gestures for use in a virtual infantry training system. We do a comparative study between the SVM and HMM classification approaches, different features (coordinates, motion vectors, a combination of both) and parameters (motion vector offset, cost, gamma, number of states et cetera) that are specific to the application of a training simulation.

We show that gesture classification can be used in a virtual infantry training situation. Less than ten minutes of training data from one instructor is sufficient for classifying nine different gestures from students with an f-measure of 0.65 on average.

This classification can be used for a plethora of applications including scoring students relative to each other, allowing the instructor gesture control over the scenario and as input to artificial intelligent agents.

Foreword

This thesis documents the final project for my Master's degree in Computer Science at the University of Twente in the Netherlands. I have very much enjoyed the Master Human Media Interaction, as it gives a nice overview of a very challenging and creative field of study. The final project has taken me way too long to complete, which I regret but which also makes me twice as pleased to present this thesis.

I would like to thank the following people for allowing me to take this step:

My supervisors Job Zwiers, Ronald Poppe and Steven Wijgerse for much wisdom and patience.

My parents Frits and Irene for their support and for stimulating me to take on an education.

My bold participants: Nico, Stefan, Michelle, Bjorn, Remco, Remco, Willemijn, Ron, Brian, Christina, Evelien and Jacob.

Contents

Chapter 1

Introduction

This Master's thesis is about the analysis of motion capture data, focussing on quickly and accurately recognizing arm gestures for use in a virtual infantry training system. The following text gives an overview of the context for this analysis and concludes with several research questions.

1.1 Social context

The motion capture driven virtual reality system that Enschede-based company re-lion has built is mainly intended to be an instructional tool for the military. New recruits normally train indoor missions in "practice villages" that consist of bare houses, most of which have no floors, no decorations and no furniture. In these villages they learn how to safely and quickly traverse an urban area and locate "hostiles". The SUIT system is intended to partially replace these (expensive) villages by allowing recruits to train in a virtual reality environment, in a gymnasium.

Within the virtual reality environment there are few limits on the kinds of scenarios that can be trained. Unlike in the villages, in virtual reality hostiles can shoot at you and you can shoot back. The simulation allows soldiers to train missions at locations where it is not possible to train in real life, for example in a museum without disturbing any of the real visitors, or abroad before actually flying there. Decoration and furniture are a matter of map design.

The military, TNO research and re-lion are currently evaluating the effectiveness of the virtual training compared to the training in the villages. Having been present at this evaluation we would say that it seems that much of the usability depends on the ability of the instructor to use the added possibilities that the virtual system offers without being disoriented by its virtual nature.

1.2 Application context

The SUIT system consists of an Xsens suit with a heads up display, a plastic replica of a weapon, an ultra wide band positioning system, a central server and one laptop per user. The user walks around in the suit, which registers his or her movements and allows the system to show an accurate 3D reproduction in the simulation. Also, it tracks the movement of the user relative to the lowest point of the body. Keeping the lowest point in the same position while moving the rest of the body around it allows the user to walk in the simulation like one would in real life. An ultra wide band positioning system corrects the drift in the Xsens suit and determines the directionality of the walking movements. The replica weapon that is part of the SUIT system also carries an Xsens sensor, allowing the user to aim in the virtual environment and a working trigger that allows him or her to shoot. The Xsens suit and weapon together allow the user to move about and respond to enemies as one would in a real life situation.

On the software side there is a client-server architecture. An almost headless server is controlled by an operator client (usually on the same machine) with which one can manipulate the simulation, trigger calibration routines, load worlds and otherwise control the system. The users of the system carry around laptops that run another client application which reads the motion capture data from the Xsens suit and weapon, renders a 3D view of the simulation in the HMD of the user, and communicates the position and pose of the user back to the server.

Both the client and server software are written mostly in C++. Much of the interfacing with the user and the scripting of the game engine however take place in a Lua[7] environment, which makes the software less prone to unrecoverable errors. Network communication is being handled by an enterprise service bus.

1.3 Research question

The broader question is whether we can apply existing motion recognition techniques to virtual infantry training. Our goal is to enable an instructor to train a classifier with just a few examples of gestures and then classify streaming motion capture data on-line in near-real time. We could then use this classification to score the students on how well they communicate non-verbally (for example: relative to each other), to give feedback to students or as gesture-based commands for the system.

Although the applications are many, for the purpose of this thesis we restrict ourselves to one scenario. We have conducted a small exploratory experiment where participants were asked to communicate only through arm gestures and otherwise take part in a normal training scenario. This experiment taught us that it is very hard to get participants to restrict themselves to a limited set of gestures, and that such a highly ecologically valid experiment, because of the high amount of noise due to all the movement, poses a great challenge for detection, recognition and even annotation of gestures.

Knowing that, we have focused on a more constrained classification problem: the recognition of single arm gestures, without gesture detection. That is to say, we expect all motion to be gestures, and we try to recognize which gesture. This means that we trade some ecological validity and we move the experiment back to the lab, but we can still find answers to some very interesting parts of the broader question.

Our use cases add a few restrictions to our classification approach. For example, we will have to classify mostly inter-participant, that is; between the instructor and the students. This raises the question how much accuracy we lose by classifying inter-participant. Also, we wish to limit the effort required to train the classification system, allowing us to train the system with just a few minutes of gestures performed by the instructor before a training. This restricts both the amount of training data and the time to process it. So we have to ask ourselves how much frames of motion capture data we really need to train our classifiers with and what impact this has on performance.

This thesis will not focus on the computational complexity of classification. Because we want to classify on-line we may have to limit ourselves in our requirements, for example which features we can use for the sake of processing speed. We will not investigate this part of the broader question.

Nine different gestures have been selected for this thesis. The gestures come from the military handbook for field soldiers to keep the ecological validity high. They were selected from a larger set on the basis that they are large gestures that can be recognized from a distance if executed with one hand. Also, they provide a sufficiently large challenge for classification because some gestures are quire similar, which allows us to draw conclusions for real world applications. We have given each gesture a number and a nickname, see figure 1.1.

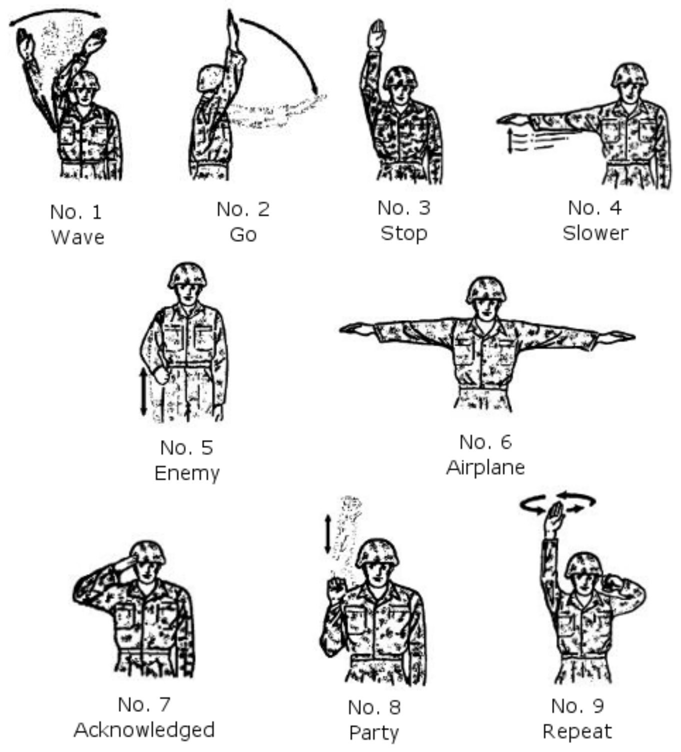


Figure 1.1: *The nine gestures that this research has focussed on*

We will try to answer the following questions with regards to the classification of 3D motion captured gestures:

- Which type of feature is the most information-rich in this application?
- Which classification approach is most suited for this application?
- What is the loss of accuracy when using only the motion capture data from one person as training set (the instructor) versus training on multiple participants?
- What is the influence of the size of the training set on the accuracy? How far can we limit the training effort with minimal performance drop?

Chapter 2

Related work

Modern history of motion analysis starts in 1973 with Gunnar Johansson[9], who showed that humans can recognize human motion from only joint positions, and thus that joint positions alone contain enough information to be able to infer the original motion. Since then this has been shown to be the case for computer analysis as well, in fields ranging from analysing tennis swings using hidden Markov models[24], transcribing the American Sign Language using neural networks[25] and hidden Markov models[19] and even applications in gaming with the Xbox Kinect[16].

2.1 Motion capture

The capture and analysis of human motion is a very active field of study. Most of the research focusses on the capturing part. There are several different ways to capture human motion in a computer: using one or more (digital) cameras, using depth cameras like time-of-flight cameras, using tracking markers in a variety of ways or using inertial sensors.

Digital cameras are popular because they are cheap and readily available, while allowing for non-invasive interaction[12]. The disadvantage of using conventional cameras is that the image analysis takes a lot of processing power and it is hard to prevent depth ambiguity. By combining several digital cameras and using stereo vision for calculating depth cues the accuracy of the detection can be improved, but the complexity and processing power increase. For a recent survey of vision based motion recognition, see [14].

Depth cameras can feed the computer a 3D video stream. A 3D image contains more information and depth cues can be used to improve the detection of the human pose. There are several different types of depth cameras, I will shortly discuss two. First, time-of-flight cameras calculate the depth of a scene by measuring the time that the light takes to get from the camera to the scene and back to the camera. This is accomplished either by modulating the frequency of the light or by flickering the light with a given frequency and adding a shutter to the camera lens[5]. A second type of depth camera is the structured-light 3D scanner. A pattern of light (which may be infra-red) is projected at the target scene. If a camera looks at the same scene under a slightly different angle it will see distortions in the pattern, which can be used to calculate the depth of the image[26]. The Kinect sensor employs this technology[16].

Motion capture using tracking markers requires that marker objects are attached to the joints of a participant. These markers are then tracked in many different ways.

One way is using optical tracking with cameras, like the Vicon system uses[20]. These systems combine reflective markers with an array of cameras, either using visible light or infra-red. The markers can either be identical, requiring the user or the software to connect the markers to the joints in software, or unique using different wavelengths (colours) to make the process easier[22]. Other systems use radio signals to determine the position of the markers, but in general any system that can locate multiple objects in 3D space can be used[23].

Finally, there are inertial motion capture systems. These do not rely on external sensors or cameras. Rather the sensors themselves are placed on the joints of the participants. These sensors use inertial and magnetic cues to detect their movement relative to their starting positions, which allows a computer to keep track of them. Such sensors are also used in the Nintendo Wii controller and most smartphones. Because of the design of an inertial motion capture system the participant always needs to calibrate the system by assuming a default, known pose. After this calibration the freedom of movement is unparalleled by the other systems, though. Because no external sensors are required, participants can walk around freely, even leaving the room or the building, and still be tracked. One such system is the Xsens motion capture system[15].

2.2 Features

Motion capture systems can produce their data in a variety of formats, including joint coordinates, joint rotations, acceleration and others. Having captured the movements of a participant, the next step in the process is to extract features from this data that contain the minimum amount of data and the maximum amount of gesture information. The time to process these features is limited and we also wish to limit the required processing power to a minimum, so we reduce the amount of data. Keeping only the relevant information also helps classification algorithms because there is less noise and more signal in the data.

Campbell et al. give us valuable advice[1][2]. They calculated different feature vectors to see which one would perform best at classifying different gestures (applied to T'ai Chi movements and ballet steps). They concluded that proper design of the features for any gesture recognition system is of great importance. They highlighted the importance of shift and rotation invariance in features, which make features less dependent on situational variables. In the case of gesture recognition these variables also include body shape and size, so shift and rotation invariance are an important part of feature selection.

Jin and Prabhakaran[8] reduce the data to just the amount of activity per region of the body. From this amount of activity they produce a semantic representation of the movement (A for arms, AT for arms and torso, TL for torso and legs, et cetera) and this representation is then aggregated into a histogram of the movement. This can then be used as a signature for search or classification. This approach could be applied to the amount of activity in the shoulder, the elbow and the wrist for the application of gesture recognition. This would greatly reduce the detail in the features, allowing for faster recognition with more confusion.

Classifying with different participants brings the challenge of normalizing data in such a way that similar movement from different participants results in similar data, dispite differing limb sizes and habits. The NATOPS signals database[18][17] uses limb length normalisation to reduce the differences between participants. To accomplish this they calculate joint angles and from these angles they calculate new coordinates with a unit arm length. The resulting coordinates are more easily compared between participants than the original coordinates.

Having normalized the data the next question is: which data will we feed the classifier. Obvious choices are the coordinates of the joints of the left arm or the joint angles of the left shoulder, elbow and wrist. A little less obvious are derivatives of this data: the velocity of the joints or the angular velocities. The NATOPS research[17] has compared these features for gesture recognition. In their application the derivative features clearly outperformed the original features. Joint coordinates performed better than joint angles.

2.3 Classifiers

Classifiers form the last step of the process. A classifier is an implementation of a mathematical model that can label a test data set given a training data set. We must construct a training set of features in known categories that we can label, and feed this set to the classifier. The classifier then builds a representation of each label which makes it possible to apply labels to each new feature and estimate to which category this unknown piece of data belongs.

There are quite a few classifiers that we can choose from, which have been used in gesture or action recognition in the literature[14][21]. However, we can split the classification in two rough categories: direct classifiers and model based classifiers.

Direct classifiers take the motion capture data one frame at a time and try to classify the features as belonging to one class. This is very fast and, depending on the features, can be quite accurate. Model based classifiers operate on sequences of frames, which enables them to model the patterns that exist in each class. This adds some complexity to the classification process, generally making it slightly slower, but allows for accurate detection of more complex patterns.

Direct classifiers discriminate using the feature space. Examples include nearest neighbour classification, which attempts to find the template that matches the given features the closest, and support vector machine classification. SVMs partition the feature space, using a hyperplane that divides the space in a binary way. Values on one side of the plane are considered matches and values on the other side are considered mismatches.

The other category holds classification methods which use or generate models. For example: with hidden Markov models a model is generated from the training data. This model can then be used to either generate feature data or to calculate the likelihood that this model generated a given sequence of features. The latter is used in classification as the likelihood of observed features is calculated for several different models, allowing the classifier to compare the probabilities and make an educated guess[10]. Other classification algorithms that fall in this category include maximum entropy Markov models and other variations on the Markov model.

Chapter 3

Approach

To find out how to best classify gesture motion we will need to create a sufficiently large data set to allow us to make assumptions based on the results. Also, we need to structure the process, the chain if you will, of operations that we perform on the data set before we even attempt classification. This chapter details this process of gathering and processing data and should give the reader enough insight to attempt a reproduction of this research.

The first choice that we need to make – how will we capture human motion for our experiment – is a very easy one. re-lion owns two Xsens inertial motion capture suits that are part of the SUIT infantry training system. The quality and flexibility of this system is very high and it is readily available in the context of this project.

3.1 Experiment

As explained in chapter 1 we have started with a small exploratory experiment and came to the conclusion that it is very hard to get a good dataset from a highly ecologically valid experiment. Because of this, for our real experiment we tried to get more gestures per recording and also eliminate annotation issues at the cost of some ecological validity. Eight participants (5m, 3f, ages 17-30) were asked to, individually, perform all nine gestures for one minute each, without running the training simulation. The gestures were all performed using the left hand because in the training scenarios students carry a replica of a weapon in the right hand. The motion data was recorded with an Xsens inertial motion capture suit and using a compact video camera. The rest of the SUIT system was not used. The recordings took place in a quiet part of the re-lion office building, either the hangar or the canteen, one participant at a time. The participants did not discuss the experiment with one another previous to the experiment.

First, it was shown to the participants how the gestures should be executed. Participants were then asked to make the gestures continuously for one minute per gesture, one after another, in the order given. Breaks were allowed during the experiment. In total this experiment generated about one and a half hour of motion capture data.These experiments resulted in a large dataset of people making continuous gestures, recorded in spatial coordinates, rotations around axis (rotation quaternions) and acceleration data, all of which the Xsens motion capture suit generates.

Annotating this data was a simple matter of finding the beginning and the end of each sequence of identical gestures and removing the breaks. In case of doubt the video recording could be used to determine if a motion was or was not intended as gesture.

3.2 Features

As Campbell et al. argue[1]: the design of the features that the actual classification is going to take place on is of the greatest importance for the applicability of the classification. In this section we outline the design that we have chosen and how to reproduce it.

3.2.1 Design

In chapter 2 we discussed the different kinds of features that we can use for classification. From the NATOPS research we have learned that coordinates work better than angles and that derivatives can increase performance. Because of this we choose the direction of the motion and the spatial position in which this motion is executed. One could also argue that the shape of the motion is important, however this shape is a function of direction over time. So if we model direction and position – or as we will name these in the rest of this thesis: motion and pose – we have covered the discriminating features of our gestures.

You may consider for a moment here that you could probably come up with more specific features to separate the given gestures. For example: we could check if the wrist is below the elbow and if so, we classify that gesture as number *5: Enemy*. Note however that we are not trying to find features that separate these particular gestures. We're trying to find ways to model and classify gestures in general, be it these gestures or others.

We have also discussed different ways to normalize the data. We will not perform semantic reduction on our data as proposed by Jin and Prabhakaran[8] for this research because the dimensionality of our data is not so large that we need it, and it would reduce the detail in the features. Many of our features are quite similar so we need the details. However we will normalize for shift and rotation as will be discussed shortly.

We also do not apply arm length normalisation in this step, even though that may seem very appropriate for this research. The reason for that is that the Xsens motion capture system does this for us. We can set the limb sizes in the capturing software for each participant, but we have kept the same sizes accross participants in our experiments. The rotations and inertial motions are imposed on a model with the given limb sizes by the software, resulting in motion capture data with similar limb sizes.

3.2.2 Implementation

The features that we have chosen to extract from all this data are the spatial coordinates of the elbow and the wrist, relative to the shoulder (that is, Xw,Yw,Zw and Xe,Ye,Ze; two times three coordinates forming feature pose) and the motion data of these joints, being their positions relative to the positions d frames ago (again, two vectors of three values forming feature motion). Figure 3.1 shows this model.

If we were to simply take the (world) coordinates that the Xsens suit gives us and start classifying with those, our accuracy would be terrible because position and rotation would have an influence on it. The classifiers would not be able to

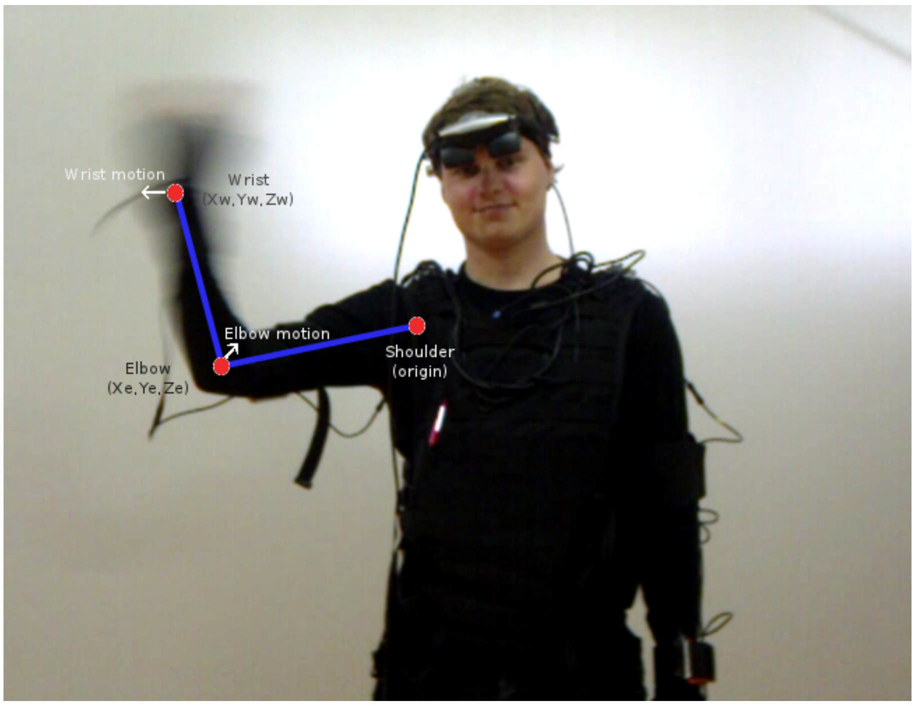


Figure 3.1: *Generating features from motion capture data*

detect the subtle patterns in gestures amid the rough patterns of where someone is standing and in what direction he or she is facing. Even standing up versus sitting down would be of great consequence to the quality of the system.

Because of this we wish to normalize these vectors. If we normalize vectors we convert them from world coordinates to local coordinates. We then get vectors that are relative to some predefined point on the user instead of relative to the world origin. This way we get comparable pose and motion data for the same gesture, that we can train our classifiers on.

Pose

So we have to calculate our local model from the raw world coordinates that the Xsens motion capture data has given us. To get to this model one has to go through five steps:

1. Get the raw motion capture pose data in a readable format
2. Select the right joints
3. Make the pose data shift invariant
4. Make the pose data rotation invariant
5. Calculate the motion vectors

To convert the motion capture data we have written a simple tool that uses the Xsens Moven DLL to extract both spatial coordinates and rotation quaternions from recorded MVN files and outputs them to a file in plain text format. We use Matlab to import this data.

The second step is selecting the right joints. As previously mentioned we have chosen to select only the joints of the left arm because the users of the SUIT system are carrying a weapon in the right hand. In the order that the Xsens DLL works with this data these are the 13th, 14th and 15th triplets of coordinates or quartets of quaternion data.

To make this data shift invariant, and thus relative to the new shoulder origin, we subtract the vector (coordinates) of the shoulder ($V_{shoulder}$) from the vectors of the elbow (V_{elbow}) and wrist (V_{wrist}).

$$\begin{aligned} V_{wrist-shift-invariant} &= V_{wrist} - V_{shoulder} \\ V_{elbow-shift-invariant} &= V_{elbow} - V_{shoulder} \end{aligned}$$

We also need to make the data rotation invariant by interpreting the vectors (V) as the vector part of quaternions (P) and rotating these four dimensional vectors using the original quaternion of the shoulder ($Q_{shoulder}$) so that the rotation of the shoulder (and thus the body) gets subtracted from the pose.

$$\begin{aligned} P_{wrist} &= (0, V_{wrist-shift-invariant}) \\ P_{elbow} &= (0, V_{elbow-shift-invariant}) \\ Q_{shoulder} &= (0, V_{shoulder}) \\ P'_{wrist} &= Q^*_{shoulder} P_{wrist} Q_{shoulder} \\ P'_{elbow} &= Q^*_{shoulder} P_{elbow} Q_{shoulder} \\ (0, V_{wrist-final}) &= P'_{wrist} \\ (0, V_{elbow-final}) &= P'_{elbow} \end{aligned}$$

What results are six shift and rotation invariant values ($V_{wrist-final}$ and $V_{elbow-final}$) for our first feature: arm pose.

Motion

Next and finally, we need to calculate our second feature: the motion vectors. These are vectors indicating the distance and the direction that the joint has moved since d frames ago. We take the elbow and wrist vectors that we have calculated previously ($V_{wrist-final}$, $V_{elbow-final}$) and subtract from these the elbow and wrist vectors that we calculated d frames ago ($V_{wrist-previous}$, $V_{elbow-previous}$).

$$\begin{aligned} V_{wrist-motion} &= V_{wrist-final} - V_{wrist-previous} \\ V_{elbow-motion} &= V_{elbow-final} - V_{elbow-previous} \end{aligned}$$

This dependence on a previous measurement means that our classification will always have a start up time of d frames. Also note that we don't necessarily need to make the data shift invariant before we can calculate the motion vectors as the subtraction does this for us automatically. We do however need to make the data rotation invariant. The easiest and least computationally intensive way to accomplish both, however, is to simply subtract the previously calculated vectors from the current vectors. In chapter 4.1 we will try to determine a good value for the motion offset d.

Finally both features (pose and motion) are scaled independently to the same scale. This makes them weigh equally heavy for the classification. Finding a good scaling factor b for these features will also be discussed in chapter 4.1.

3.3 Classification

We have chosen to compare one direct discriminative classification algorithm and one model based generative classifier for this research. In the former category we use support vector machines and in the latter hidden Markov models. Both have reports of good classification results from various authors and both have comparable implementations in both Matlab and C++[3][13].

In this section we will go a bit deeper into the workings of these classification algorithms.

3.3.1 Support Vector Machine

The first method of classifying the motion capture data is through the use of a support vector machine (SVM). An SVM calculates a plane that divides the "space" described by the features of some training dataset in such a way that all or most instances of one class fall in the same partition of that space. Or in simpler language: you show it which classes exist and give it examples of each class and it tries to find some common pattern in the data that it can use to classify future unlabelled data.

For the SVM classifying software we used LIBSVM[3][4] because it is available in both C and Matlab code and because it is a much used implementation. The features as discussed in this chapter so far were converted into a format that libsvm can use as input. There are roughly four types of SVM kernels:

- Linear
- Polynomial
- Radial Basis Function (RBF)
- Sigmoid

All these kernel types have different characteristics and parameters. However their purpose remains the same; they all partition the feature space to fit the data. The linear function tries to do this with linear planes, the polynomial with polynomial functions, et cetera. According to Hsu, Chang and Lin[6] the RBF kernel is a good choice to start with because it is reliable, has a reasonable amount of parameters and the linear and polynomial kernels are special cases of the radial basis function.

In figure 3.2 you can see the difference in behaviour between the four kernels when applied to a simple two dimensional classification problem. The dots are (unchanging) samples, the background colours show the partitioning.

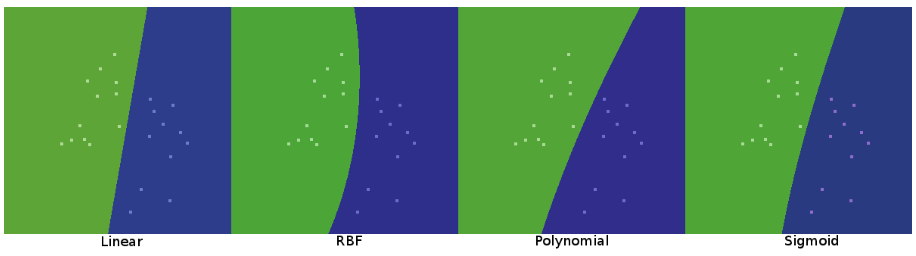


Figure 3.2: *Partitioning a space using different SVM kernels*

3.3.2 Hidden Markov Model

Hidden Markov Models are slightly more complex. They assume that the observations are not telling the full story. They assume that there is a model that cannot be observed that governs the observed behaviour. So instead of trying to partition the space in absolutes, as an SVM tries to do, it tries to define the result in terms of probabilities.

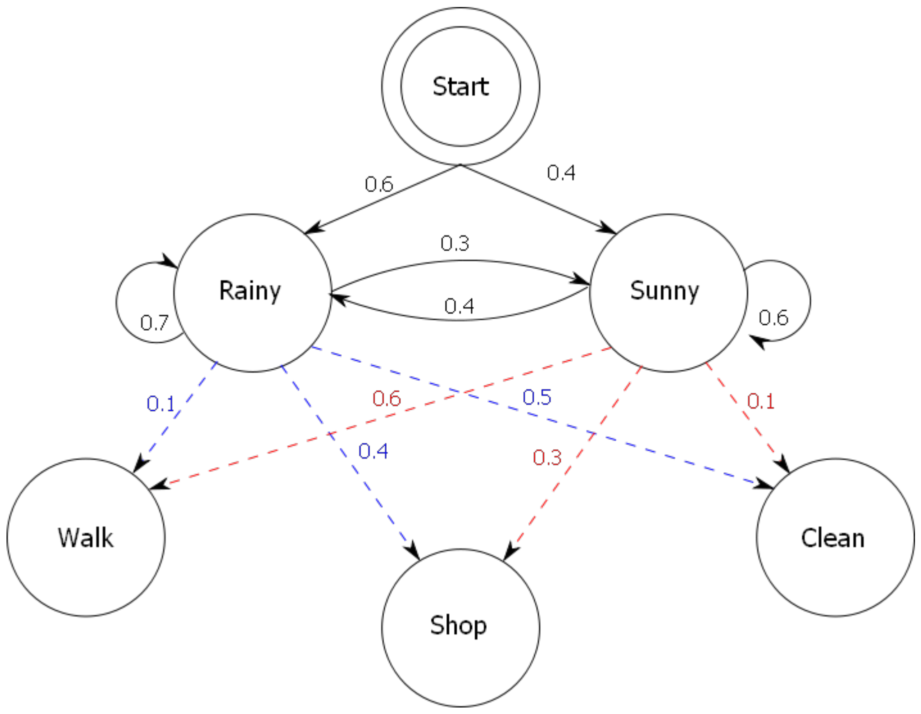


Figure 3.3: *An example of a hidden Markov model*

For example, let us assume that you have a neighbour who only does one of three things on a given day: he goes for a walk, he goes shopping or he cleans the house. And after observing him for a while, you discover a pattern in his actions. When it's sunny he goes for a walk 60% of the time, shops 30% of the time and cleans the remaining 10% of the time. However, when it's raining he cleans the house 50% of the time, goes shopping 40% of the time and walks the remaining 10% of the time.

You also have a general idea of the behaviour of the weather where you live. If it's a sunny day today, chances are that it will be again tomorrow and rainy days are a bit more likely than sunny days. All these parameters can be modelled as shown in figure 3.3.

Having observed this behaviour, you go on a holiday to Japan and call your neighbour every day. He then tells you what he did that day, let's say he went for a walk. This will allow you to estimate that the weather is probably sunny at home. The algorithm you need to make this estimation properly is called the Viterbi algorithm. With it you can find the most likely sequence of weather events that took place, or the most likely path through the hidden Markov model, which has lead to the observation of your neighbour going for a walk that day.

The weather model described above satisfies what is called the Markov property. This property requires that the chances of going to one state or another in the model only depend on the current state, and not the entire history of states. In our simplistic model the chance that tomorrow will be rainy or sunny only depends on today's weather, so the Markov assumption holds and this is a hidden Markov model instead of just a hidden model.

In classification you take this concept one step further and build hidden Markov models for each class. Then you try to match the observed output with the hidden Markov model of each class and see which one gives the highest probability and thus fits best, like you're searching for Cinderella using only the slipper that she left behind.

To train the models and find the most likely path we have used the hidden Markov model Toolbox for Matlab by Kevin Murphy[13].

Chapter 4

Parameter estimation

As explained we will be comparing support vector machines with hidden Markov models. Both SVMs and HMMs require some tuning of parameters for this specific application. The feature extraction also needs some calibration.

4.1 Parameters

We need to make a choice here. Finding good parameters is very computationally intensive and within the context of this Master's thesis we cannot simply calculate every possible combination of parameters, features and classifiers. We have four parameters, three features and two classifiers. That means that we would have to do each experiment 24 times. One experiment deals with the data from eight participants, who have recorded ten minutes of motion capture data each. All participants are classified on the learning data of all others. One second of motion capture data equals to 60 frames of poses and motions, after feature extraction.

We can save on time by making some assumptions. For example, we assume that optimal values for d and b have little dependency on each other and that the curve for b will remain the same for any value of d. Also, the classifier-specific parameters are assumed to be independent from the feature parameters b and d. Another important assumption made is that the results can be generalized over the different participants. We want to build a system that we can apply to any user without user-specific training.

We don't calculate the parameters per gesture (class) to save time, both for this project and because we want to keep the classification generic. Calculating the parameters per feature would also require the resulting system to calculate its parameters per feature after training and before operation, because we do not want to restrict an instructor to our choice of gestures.

4.1.1 Feature calibration

We have to decide on an offset d for the motion vectors that we discussed in chapter 3.2. This variable determines how much motion we consider in our system. Also we need to set an upper boundary b for the scaling that was also discussed there.

To calculate b and d we take the most basic case for all other properties of the system. We train the classifier on one participant using all the default parameters for that classifier and train on all others. We do this for all participants and for both classifiers. The only variable we change is respectively the offset d or the scaling boundary b. We then average the data over all the participants. This gives us two graphs for both classifiers that we can analyze to determine the optimal values.

4.1.2 SVM parameters

Next we have to do the same for the classifier-specific parameters. We start with the SVM classifier, which has two parameters for the RBF kernel: cost and gamma. These values are interdependent so we need to determine them together. Again we train using one participant and classify all others, for each participant. We use the parameters d and b as determined in the last paragraph. We then average the data over the participants and we get one graph in the form of a height map in which we can find the optimal values.

4.1.3 HMM parameters

Finally, the HMM classifier. This classifier has to be applied on sequences of frames or otherwise it will lose its value, so instead of feeding it each frame individually we feed it windows of frames. The window size thus becomes an important factor. If we take a window size that is too large it may encompass multiple gestures and thus lose its value. If we take a window size that is too small it may not contain enough frames in the sequence for the hidden model to apply. So again we have to find this optimal value. For each window size we train the HMM classifier on one participant and test it on all others. We do this for each participant. We use the values for b and d that we found before. We average the results over the participants so we can plot the f-measure as a function of window size and determine a good value.

Two other properties of HMMs, that are interdependent, are the number of states in the model and the number of Gaussians that make up a state. To find optimal values for these variables we have to train the classifier for each combination of values once more. We train on one participant and test on all others, for each participant. We use the parameters d and b as determined before. We then average the data over the participants and analyze the resulting two dimensional height map for the highest f-measure scores.

4.2 Results

The previous paragraph describes how we have found optimal values for the motion vector offset d, the scaling maximum of the features b, the SVM parameters cost and gamma and the HMM parameters window size, number of Gaussians and number of states. This paragraph will document the results.

4.2.1 Feature calibration

As described, we have plotted the f-measure of the classification against the motion offset variable d for both types of classifiers. The result, as shown in figure 4.1a, is surprising. The influence of the motion offset appears to be smaller than expected on average for either classifier. We also expected to see much worse results for a small offset. At more than around 36 frames both classifiers show a decline in f-measure. We see a maximum at around 30 frames for both classifiers, which seems a good value for our d.

We can explain the good results for small offsets from the fact that the raw data we receive from the Xsens motion capture suit is not entirely “raw”. We expected to see jitter in two consecutive frames from inaccuracies in the measuring system. However, the Xsens suit and software do some pre-processing on the data, smoothing out such jitter. This means that two consecutive frames can be used to determine the motion vector with much more accuracy than anticipated.

We move on to parameter b, the scaling maximum. Plotting the f-measure against the upper scaling boundary as in figure 4.1b shows much more effect on the classification than you might expect. Scaling the data on a scale from zero to two gives a much better result for HMMs than it does for the support vector machine. Scaling the data from zero to twelve shows the opposite effect. In general we see that both classifiers get worse for scaling maxima over twelve. We would have expected that the influence of this parameter would be much smaller.

From these values we have chosen a value of two for the HMM classifier and a value of eleven for the SVM classifier. However after some experiments these values turned out to vary greatly with data sets. After several attempts to find optimal values we resorted to sticking with scaling all data to a scale of zero to one. This may not give optimal results but it does give reliable results.

4.2.2 SVM parameters

For the classifier using support vector machines we find that the choice of the parameters cost and gamma does indeed have a significant influence on the f-measure. This is according to expectations. What is not according to expectations is the large difference between the graphs for pose (figure 4.2a), motion (figure 4.2b) and pose plus motion (figure 4.2c).

Note that the scale in the three graphs is not the same and that the graphs have been “patched” a bit. Especially the motion only graph in figure 4.2b, which has been put together from several files, hence the weird bump in the top left (-15,15 to -3,19).

The pose graph shows a steep decline to the right and a slight decline to the bottom left. The center, fanning out to the top left, is a plateau with good results. The best result lies around cost -2 and gamma 2. This highest value lies very close to both declines, so this introduces a risk for future, unknown data.

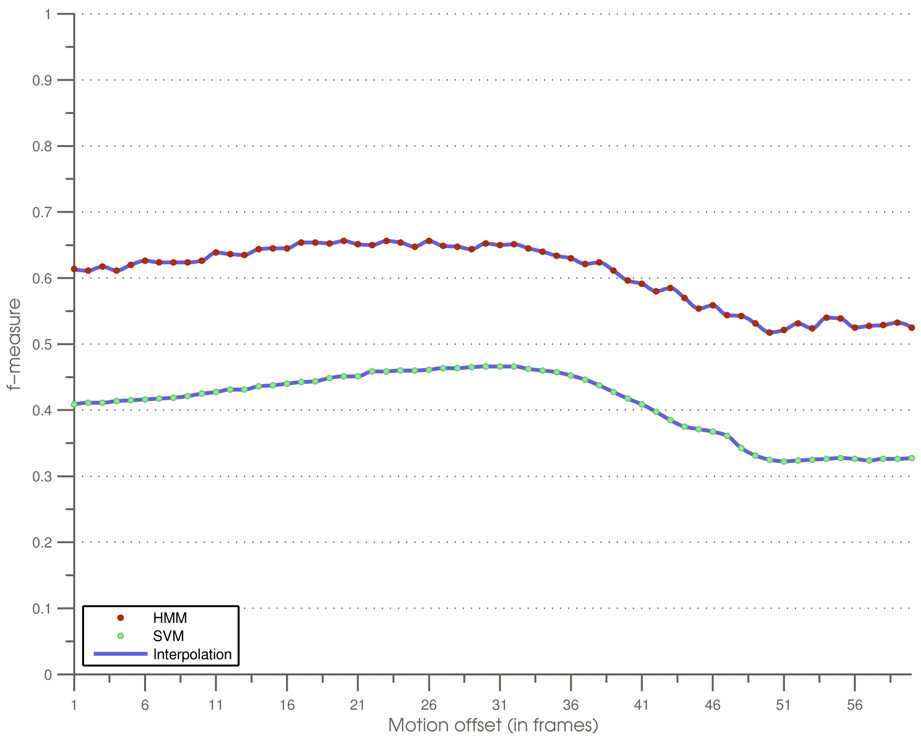
In the motion only graph we see a steep decline in f-measure to bottom left, to the top right appears to be a plateau where the classification results are quite good. The maximum is in the top center region around cost 21 and gamma -2. Again, we have to make sure with these kinds of graphs that we do not pick values too close to the precipice. In this case, because of the unknown “patch”, we pick the slightly safer values of cost 18 and gamma 1.

Finally, the graph on pose and motion. This graph looks very similar to the pose graph, only with greater differences. Also, the highest value (at cost -7 and gamma 3) is even more in the “dangerous” corner. Again, we pick the safer values of cost -3 and gamma 2.

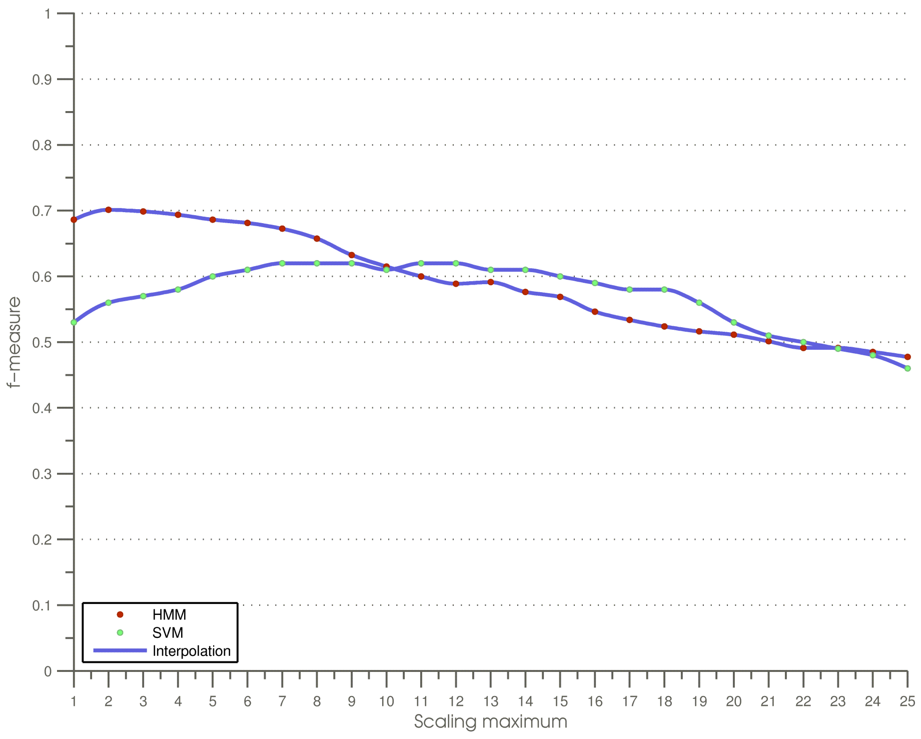
4.2.3 HMM parameters

Increasing the window size for the classifier results in an increasingly better f-measure score, as can be seen in figure 4.3. This is surprising because one would expect to see a maximum after which the result gets worse. Using larger windows may wrongly

Figure 4.1: *Calibrating the feature extraction procedure*

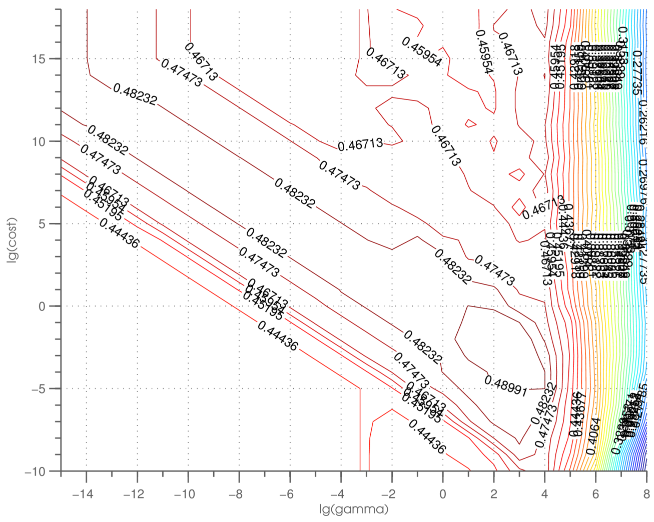


(a) *Finding the right motion offset*

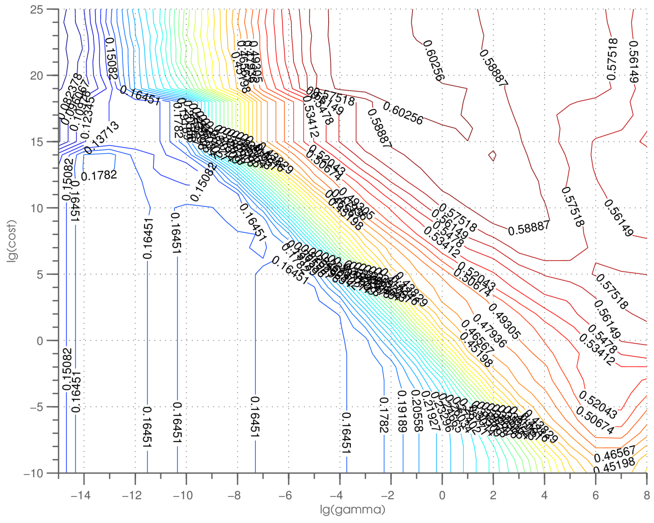


(b) *Finding the right scaling maximum*

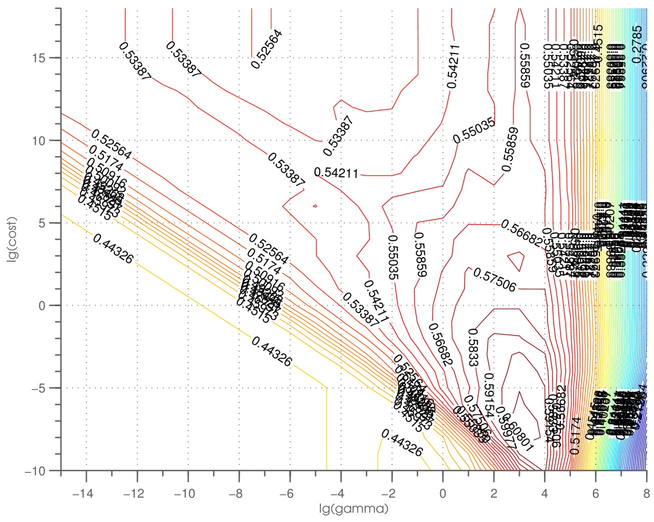
Figure 4.2: *Finding the cost and gamma optima for SVM*



(a) *Pose*



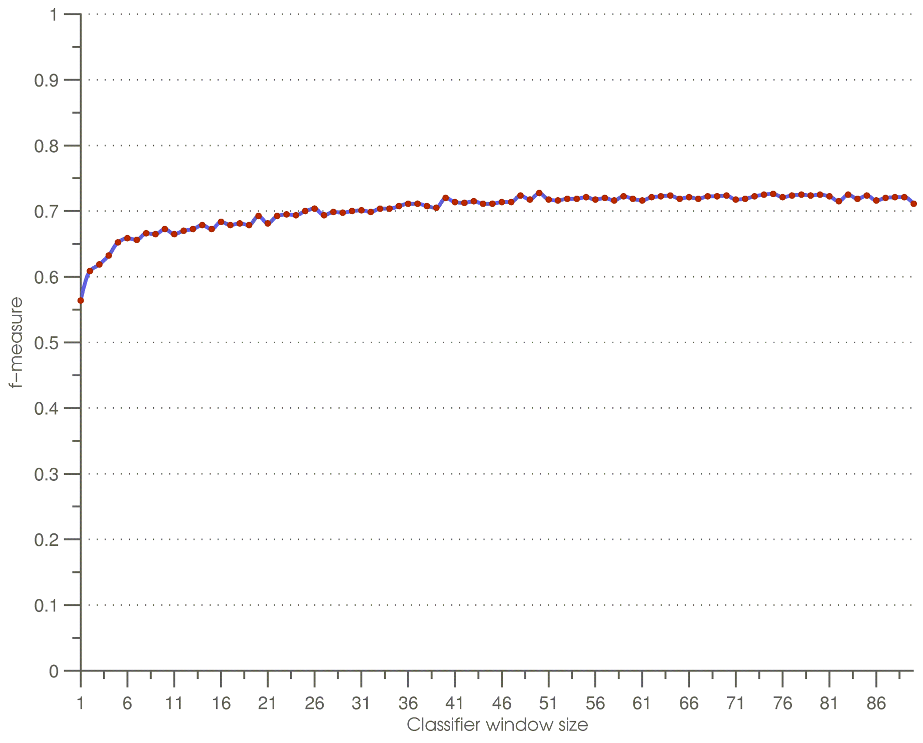
(b) *Motion*



(c) *Pose & Motion*

generalize data as the window starts to overlap several gestures, wrongly classifying some of them or most of them. However we can explain this from the fact that in our data set participants make the same gesture for a minute each. One gesture takes, on average, a little under a second to perform. This is probably why you see the curve levelling off at around 50 frames. However, to keep the gesture classification snappy for our purposes, it would probably suffice to select a window size of 30 frames (half a second).

Figure 4.3: *Finding the right classifier window (HMM)*



Finally, the number of states and Gaussians in the hidden Markov model. Again, we have had to determine these per feature, so once for pose only, once for motion only and once for both pose and motion. The differences are not quite as large as for the support vector machine, but they are relevant nonetheless. The values have been plotted as height maps in figure 4.4.

For all graphs goes that the classifier performs less at very low numbers of states and Gaussians. Less than three states or less than two gaussians is not to be recommended. The motion graph in figure 4.4b shows that, other than that, it does not matter much which number we pick. It shows a slight performance increase for higher numbers of states. We pick the highest value at nine states and four Gaussians.

The pose graph in figure 4.4a shows several “holes” in the height map. There appears to be a safe triangle in the lower left corner with a maximum at eight states and three Gaussians. The differences are small though so we will choose a value in the center of the triangle at five states and four Gaussians.

Finally the pose and motion graph in figure 4.4c shows the same safe triangle, but again, as with the SVM graphs, the differences are bigger. Within this triangle we find a high plateau around six states and three Gaussians and we pick this as our values.

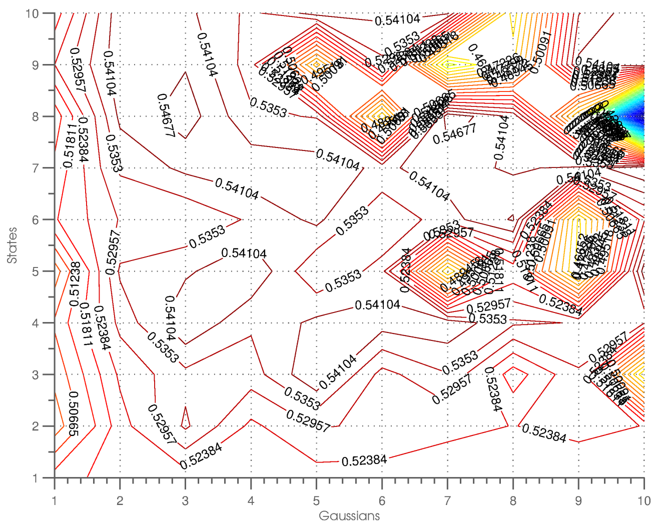
4.2.4 Final parameters

This search has resulted in the optimized parameters shown in table 4.1. With these parameters we can start the experiments that can answer our research questions without bias.

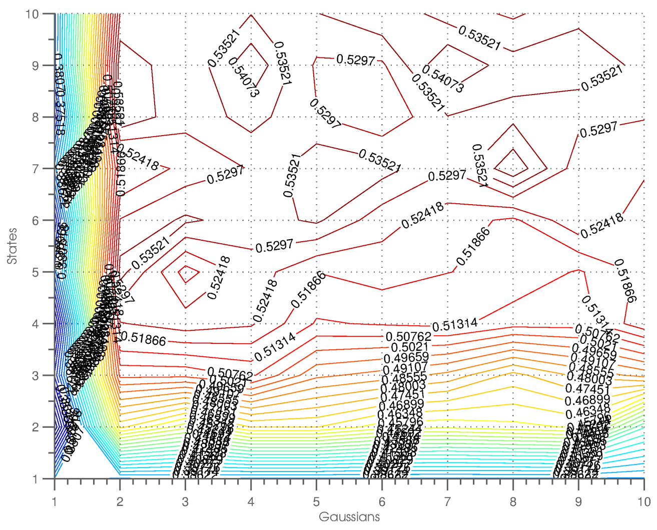
Parameter	**Value**		
	Pose	Motion	Pose&Motion
Motion vector offset d	30 frames		
Scaling maximum b	1		
SVM: Cost	2^{-2}	2^{18}	2^{-3}
SVM: Gamma	2^{2}	2^{1}	2^{2}
HMM: Window size	30 frames		
HMM: Number of Gaussians	4	4	3
HMM: Number of states	5	9	6

Table 4.1: *Gesture recognition parameters*

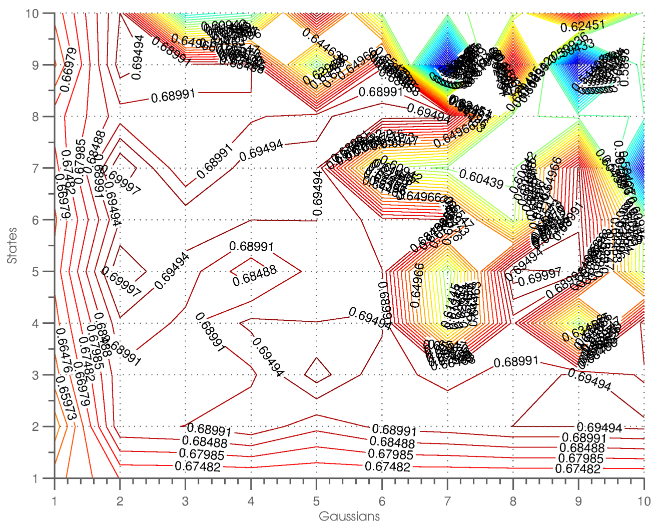
Figure 4.4: *Finding the states and Gaussians optima for HMM*



(a) *Pose*



(b) *Motion*



(c) *Pose & Motion*

Chapter 5

Results

The last step of the process, having constructed usable features and optimized classifiers, is to use the classification to answer the research questions that we began with. This chapter will explain in more detail how we have tried to answer these questions, and then describe the answers that we found and how we came to those answers.

5.1 Research questions

We have asked four research questions in the introduction to this thesis. In this section we will briefly cover these four questions and describe the steps we have taken to answer them.

5.1.1 Pose versus motion

Which feature is more information-rich in this application: pose or motion?

To answer this question we have used the straight-forward approach of training and classifying all data three times: once with only pose data, once with only motion data and one final time with both features. We do this twice, once with the SVM classifier and once with the HMM classifier. We train the classifiers on all the data of one person, and test it on all other data. We do this for each person. This results in averaged f-measures and confusion matrices for all three options, for both classifiers and for each person. Averaging this data over the participants gives us six f-measures and six confusion matrices for comparison.

For this experiment we expected that motion would be best for some gestures and pose for others. We expected the combination to work best.

5.1.2 Intra- versus inter-participant

What is the loss of accuracy when using only the mocap data from one person as training data versus training on multiple participants?

To be able to answer this question we first need to know what the classification accuracy would be if we split the data up in the traditional way for an intra-participant test: from the data of each participant we use a part for training and a part for testing. If we create a train set from various participants in this way that is the size of one entire set for one participant and use the rest for testing, we can compare the two approaches and see what the difference is. Of course we run this test twice again, once with SVM and once with HMM, for each participant. We can compare this result to the results of the previous test to see what the difference in accuracy is.

We expected the intra-participant training set to give better results because it makes the classifiers less specific to a single participant and better able to ignore the differences between participants.

5.1.3 Train size influence

What is the influence of the size of the training data on the accuracy?

Because we need to be able to answer this question independently of the last we have kept the training set size equal across the different tests. In this test we're going to change that variable. We stick to training with one person again, the variable being how many frames we use to train each class. We again do this once for both classifiers. We can also do this with each participant as training set. This generates many f-measures and confusion matrices, which we can average over the participants. We can then plot the f-measure of the classifications against the number of frames used for training.

For this experiment we expected to see an increasing accuracy for both classifiers with an increasing training set, to a certain maximum.

5.1.4 SVM versus HMM

Which classification approach is best suited for this application: SVM or HMM?

Of the six f-measures and confusion matrices from the pose versus motion experiment, three are generated using SVMs and three using HMMs. This should give us all the data we need to be able to draw conclusions about which approach is more suitable in the specific case where we use one person's data as training data and test on the others. The second test should give us an idea which classifier is better when testing within subject. The third test should give us an idea of which classifier is better at handling small training sets.

We expect to see HMMs perform better on classes dealing with motion and on the inter-participant test. SVM could be better in the intra-participant test and perhaps with static poses. Overall we expect to see HMMs outperform SVMs because of the nature of the data. Below the motion capture data lie patterns of gestures, that we expect HMMs to be able to detect. Although we will not be measuring it, it might be good to mention that SVMs are faster and less CPU-intensive than HMMs, so there are certainly good reasons to go with SVMs.

5.2 Results per research question

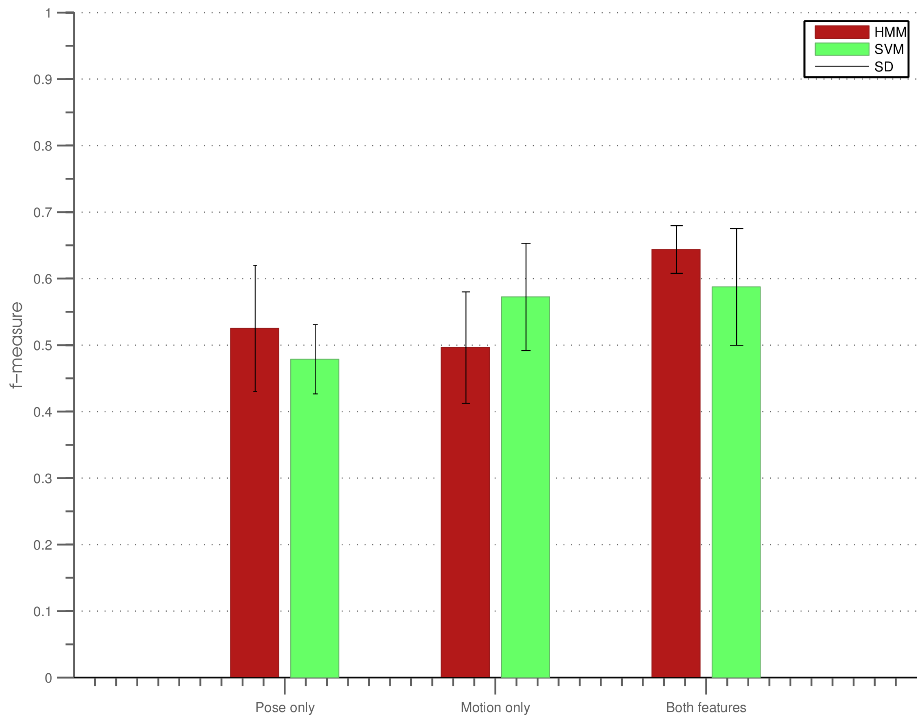
This section will present the results of the four central questions, using the experiments that we just described. Each answer will be illustrated with graphs and confusion matrices.

We have used a validation set for the experiments in the chapter on set-up and parameter estimation. We have conducted these experiments on a (different) test set. Because of this you may see some apparent inconsistencies in the specific f-measures between the chapters.

5.2.1 Pose versus Motion

First we will compare the different features that we have used: pose, motion and a combination of the two. Which gives us the best classification results? As shown in figure 5.1 our expectation was correct: the combination of pose and motion gives the best results for both the SVM and HMM classifiers. Motion scores higher than pose for the support vector machine and pose wins over motion for the hidden Markov model.

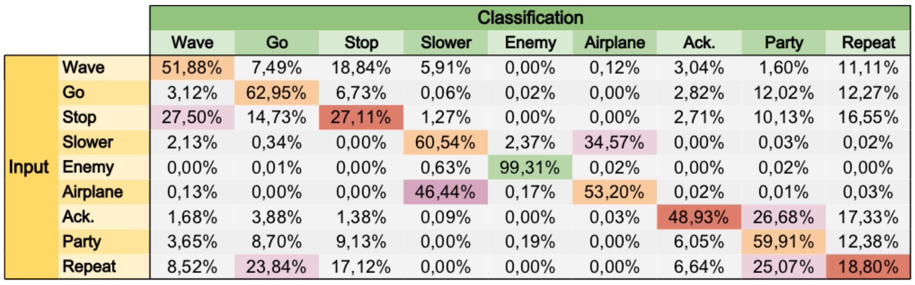
Figure 5.1: *Pose versus motion*



We had expected HMMs to outperform SVMs for the motion feature, because HMMs are better able to model dynamic processes. However, it seems that the motion feature does a really good job at making our dynamic process easy to model in a static way. Such a good job in fact that the motion feature alone classifies our gestures almost as good as the combined features for SVMs. With HMMs you can clearly see that the static poses and the dynamic motions complement each other and drive the accuracy up when used in combination.

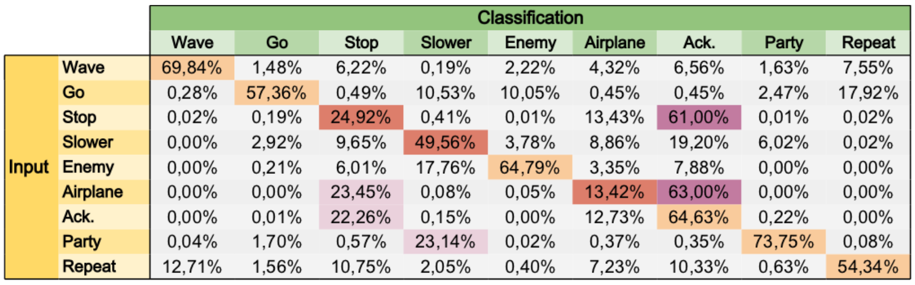
We would expect that some gestures are recognized better by motion features and others by pose features. To investigate this we have to look at the six bars in the graph with more detail. In figure 5.2 you can see the confusion matrices of some of the bars.

Figure 5.2: *Confusion matrices for HMM*



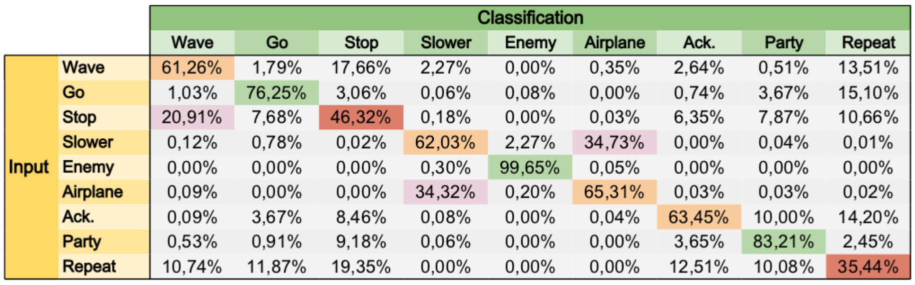
		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
Input	Wave	51,88%	7,49%	18,84%	5,91%	0,00%	0,12%	3,04%	1,60%	11,11%
	Go	3,12%	62,95%	6,73%	0,06%	0,02%	0,00%	2,82%	12,02%	12,27%
	Stop	27,50%	14,73%	27,11%	1,27%	0,00%	0,00%	2,71%	10,13%	16,55%
	Slower	2,13%	0,34%	0,00%	60,54%	2,37%	34,57%	0,00%	0,03%	0,02%
	Enemy	0,00%	0,01%	0,00%	0,63%	99,31%	0,02%	0,00%	0,02%	0,00%
	Airplane	0,13%	0,00%	0,00%	46,44%	0,17%	53,20%	0,02%	0,01%	0,03%
	Ack.	1,68%	3,88%	1,38%	0,09%	0,00%	0,03%	48,93%	26,68%	17,33%
	Party	3,65%	8,70%	9,13%	0,00%	0,19%	0,00%	6,05%	59,91%	12,38%
	Repeat	8,52%	23,84%	17,12%	0,00%	0,00%	0,00%	6,64%	25,07%	18,80%

(a) *Pose feature*



		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
Input	Wave	69,84%	1,48%	6,22%	0,19%	2,22%	4,32%	6,56%	1,63%	7,55%
	Go	0,28%	57,36%	0,49%	10,53%	10,05%	0,45%	0,45%	2,47%	17,92%
	Stop	0,02%	0,19%	24,92%	0,41%	0,01%	13,43%	61,00%	0,01%	0,02%
	Slower	0,00%	2,92%	9,65%	49,56%	3,78%	8,86%	19,20%	6,02%	0,02%
	Enemy	0,00%	0,21%	6,01%	17,76%	64,79%	3,35%	7,88%	0,00%	0,00%
	Airplane	0,00%	0,00%	23,45%	0,08%	0,05%	13,42%	63,00%	0,00%	0,00%
	Ack.	0,00%	0,01%	22,26%	0,15%	0,00%	12,73%	64,63%	0,22%	0,00%
	Party	0,04%	1,70%	0,57%	23,14%	0,02%	0,37%	0,35%	73,75%	0,08%
	Repeat	12,71%	1,56%	10,75%	2,05%	0,40%	7,23%	10,33%	0,63%	54,34%

(b) *Motion feature*



		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
Input	Wave	61,26%	1,79%	17,66%	2,27%	0,00%	0,35%	2,64%	0,51%	13,51%
	Go	1,03%	76,25%	3,06%	0,06%	0,08%	0,00%	0,74%	3,67%	15,10%
	Stop	20,91%	7,68%	46,32%	0,18%	0,00%	0,03%	6,35%	7,87%	10,66%
	Slower	0,12%	0,78%	0,02%	62,03%	2,27%	34,73%	0,00%	0,04%	0,01%
	Enemy	0,00%	0,00%	0,00%	0,30%	99,65%	0,05%	0,00%	0,00%	0,00%
	Airplane	0,09%	0,00%	0,00%	34,32%	0,20%	65,31%	0,03%	0,03%	0,02%
	Ack.	0,09%	3,67%	8,46%	0,08%	0,00%	0,04%	63,45%	10,00%	14,20%
	Party	0,53%	0,91%	9,18%	0,06%	0,00%	0,00%	3,65%	83,21%	2,45%
	Repeat	10,74%	11,87%	19,35%	0,00%	0,00%	0,00%	12,51%	10,08%	35,44%

(c) *Both features*

From these confusion matrices we can see that some gestures get mixed up when using only the pose feature and others get mixed up when using the motion feature. The results for the SVM classification are comparable, see figure 5.3.

For pose we clearly see that the gestures *Slower* and *Airplane* get mixed up. This makes sense because these gestures are made in the same general area: with a stretched arm away to the side. Also, the classifier has great trouble discerning between the gestures that are made next to the head, and thus have overlapping features: *Wave*, *Go*, *Stop*, *Acknowledge*, *Party* and *Repeat*. Some of these gestures perform better than others, like *Wave*, *Go* and *Party*. This is because these gestures overlap only in a part of their trajectory. For most of their trajectory they are respectively further away from the head, in front of the body and below the shoulder, which makes them easy to classify from pose information. The class *Enemy*, finally, classifies very well because it is the only gesture executed downwards.

When we look at the confusion matrix for the motion feature we see a very different pattern. The classification is not able to keep the three static gestures apart: *Stop*, *Airplane* and *Acknowledge*. This makes sense because the motion vectors for these classes are all very close to zero. The same goes for the confusion between *Slower*, *Enemy* and *Party*, which all three have an oscillating motion going up and down. However, the HMMs can discern these classes relatively well from the ratio of movement between the elbow and the wrist. Finally, we can see that classes like *Wave* and *Go*, which are executed in the same general area but in opposite directions, are virtually not mixed up at all, but do both get mapped wrongly to *Repeat*, which contains movement in both directions.

The last matrix contains the results of the combined feature. Here we see most classes being classified quite well. The errors of the pose and motion matrices get smoothed out by combining the features, which results in a better accuracy overall. Two classes however consistently give very bad results; *Stop* and *Repeat*. This is probably the case because both their position (left of the participant's head) and their motion (either static or small motions in the X and Z direction) are not very unique. They clearly also reduce the accuracy of the rest of the classifications, as they both get false positives, so leaving these gestures out would increase the quality of the whole system.

Figure 5.3: *Confusion matrices for SVM*

		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
	Wave	55,90%	6,87%	9,91%	9,38%	0,00%	0,23%	2,67%	1,99%	13,07%
	Go	6,72%	52,71%	4,99%	0,24%	0,23%	0,00%	2,05%	18,45%	14,61%
	Stop	31,33%	12,79%	21,00%	1,79%	0,00%	0,00%	2,50%	10,82%	19,77%
	Slower	6,97%	0,21%	0,01%	54,24%	3,03%	35,46%	0,00%	0,00%	0,08%
Input	Enemy	0,00%	0,00%	0,00%	1,48%	98,44%	0,00%	0,00%	0,06%	0,02%
	Airplane	0,47%	0,00%	0,00%	39,10%	0,01%	60,43%	0,00%	0,00%	0,00%
	Ack.	5,54%	3,59%	2,44%	0,00%	0,00%	0,00%	44,18%	21,67%	22,59%
	Party	10,83%	11,56%	5,94%	0,01%	1,38%	0,00%	8,88%	44,06%	17,35%
	Repeat	18,37%	21,67%	9,43%	0,06%	0,14%	0,00%	8,74%	21,02%	20,56%

(a) *Pose feature*

		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
	Wave	60,57%	11,37%	1,88%	0,58%	2,60%	0,37%	0,24%	2,08%	20,31%
	Go	7,17%	55,51%	0,88%	5,89%	2,01%	0,22%	0,10%	6,04%	22,18%
	Stop	3,13%	1,02%	38,06%	2,43%	0,92%	18,61%	33,97%	1,79%	0,07%
	Slower	2,48%	18,77%	1,68%	62,40%	3,20%	3,95%	0,51%	4,83%	2,19%
Input	Enemy	7,53%	8,50%	0,80%	7,45%	70,20%	0,47%	0,20%	2,78%	2,07%
	Airplane	0,20%	0,16%	13,01%	6,42%	0,22%	49,68%	30,26%	0,05%	0,00%
	Ack.	0,53%	0,15%	13,69%	1,96%	0,64%	21,49%	60,29%	1,26%	0,00%
	Party	2,66%	13,38%	2,09%	3,72%	0,90%	0,22%	0,24%	75,09%	1,68%
	Repeat	22,02%	24,14%	0,08%	0,47%	0,98%	0,03%	0,01%	6,85%	45,42%

(b) *Motion feature*

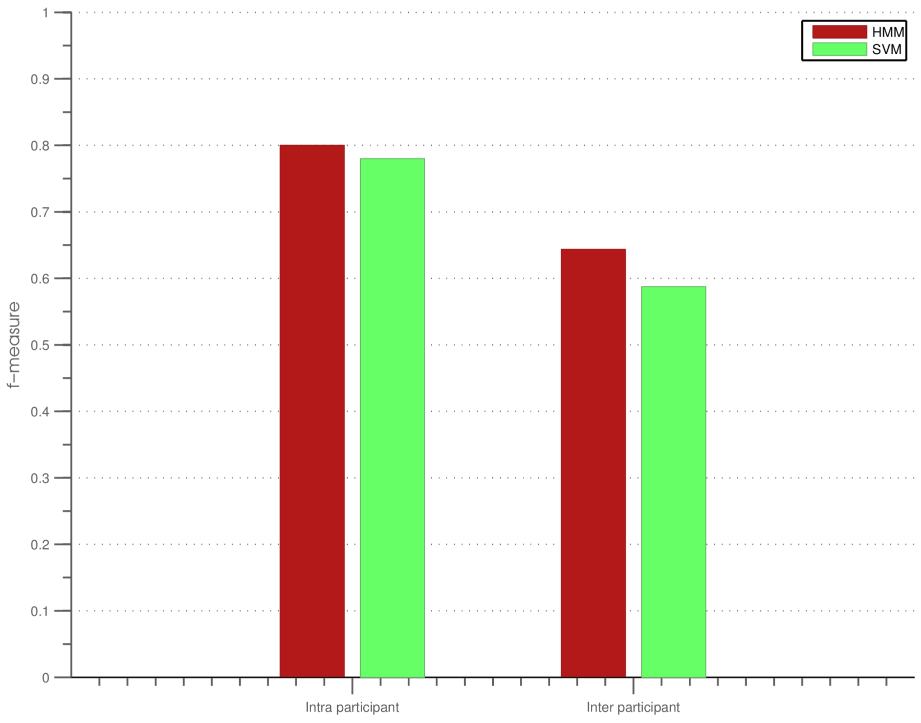
		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
	Wave	68,14%	6,83%	6,83%	1,91%	0,00%	0,13%	0,83%	0,78%	14,54%
	Go	4,66%	65,14%	3,10%	0,04%	0,09%	0,00%	1,27%	7,97%	17,73%
	Stop	34,87%	12,01%	28,93%	0,00%	0,00%	0,00%	2,21%	6,41%	15,56%
	Slower	0,93%	0,98%	0,00%	62,18%	2,56%	33,30%	0,00%	0,05%	0,00%
Input	Enemy	0,00%	0,00%	0,00%	0,54%	99,46%	0,00%	0,00%	0,00%	0,00%
	Airplane	0,00%	0,00%	0,00%	39,68%	0,07%	60,25%	0,00%	0,00%	0,00%
	Ack.	0,06%	13,91%	3,50%	0,00%	0,00%	0,00%	52,84%	18,48%	11,22%
	Party	4,26%	12,10%	3,87%	0,02%	0,03%	0,00%	7,21%	64,68%	7,84%
	Repeat	17,91%	24,00%	7,08%	0,00%	0,02%	0,00%	7,91%	15,58%	27,50%

(c) *Both features*

5.2.2 Intra- versus inter-participant

Next we try to answer the question how much accuracy is lost because we train our classifiers on only one instructor and not on all participants. So as a test we have trained the classifiers on a small part of each participant and then tested on the rest of the data of the participants (intra-participant). The total amount of training data remains the same as in the previous experiment. We compare this to the result from the pose versus motion experiment (inter-participant). This test has only been done for the combined motion and pose features.

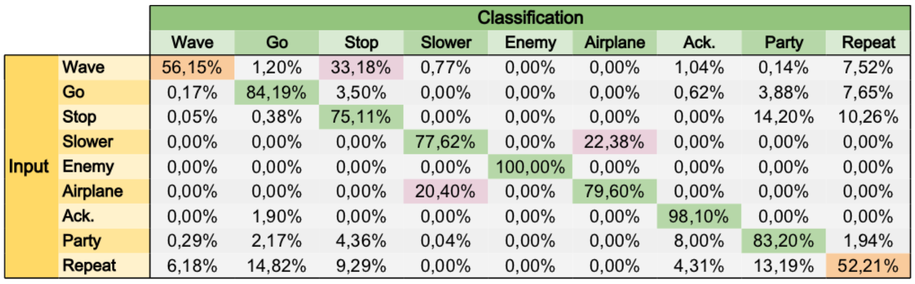
Figure 5.4: *Intra- versus inter-participant*



As you can see in graph 5.4, for HMMs we lose about 16 percentage points in our inter-participant experiment compared to if we would have chosen an intra-participant approach. For SVMs we lose 19 percentage points. This means that we could clearly gain some accuracy from changing our usage scenario. If we let both the instructor and the students record a few minutes of motion capture data, we probably see better results.

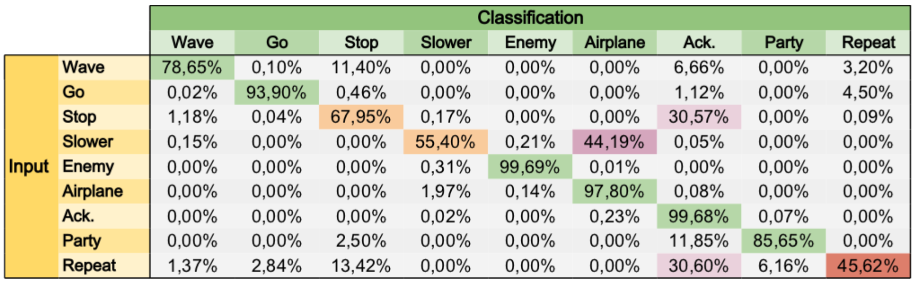
When we look at the confusion matrices for the intra-participant experiment in figure 5.5 we can see that the classification is less "chaotic": the confusion is more concentrated than in our original experiment. Again we see that the *Repeat* gesture scores badly. We can also see more explicitly the confusions from the individual features pose and motion: *Slower* and *Airplane* are mixed up because their motion features are very similar. *Stop* and *Acknowledge* are confused by the HMM because both their motion and position features are very much alike. The SVM seems to have little trouble with this though, but it mixes up *Wave* and *Stop* which are very similar in pose but not in motion.

Figure 5.5: *Confusion matrices for intra-participant experiment (pose + motion)*



		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
Input	Wave	56,15%	1,20%	33,18%	0,77%	0,00%	0,00%	1,04%	0,14%	7,52%
	Go	0,17%	84,19%	3,50%	0,00%	0,00%	0,00%	0,62%	3,88%	7,65%
	Stop	0,05%	0,38%	75,11%	0,00%	0,00%	0,00%	0,00%	14,20%	10,26%
	Slower	0,00%	0,00%	0,00%	77,62%	0,00%	22,38%	0,00%	0,00%	0,00%
	Enemy	0,00%	0,00%	0,00%	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%
	Airplane	0,00%	0,00%	0,00%	20,40%	0,00%	79,60%	0,00%	0,00%	0,00%
	Ack.	0,00%	1,90%	0,00%	0,00%	0,00%	0,00%	98,10%	0,00%	0,00%
	Party	0,29%	2,17%	4,36%	0,04%	0,00%	0,00%	8,00%	83,20%	1,94%
	Repeat	6,18%	14,82%	9,29%	0,00%	0,00%	0,00%	4,31%	13,19%	52,21%

(a) *SVM confusion*



		Classification								
		Wave	Go	Stop	Slower	Enemy	Airplane	Ack.	Party	Repeat
Input	Wave	78,65%	0,10%	11,40%	0,00%	0,00%	0,00%	6,66%	0,00%	3,20%
	Go	0,02%	93,90%	0,46%	0,00%	0,00%	0,00%	1,12%	0,00%	4,50%
	Stop	1,18%	0,04%	67,95%	0,17%	0,00%	0,00%	30,57%	0,00%	0,09%
	Slower	0,15%	0,00%	0,00%	55,40%	0,21%	44,19%	0,05%	0,00%	0,00%
	Enemy	0,00%	0,00%	0,00%	0,31%	99,69%	0,01%	0,00%	0,00%	0,00%
	Airplane	0,00%	0,00%	0,00%	1,97%	0,14%	97,80%	0,08%	0,00%	0,00%
	Ack.	0,00%	0,00%	0,00%	0,02%	0,00%	0,23%	99,68%	0,07%	0,00%
	Party	0,00%	0,00%	2,50%	0,00%	0,00%	0,00%	11,85%	85,65%	0,00%
	Repeat	1,37%	2,84%	13,42%	0,00%	0,00%	0,00%	30,60%	6,16%	45,62%

(b) *HMM confusion*

5.2.3 Train size influence

We are also interested to know if the result that we found is greatly influenced by the limited amount of data of the training set. The data from one participant, which is nine minutes of motion capture data in total, might just be insufficient for optimal results.

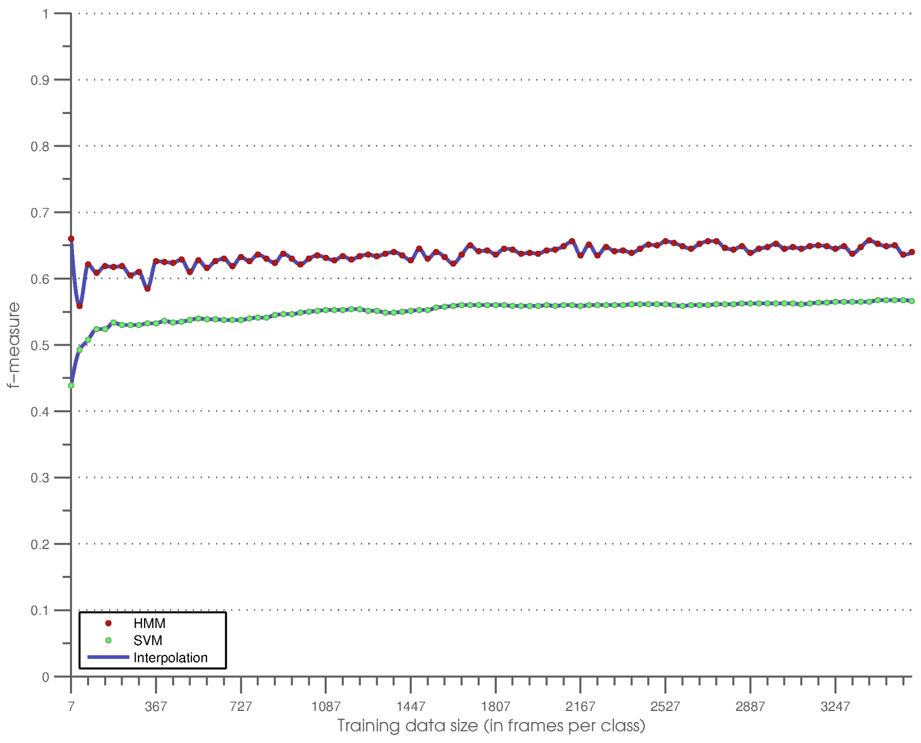
Graph 5.6 answers this question. We can clearly see that we don't need a very large training set. If we have more than 200 frames for each class the f-measures for both classification methods are very close to the f-measures for the full data set. 200 frames equals a little over three seconds. However, we do still see a slight increase in f-measure for increasing numbers of frames. At the far right in the graph is our full training set of 60 seconds per class times 60 frames per second, which gives 3600 frames per class. At that point the graph is still rising, so having more frames per class may still increase the classification results, but the influence will not be quite as large as the influence of testing intra-participant, for example.

5.2.4 SVM versus HMM

If we look at graph 5.1 and graph 5.4, where the results have been split in an f-measure for SVMs and one for HMMs, we can see that, in most cases, the f-measure for HMMs is higher than for support vector machines. As expected the SVM classifier performs better in the intra-participant test than in inter-participant tests, but it still does not outperform the hidden Markov model.

If we compare the classifiers in motion or pose we do not see the expected result that SVMs are better than HMMs at classifying static poses and HMMs better than SVMs for motion, quite the opposite. With our features the SVM classifier beats the HMM classifier for motion and the HMM classifier beats the SVM classifier for pose. The differences are not very large though.

Figure 5.6: *Train size influence*



Looking at the confusion matrices in figures 5.2, 5.3 and 5.5 we can see that the SVMs and the HMMs behave in a very similar way, mostly confusing the same classes. They do however have a different tendency for misclassification, picking different classes to map gestures to about which they are uncertain. The intra-participant experiment is a good example of this: both classifiers had trouble keeping *Wave* and *Stop* apart but the SVM classifier mapped unknowns on *Stop* and the HMM classifier on *Wave* and *Acknowledge*. This makes the difference between *Wave* being classified well and *Stop* only acceptable, or the other way around.

We can only conclude that the difference between SVMs and HMMs is not as large as we had expected, but we do see HMMs outperform SVMs in most applications, as expected.

5.3 Discussion

In this section we will be discussing the results and their relevance to the application of virtual infantry training. We describe several use case scenarios that, given the results above, could be designed around our gesture recognition approach in future infantry training systems.

We have seen an acceptable (but not great) classification result for both HMMs and SVMs, given that we classify inter-participant with little training data from just one instructor. We have also seen that we can improve this result significantly by classifying intra-participant.

5.3.1 Intra-participant

Whether we are able to classify intra-participant, and if we have a need for this, depends highly on the application. Our classification system could be used very well to allow an instructor to control the simulation from within, which makes for an excellent intra-participant scenario. For example: the instructor could pause the simulation to explain something to the students, he could control the recording of the simulation for an after action review when he sees something of interest happening or he could trigger a scenario change (an ambush, civilians passing by, et cetera). Such applications are quite realistic with the classification results as they are.

In fact, the intra-participant experiment is based on a training set which consists of training data from all participants. Then, we classify test data from all participants and we get classification results in the range of 80%. If we were to train and classify on only one person, it is very likely that the results would be even better because the data set would then have less variance and the way in which different users perform different gestures would no longer be of any importance.

Another use we could make of this knowledge is that we allow students to record a few of their own gestures under supervision of the instructor before entering the training simulation. We could then use these gestures (either in combination with gestures from the instructor, or on their own) to train a classifier specific for their system. We should however ask ourselves if this is desirable. Students are likely to make mistakes, which would then influence the classification.

5.3.2 Inter-participant

The classification results for an inter-participant setting may not be as high, but depending on the use case it could still be very useful. Giving direct feedback on gestures with these classification accuracies is probably not a very good idea. However, there are ways of giving indirect feedback that solve this problem.

For example, the system could score students' gestures relative to other students, and produce a ranking of who has made their gestures best. This could be added to the application as a serious mini-game, challenging students to perform best at making understandable gestures.

Another way to give indirect feedback is to use the recognized gestures as input for the internal model of artificial agents in the system. People make mistakes, and so may virtual agents pretending to be people. These agents could be either allies that the students communicate with through gestures, or enemies that "observe" the students from a distance and try to gain a strategic advantage from reading their gestures.

These are all use cases of in-game gesture recognition that do not require a very high accuracy or reliability but are nevertheless quite useful. These applications would require no training data from the participants, only from an instructor or a previously recorded session. The approach described in this thesis could be directly applied to these scenarios.

5.3.3 Gestures

In section 5.2.1 we have seen that some gestures are better classified with motion features and others with pose features. We have also seen that some gestures are not classified very well with either pose, motion or combined features. Part of the reason for this is that the nine gestures that this thesis focusses on lie very close together, which is what they were selected for. If we have control over the gestures that we choose to allow in our system we can use this knowledge to our advantage.

For example, if we wish to classify three gestures very fast and with high accuracy, we need to select gestures that lie far apart in the pose plane. We choose the pose feature because it is slightly faster to calculate. For gestures we could classify *Stop*, *Enemy* and *Airplane*. These gestures are easy to keep apart in the pose plane by either HMMs or SVMs because they are very different in execution physically and they are hardly ever confused.

Note that this way we could classify three gestures with great accuracy and great speed, even though the classification is inter-participant, with limited training data from just one instructor. This could be quite useful for all kinds of applications, one of which could be controlling an in-game menu without a controller. If we use *Stop* and *Airplane* for respectively right and left and *Enemy* for select, this could provide a very natural and easy to learn interface.

5.3.4 Classifiers

From our results we can not distinguish a huge difference between support vector machines and hidden Markov models. The HMMs are slightly better overall but for some applications, like classifying motion only features, the SVMs outperform the HMMs. This small difference in classification accuracy enables us to look at other advantages and disadvantages of both methods and of their implementations.

For example: SVMs are faster than HMMs as shown in [11]. We have not measured this but if speed is an issue for a real-life application then this could be an important factor, either concerning the time it takes to train the system or the speed of classification. Another consideration can be if the classifiers are available in your programming language of choice.

5.3.5 Training size

We have seen that the classification can work quite well with very little training data. Because of this, training the classifier could be a very trivial task that is part of the initialisation of the training simulation, like calibration is now. Users are currently asked to assume a series of poses to calibrate the motion capture suit. In the future, each training scenario could use different gestures and ask the user to make those gestures to initialize the scenario.

To give one example: imagine a scenario where the students have to press a button to disarm an explosive. Before the scenario starts, the students could be asked to hit a virtual token twice, which would give the system enough motion capture data to know what "hitting" looks like. Next, if this hitting motion is detected close enough to the location of the button, this could trigger the scenario to disarm the explosive.

In a broader sense, small requirements on the training size open up possibilities for use case scenarios where we want to recognize repetition. You have made some gesture in situation A, now we can detect if you are making this gesture again in situation B.

Chapter 6

Conclusion

In this thesis we have shown that classification can be used in a virtual infantry training situation. Less than ten minutes of training data from one instructor is sufficient for classifying nine different gestures with an f-measure of 0.65 on average. This classification can then be used for a plethora of applications including scoring students relative to each other, allowing the instructor gesture control over the scenario and as input to artificial intelligent agents.

6.1 Recap

We wanted to know if it is feasible to add gesture recognition capabilities to the SUIT virtual infantry training simulation system. To answer this broad question we have looked at different classification approaches, different features and different training set sizes. Also, we have looked into the consequences of classifying inter-participant (instructor - student) versus an intra-participant approach.

For this research we conducted an experiment to construct a data set of over an hour of motion capture data, expressing nine gestures as explained in section 1.3 and chapter 3. In the same chapter we have constructed several different features for later comparison (section 3.2). In chapter 4 we determined optimal parameters for classification algorithms using support vector machines and hidden Markov models.

Finally we have used these features, algorithms and optimized parameters to conduct several experiments as described in chapter 5 to find answers to the questions that we set out with.

6.2 Findings

Concerning classification performance, hidden Markov models give the best results for this application, although the difference with support vector machines is not very large. For our motion-only feature SVMs even outperform HMMs. The combined motion and pose feature clearly scores best for both classifiers. Hidden Markov models with the combined feature give the best classification result, with an f-measure of 0.65.

If we look at the intra-participant score we see that we can classify the instructor's gestures with an f-measure of 0.80, which should be more than enough to enable the instructor to send gesture based commands to the system. As explained in section 5.3.1, the results will probably be even better than that if we train and classify the system on only one person.

Finally, we have seen that the system is very forgiving for a small training data set. Just a few seconds of motion capture data per gesture class allows us to classify subsequent frames with acceptable accuracy and the quality of the system is only slightly improved with more training data.

6.3 Future work

One very important aspect for the implementation of this approach in the SUIT system that has seen no attention at all in this thesis is gesture detection. Our system expects all input motion capture data to be one of nine gestures, and tries to recognize which gesture it is. In real training situations, much of the motion capture data does not contain any gesture and should thus be filtered out before the classification is applied. Detecting gestures in a stream of motion capture data is a completely different topic altogether and would be a good starting point for future work.

Another aspect that has seen little attention in this thesis is the on-line classification in near-real time. This has been a concious choice because the implementation in Matlab is not comparable to the C++ and Lua implementation that would be implemented in re-lion's SUIT system. However, based on our results in Matlab it should be possible to implement this on-line.

As explained in section 4.2.1, we have seen that scaling the feature data has an unexpectedly large impact on the quality of the classification. It also made the classification unstable because the scaling was very dependent on the data set. We use scaling to make sure that the classifiers weigh the pose component and the motion component equally heavy. It may be possible to achieve better results using optimized scaling bounds for a specific data set. Also, maybe we don't want the motion and pose components to weigh equally heavy, perhaps we can improve the classification by shifting this balance.

Finally, future work into making this gesture recognition system an integrated part of the training simulation would do good to focus more on a specific application. For this thesis we have tried to keep the system as generic as possible, allowing an instructor to record any gestures at the start of a session. This will work, however if we want to improve the accuracy of the system we will need to focus on a specific application, select gestures that fit and optimize the features for these gestures. That way very high accuracies can be achieved.

Bibliography

[1] L.W. Campbell, D.A. Becker, A. Azarbayejani, A.F. Bobick, and A. Pentland. Invariant features for 3-d gesture recognition. In *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 157–162, oct 1996.

[2] L.W. Campbell and A.F. Bobick. Recognition of human body motion using phase space constraints. pages 624–630, 1995.

[3] C.C. Chang and C.J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[4] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[5] M. Hansard, S. Lee, O. Choi, and R. Horaud. Time-of-flight cameras: Principles, methods and applications. 2012.

[6] C.W. Hsu, C.C. Chang, C.J. Lin, et al. A practical guide to support vector classification, 2003.

[7] R. Ierusalimschy, Celes W., and L.H. De Figueiredo. *Lua: a powerful, fast, lightweight, embeddable scripting language*, 1993. Software available at `http://www.lua.org/`.

[8] Y. Jin and B. Prabhakaran. Knowledge discovery from 3d human motion streams through semantic dimensional reduction. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 7(2):9, 2011.

[9] G. Johansson. Visual perception of biological motion and a model for its analysis. *Attention, Perception, & Psychophysics*, 14(2):201–211, 1973.

[10] D. Kulić, W. Takano, and Y. Nakamura. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains. *The International Journal of Robotics Research*, 27(7):761–784, 2008.

[11] Q. Miao, H.Z. Huang, and X. Fan. A comparison study of support vector machines and hidden markov models in machinery condition monitoring. *Journal of Mechanical Science and Technology*, 21(4):607–615, 2007.

[12] T.B. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2):90–126, 2006.

[13] K. Murphy. Hidden markov model (hmm) toolbox for matlab. *Online at* `http://www.ai.mit.edu/~murphyk/Software/HMM/hmm.html`, 1998.

[14] R. Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[15] D. Roetenberg, H. Luinge, and P. Slycke. Xsens mvn: full 6dof human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV, Tech. Rep*, 2009.

[16] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, volume 2, page 7, 2011.

[17] Y. Song, D. Demirdjian, and R. Davis. Multi-signal gesture recognition using temporal smoothing hidden conditional random fields. In *Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pages 388–393. IEEE, 2011.

[18] Y. Song, D. Demirdjian, and R. Davis. Tracking body and hands for gesture recognition: Natops aircraft handling signals database. In *Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pages 500–506. IEEE, 2011.

[19] T. Starner and A. Pentland. Real-time american sign language recognition from video using hidden markov models. In *Computer Vision, 1995. Proceedings., International Symposium on*, pages 265–270. IEEE, 1995.

[20] Vicon Motion Systems. The vicon commercial website. *Online at* `http://www.vicon.com`, 2013.

[21] J.Y. Wang and H.M. Lee. Recognition of human actions using motion capture data and support vector machine. In *World Congress on Software Engineering*, pages 234–238. IEEE, 2009.

[22] R.Y. Wang and J. Popović. Real-time hand-tracking with a color glove. In *ACM Transactions on Graphics (TOG)*, volume 28, page 63. ACM, 2009.

[23] G. Welch and E. Foxlin. Motion tracking: No silver bullet, but a respectable arsenal. *Computer Graphics and Applications, IEEE*, 22(6):24–38, 2002.

[24] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 379–385. IEEE, 1992.

[25] M.H. Yang and N. Ahuja. Recognizing hand gesture using motion trajectories. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1. IEEE, 1999.

[26] L. Zhang, B. Curless, and S.M. Seitz. Rapid shape acquisition using color structured light and multi-pass dynamic programming. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pages 24–36. IEEE, 2002.

List of Figures

List of Tables

Appendix A

Reproducing the results

It is possible to reproduce the results of the experiments in this thesis by downloading the software and the data files and running the experiments yourself.

A.1 Platform

The software was written for Matlab 7.12.0 (R2011a) and has been used on Windows XP and Ubuntu Linux. Most of the software can be run headless to perform the experiments on a Linux server. The HMI server that most experiments were run on uses Matlab 7.5.0 (R2007b).

A.2 Data

The motion capture data that this thesis uses was recorded using Moven Studio and exported to a plain text format using a small export tool of our own making. This tool basically just dumps the data, using the Xsens DLL to read the data from the `.mvn` file. These plain text files, which are quite large, were then imported in Matlab and stored as a binary `.mat` file, which is much smaller and faster to load.

A.3 Fast workflow

If you just wish to run all the tests again and generate all the data and graphs again yourself, follow this procedure. But be warned that this will take many days to complete and that it may give errors as all experiments were tested individually, not at once.

```
> init
> runTests
> runExperiments
> renderGraphs
```

A.4 Structure of the software

The software is structured in five directories. In the root of the project we only find two initialization scripts and a script to render graphs. The `init` script will load all the motion capture data from file and generate features from it. These features are then split up in training, validation and test sets. These last two steps however are also performed by many of the tests and experiments because they have to be tweaked for each one. The `init_fast` script will load the motion capture data, the features and the sets straight into memory from a `.mat` file, which saves a lot of time. The `renderGraphs` script does just that; it will render the selected graphs from data in files at `tests/results` to graphs which it stores as EPS files in `tests/figures`.

The `data` directory contains all the motion capture data from the experiment, the `.mat` file that `init_fast` utilizes and an `initdata` script that loads in all the data and cuts it into classes.

The `hmm` directory contains the HMM classifier software and some helper scripts. Of these, most notable are `trainhmm` and `classifyhmm`, which both operate on the training, validation and test sets that are generated by the initialization. Typical operation would be:

```
hmmstruct = trainhmm(trainset [, numberofgaussians, numberofstates]);
setwithpredictions = classifyhmm(hmmstruct, testset [, windowsize]);
```

The `svm` directory is structured in much the same way. It contains the SVM classification software and the `trainsvm` and `classifysvm` scripts that can be used in the same manner as the HMM scripts mentioned before. Only the optional parameters differ: `trainsvm` has optional parameters cost and gamma and `classifysvm` has none.

The `test` directory contains all the experiments. Scripts starting with `find` are used to find classification parameters as described in chapter 4.1. Scripts starting with `test` are used to find the answers to the research questions using the experiments described in chapter 5. You can run all these scripts individually. Each script will calculate the result for its experiment and store the result as a `.mat` file in the subdirectory `results`. The subdirectory `figures` contains the output from `renderGraphs`, which converts the data in `results` to good looking graphs.

The `util` directory, finally, contains all the tools necessary to work with the data and the features. Some notable scripts:

- The `dataToFeatures*` series of functions convert the raw motion capture data to the features described in chapter 3.2.
- The `build*Sets` series of functions split the feature data in sets that can easily be used in experiments, like train, validation and test sets.
- The `fancy*` series of functions are used to generate good looking graphs and height maps.
- The functions `addToSet`, `joinSets`, `makeSet` and `splitSet` can be used to operate on the sets that the `build*Sets` series of functions generate.
- Finally, there are several `show*`, `plot*` and `play*` functions. These can be used to visually display the motion capture data.