Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Spatial Predictive Models of Object Manipulation

Jan Hoffmann

Course of Study:	Informatik
Examiner:	Ph.D. Daniel Hennes
Supervisor:	Ph.D. Jim Mainprice
Commenced:	December 20, 2017
Completed:	June 20, 2018

Abstract

Human-robot interaction requires good coordination so that the human and robot can perform their tasks without interruption. Most importantly, the robot needs to be able to react to movements of the human in order to prevent harm to the human or objects. A basic necessity is tracking of the current position of the human and objects. For this, existing motion tracking systems can be used to track both the human and objects in the environment. In addition, information about future movements can be useful to improve the interaction. In this thesis, methods for detection and prediction of human activities using machine learning are developed and evaluated. Our system can discern a set of different activities based on 3-dimensional position data of objects and the human skeleton together with the eye gaze direction. Further, it generates trajectories of possible future movement for each of these activities. These are scored in order to anticipate the most likely activity in the near future. Our results show an improvement in activity recognition performance from using gaze data.

Kurzfassung

Mensch-Roboter-Interaktion erfordert eine gute Koordination, sodass Mensch und Roboter ihre Aufgaben ungestört erfüllen können. Besonders wichtig ist es, dass der Roboter auf Bewegungen des Menschen reagieren kann, um Verletzungen und Schäden an Gegenständen zu verhindern. Eine grundlegende Notwendigkeit ist die Erkennung der aktuellen Position des Menschen und von Gegenständen. Zu diesem Zweck können bestehende Motion-Tracking-Systeme benutzt werden, um sowohl den Menschen als auch Objekte in der Umgebung zu verfolgen. Zusätzlich können Informationen über zukünftige Bewegungen nützlich sein, um die Interaktion zu verbessern. In dieser Arbeit werden Methoden zur Erkennung und Vorhersage von menschlichen Aktivitäten mithilfe von maschinellem Lernen entwickelt und evaluiert. Unser System kann eine Menge verschiedener Aktionen aufgrund dreidimensionaler Positionsdaten von Objekten und des menschlichen Körpers zusammen mit der Blickrichtung unterscheiden. Darüber hinaus erzeugt es Trajektorien möglicher zukünftiger Bewegungen für jede dieser Aktivitäten. Diese werden bewertet um die wahrscheinlichste Aktivität in der nahen Zukunft vorherzusagen. Unsere Ergebnisse zeigen eine Verbesserung in der Leistung der Aktivitätserkennung durch die Nutzung von Blickdaten.

Contents

List of Figures

List of Tables

1 Introduction

There are many different applications for robots. Personal robots support individuals during their everyday lives, for example by automating repetitive household tasks. Such a robot operates in the same space as a human, and often interaction with the human is required during operation. This leads to different safety and coordination requirements. A robot must not harm humans or objects in its environment. To be useful, it also should not interfere with actions performed by a human. Ideally, it should be able to work together with the human and perform tasks collaboratively.

In order to fulfill these goals, good motion planning is required. To implement this, the robot needs to have a model of its environment and what is happening around it. Knowing future trajectories of the human allows to avoid performing actions that would interfere with them or even cause harm. Information about current and future activities can be useful for collaborative work. By using anticipation, the robot can prepare its next action earlier and decrease reaction time.

For activity recognition and anticipation, human motion data is required. Existing motion tracking systems can be used for this. In addition, gaze is also a possible indicator for future movements of a human, as it often precedes the actual movement. Previous work on activity recognition, such as the one of Koppula et al. [KGS13], does not make use of gaze. One of the goals of our work is to evaluate the impact of gaze data on the quality of activity recognition.

Our system provides activity detection using different machine learning models. We also implemented a method for activity anticipation that makes use of the existing models for activity detection. During the activity anticipation process, trajectories for future movement are also generated. For training and validation of our system, we recorded a dataset of an actor performing different activities.

We compare two machine learning methods, a Support Vector Machine (SVM) with hand-crafted features, and a neural network with Long Short-Term Memory (LSTM) layers using raw data as input. The SVM shows the best results for activity recognition with a F_1 score of 0.84, and for activity anticipation with a score of 0.37. Comparison of the SVM shows an improvement by 0.03 from inclusion of gaze data.

Structure

The thesis is structured as follows:

Chapter 2 – Background: In the next chapter, related work on activity recognition and anticipation is presented. We also show some background on the behavior of human gaze. Afterwards, there is a short overview of the used machine learning algorithms.

Chapter 3 – Data Capture and Processing: Here, an overview of our recorded dataset is given. It starts with the model of the scene which describes the contained objects and the human skeleton. Then, the objects in the scene and the way how the actor interacts with them is described. We also show how we calculate features that are used for activity recognition and anticipation.

Chapter 4 – Activity Recognition: This chapter describes the models for activity recognition that we implemented. We use two different approaches, a SVM based on features, as well as two LSTM-based models using raw data from the dataset.

Chapter 5 – Anticipation Of Future Activities: We implemented an activity anticipation method that samples future trajectories for multiple activities. It makes use of our activity recognition models for finding the best trajectories and activities.

Chapter 6 – Evaluation: In this chapter, the performance of our system is evaluated. Following a short description of the used metrics, the results are presented. We compare the different models and discuss possible reasons for the results.

Chapter 7 – Discussion: The last chapter gives a short summary of our work, followed by a lookout on possible changes for further improvement.

2 Background

In this chapter, some relevant related work for activity recognition and anticipation, as well as learning from human motion in general is presented. This also includes some background on human gaze behavior. Afterwards, we give a short overview over the learning algorithms used for our implementation.

2.1 Related Work

There has been various prior work related to human activity recognition and anticipation. One of them is by Koppula et al. [KGS13; KS13; KS16]. They recorded a dataset consisting of 120 activity sequences for 10 high-level activities.

Koppula et al. use RGB-D data, which they segmented and labeled with sub-activities, as well as object affordances. The sub-activities consist of ten labels: reaching, moving, placing, opening, closing, eating, drinking, pouring, scrubbing, and null. Affordances describe how a human can interact with an object, such as reachable or movable.

Their activity recognition method is based on a structural Support Vector Machine (SVM), and can predict object high-level activities, sub-activities, as well as object affordances. To do this, they use a Markov Random Field (MRF) structure to model the input data. It is a graph that contains objects and human pose as nodes, and interactions between them as edges. There are also links between nodes for different time steps.

Each of the nodes and edges in the MRF corresponds to a set of features. Those features consist mainly of positions and distances. For objects, there is a 2D bounding box from the recorded image data. For the human body, several body and hand pose features are included.

For activity anticipation, Koppula et al. use a graph structure which they call Augmented Temporal Conditional Random Field (ATCRF), as shown in Figure 2.1. The idea is to take the existing CRF structure for past segments, and extend it by sampled future segments. For this sampling, first a segmentation and labeling for the past CRF is generated. Then, possible future sub-activities and affordances are sampled. Using these, future object locations are sampled from affordance heatmaps. Based on the target locations of the objects, trajectories and poses for future frames can be generated. These are then used to augment the CRF to ATCRF particles. For each ATCRF, features and the best segmentation are calculated. Finally, a weight is calculated for each ATCRF particle. Based on the weights, the particles are sorted and the best three ones are the output, together with their sub-activity and affordance labels.

In general, there are a lot of similarities between the work of Koppula et al. and ours. Our activity recognition is comparable to the sub-activity recognition of Koppula et al. We also use sampling and rating of future trajectories to perform anticipation of future activities. While Koppula et al.

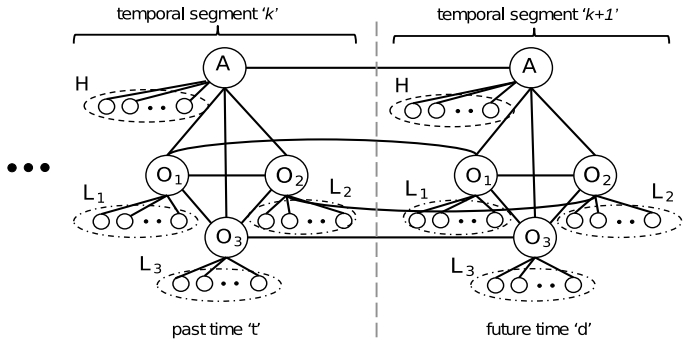


Figure 2.1: ATCRF structure as used by Koppula et al. [KS13] for activity anticipation. Two temporal segments are shown, one for the recent past and one for the future. For every frame, the human poses $\mathcal{H}$, and object locations $\mathcal{L}$ are represented. Every segment has object affordance labels O and sub-activity labels $\mathcal{A}$.

use RGB-D data, we replace that by 3D motion tracking of the human and objects in the scene, and add tracking of the human gaze. Our SVM-based model has a simpler construction, as it is just a standard SVM model. We also do not perform explicit segmentation of the input data, instead the model uses data from a fixed-length period of past movement. In addition, we add a model based on a Long Short-Term Memory (LSTM) neural network that does not require calculation of features.

Others have also applied different learning algorithms to the dataset that Koppula et al. recorded. Jain et al. [JZSS16] use a structural Recurrent Neural Network (RNN) model. It takes into account spatial human-object and temporal human-human information for affordances, and spatial human-object, spatial and temporal object-object data for sub-activities. Truong and Yoshitaka [TY17] use a structural LSTM model that adds spatial and temporal object-object data for sub-activity prediction. However, both approaches still use features as input data. Our LSTM-based model has a similar structure to the one from Truong and Yoshitaka, but uses raw motion data instead of features.

There are also implementations of activity recognition for different datasets, which may contain different sets of activities. Still, similar methods can be used for recognition of these activities.

Jain et al. [JSK+16] implemented a system that anticipates driving maneuvers, with the purpose of warning and preventing dangerous maneuvers. The dataset consists of recorded driving data for a total of 1180 miles from ten drivers. A set of five different maneuvers is predicted, consisting of turns and lane changes in both directions, as well as straight driving. The system is able to predict maneuvers several seconds in advance. Data from different sensors is combined and used as input. A camera detects the 3D head pose of the driver, and a road-facing camera provides context. Additional features include GPS data, the vehicle speed and road information, such as intersections. A simple RNN which simply concatenates data from different sensor streams is compared with a fusion RNN. The fusion RNN uses separate RNNs for each sensor stream, and performs a fusion with a fully-connected layer afterwards. The results show that the fusion RNN improves precision and recall by 6 percent points. Similarly to Jain et al., we compare a simple LSTM model with a structural LSTM model for our dataset.

Sharoudy et al. [SLNW16] recorded a large RGB-D dataset with 40 subjects, 80 different camera viewpoints, of which 3 are used at the same time, and 60 activity classes. These activity classes include daily activities like drinking, eating or reading, health-related activities such as sneezing,

and mutual activities like punching or hugging. To reduce differences between subjects, the recorded skeleton is normalized in size, and the coordinate origin is set based on the spine of the actor with the largest movement. The body is split into five body part groups, namely the torso, both arms, and both legs. Compared to previously discussed methods, Sharoudy et al. use a different approach for implementing this structure in their model. The LSTM unit itself is modified to be part-aware. Each part has its own input and forget gate, as well as memory, but there is a shared output gate for all parts. This part-aware model improves accuracy by around 3 percent points.

Human motion datasets can also be used to learn the motion behavior and imitate it with a robot. The design choices for learning algorithms for motion imitation can also be relevant for activity recognition. In addition, models for motion imitation could also be used to improve anticipation performance.

Shon et al. [SGR05] learn human poses for robot imitation using Gaussian processes. The observations of the human are transformed to a low-dimensional latent variable space. This is the transformed to a high-dimensional space based on the degrees of freedom of the robot. They compare using a single 10D latent variable space to using separate 2D spaces for each of the five different body segments. As expected, the results show that separate latent variable spaces improve the results.

Duarte et al. [DTC+18] use very similar capture system to our implementation. They combine motion tracking with a headset for gaze tracking. The goal is developing a robot that uses motion and gaze for non-verbal communication that can be understood by humans. For their study, a tracked actor and three participants sit at a table. The actor either places a ball in front of one of the participants, or gives it to one of them. To evaluate how well humans are able to detect intention from gaze and motion, they are asked to guess the action and recipient from watching recorded video data of the actor. The same is done for a robot that has learned gaze behavior based on the recorded data. The results show that predictions by humans improve when the robot uses gaze. For placing, the fixation point moves to the target position, whereas for giving, the hand or face of the recipient are fixated.

In general, people look at objects ahead of manipulating them. The gaze direction is typically tied to the task and the intended action. Gaze changes occur before hand movement, and may even shift to the next object before the current action is complete [AS17]. This makes gaze data a promising indicator for use in activity anticipation.

Huang and Mutlu [HM16] performed a robotic experiment including anticipation using gaze. In the experiment, a robot prepares smoothies for a user, who looks at a menu containing twelve different fruits. The user can order smoothies with a spoken command. Anticipation of the user's choice is performed using an SVM based on the number and duration of glances at different fruits. The robot can then anticipate to prepare the smoothie before the user has ordered it, resulting in a shorter preparation time. An evaluation of the results shows an accuracy of 76 % at 1.8 seconds before the order. This experiment shows that anticipation using gaze data can improve the interaction between a robot and a human.

2.2 Learning Algorithms

2.2.1 SVM

The Support Vector Machine (SVM) is a binary classifier, with the simplest form being the linear SVM. It is based on a vector space representation, thus each input sample is represented as a vector. The model is trained to determine the hyperplane which separates the two classes best, as shown in Figure 2.2. The best hyperplane is the one with the highest margin, which is defined by the perpendicular distance to the nearest sample. Since the margin is maximized, it is also called a maximum margin classifier. It has a sparse solution, as the separating hyperplane is defined only by the support vectors, the samples which lie on the edge of the margin. As the corresponding optimization problem is convex, every solution represents a global maximum. [Bis06]

However, a typical dataset is not linearly separable. A possible solution for this is to transform the vector space into a higher-dimensional feature space. The kernel trick allows to use specially constructed kernel functions so that the higher-dimensional space does not actually need to be computed. Using this trick, even kernels that correspond to an infinite feature space can be used. Another way to handle non-linearly separable data is to allow misclassification for a fraction of data points. [Bis06]

2.2.2 LSTM

A neural network consists of a set of neurons with activation function. They are connected to each other, as well as inputs and outputs via directed weighted connections. The learning process involves finding weights so that the network produces the desired output. A common method for this is backpropagation: The input data is first run through the network in forward direction. Then the error is calculated by comparison with the target output. The error is then propagated through the network backwards, while updating the weights. [Sch15]

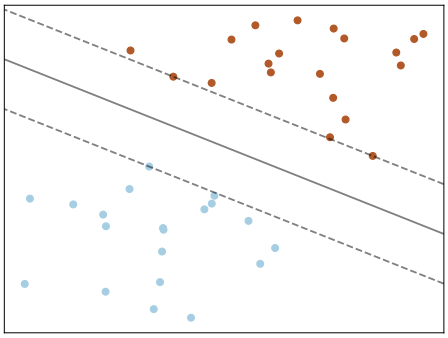


Figure 2.2: Example of an SVM hyperplane with margin for a simple 2D dataset.

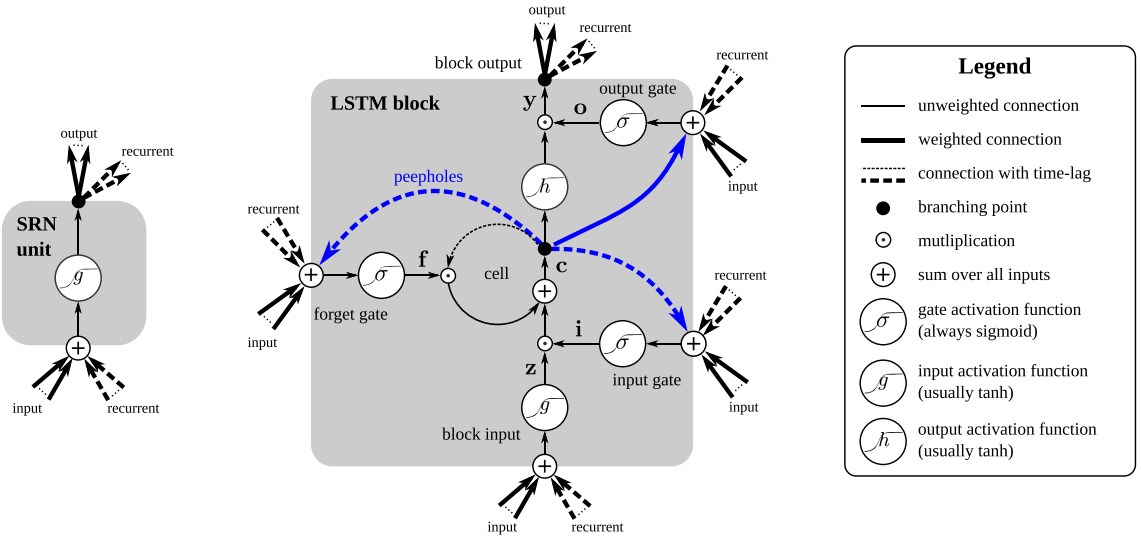


Figure 2.3: Structure of a simple recurrent unit and an LSTM unit, taken from Greff et al. [GSK+17]

A Recurrent Neural Network (RNN) is defined by having a cyclic graph. This means that neuron outputs are fed back as inputs. RNNs are very powerful, and can theoretically be trained to replicate any algorithm, if the network is large enough. A problem with traditional RNNs is the vanishing or exploding gradient issue. For long sequences, the backpropagated values shrink or grow rapidly, causing the network not to remember data about those sequences. [Sch15]

The Long Short-Term Memory (LSTM), which was presented by Hochreiter and Schmidhuber [HS97] in 1997, provides a solution for this issue. Figure 2.3 shows the construction of an LSTM unit. The Constant Error Carousel (CEC), named as *cell* in the figure, is connected to itself and stores the unit's state. The connection has a constant weight, fixing the vanishing/exploding gradient problem, and allowing the unit to memorize data over thousands of time steps. [Sch15]

In the figure, data passes from the bottom to the top. Different gates with their own activation functions control access to the CEC. The input gate controls write access, the forget gate allows resetting the state, and the output gate controls read access. Peepholes are an extension, that allow the gates to see the state, even when the output gate is closed. [GSS02]

Every LSTM unit has 15 weights that are learned during training. For the unit's input and each gate input, there are weights for the input and the recurrent connections. Additionally, there is a bias weight for each of those. Finally, each of the peephole connections is weighted. [GSK+17]

Since the storage capacity of a single unit is not enough, many of them are combined in a single network. It is possible to place multiple units in parallel, which all have the same input. This defines a single layer with its size determined by the number of units. Multiple layers can be stacked, by connecting the outputs of one to the inputs of the next.

Greff et al. [GSK+17] evaluated LSTM networks for different parameters. Among those are the layer size and learning rate, as well as structural changes to the LSTM unit structure. They found that the different gates are important for the functionality of the network. However, the peepholes did not add a significant improvement to the results for the tested datasets. The most important parameter is the learning rate, which needs to be tuned properly to get good results. This can be

done by gradually using a lower learning rate until results no longer improve. The layer size is also important, as it needs to be large enough for a given dataset. For further increases there are diminishing returns.

2.2.3 Comparison

An SVM does not require a large dataset for training. In fact only a small part of the samples is used to define the separating hyperplane, which is a global optimum. However, it requires pre-calculated features to work. A more powerful model like an LSTM-based neural network can provide end-to-end learning from raw data instead. In order to provide good results, it needs a lot of input data, though. Due to the learning process, it only reaches a local optimum.

3 Data Capture and Processing

We have recorded motion and gaze data in a scene where an actor interacts with different objects. The scene consists of an area of around 3×4 meters where the objects are placed. Figure 3.1 shows the actor within the scene.

For tracking, commercially available systems for motion tracking and gaze tracking are used. A marker based motion capture system tracks movement of the human movement of the actor, as well as rigid bodies for each of the objects in the scene. A headset worn by the actor records gaze information by tracking the actor's pupils. The data from both systems is merged to obtain the state of the scene.

To handle the data, we make use of the Robot Operating System (ROS)[1], which provides some useful software components, such as the visualization tool RViz, which can be used to view the recorded data. An example of this is shown in Figure 3.2.

More details about the technical implementation of data capture and processing are presented in Appendix A.

3.1 Scene Model

Our scene contains a set of objects and the skeleton of a human actor. The actor interacts with the objects in different ways depending on the type of object and the specific activity.

Figure 3.1: Actor wearing the motion capture suit and gaze headset scrubbing a table.

[1] http://www.ros.org/

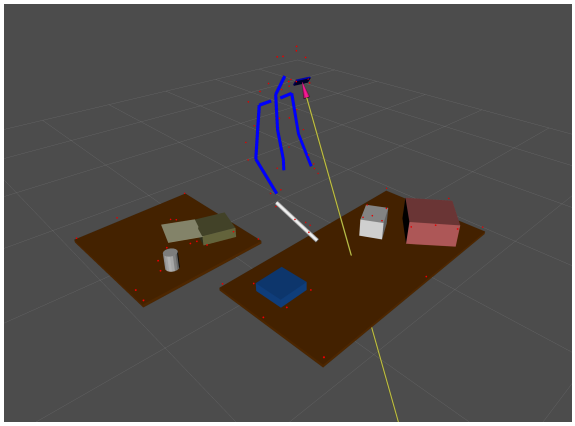


Figure 3.2: Visualization of the recorded scene in RViz, showing an actor scrubbing a table while looking at it.

The raw motion tracking data contains for each rigid body the centroid as 3D coordinate, the orientation as 4D quaternion, and the unique identifier of the object. In order to represent the data consistently and perform calculations on it, the coordinate system needs to be defined. The x and y axes lie in the ground plane, with the z axis going upwards.

To process this data in a useful way, semantic information about the objects is required. Thus, the object identifiers map to a manually defined description of the object. Each object description contains a set of values:

- The **object type** is a human-readable string, which is used to label the object in the output of some processing scripts.
- Objects can have different **shapes**, either they are a cube or a cylinder. This attribute is only used for visualization to make objects easier to recognize. For data processing we treat every object as a cube, making calculations simpler. This does not cause large errors, as all cylindrical objects in our scene are small.
- The **scale** of objects is given as the size of the object in x-, y- and z-directions in its local coordinate system. This property is used for both visualization and processing.
- The **color** allows objects to be discerned more easily in the visualization. It is given as the alpha, red, green, and blue components of the color.
- To describe the ways an object can be interacted with, a list of **properties** is defined. *Movable* objects can be grabbed by the actor and placed on any object with the *plane* property. A *pourable* object holds content that can be poured into a *pourto* object. Similarly, an object with the *drinkable* property can be drank from. The *openable* property marks an object that acts as a lid for another object and can be opened. A *scrubber* object can be used to scrub any other *scrubbable* object.

The skeleton consists of a set of body joints, each of which is stored in the same way as a rigid body. Each skeleton joint id maps to a specific body part. In the visualization, the skeleton is represented using a set of lines.

Object	Shape	Scale (in cm)	Properties
table	cube	80 × 80 × 2	plane, scrubbable
bigtable	cube	160 × 80 × 2	plane, scrubbable
cup	cylinder	8 × 8 × 10	movable, pourable, drinkable, scrubbable
bowl	cube	20 × 20.5 × 6.5	movable, pourto, scrubbable
scrubber	cylinder	3 × 3 × 37	movable, scrubber
box	cube	20 × 15 × 13	movable, plane
bigbox	cube	31 × 31 × 16.5	movable, plane
openable_base	cube	20.5 × 20 × 6.5	movable, pourto
openable_lid	cube	20.5 × 20 × 0.1	openable

Table 3.1: List of objects in the scene. Each of the objects has a different size and different properties describing how the actor can interact with them.

3.2 Objects

Our scene contains a total of eight objects that the actor interacts with, which are shown in Table 3.1. Two of the objects are tables in different sizes, which remain static during the recording. In the model, only the surfaces of the tables are represented, as there is no interaction with any other part.

The other six objects can be moved by the actor. For some of these objects we use placeholders made out of cardboard instead of the actual object being represented. The movable objects include two types of containers: a cup and a bowl. For the bowl we use an open cardboard box as substitute. The cup is drinkable and pourable, however for simplification the recording process no actual liquid is used, so the actor simulates these actions.

A scrubber can be used to clean other planes and containers. We use a stick with a cleaning cloth attached to one end of it. Figure 3.1 shows an example of an actor scrubbing a table.

Two boxes of different sizes are moved by the actor and can be used to place other objects on them. An additional box has an openable lid, and can be used as a container. In the model, this box is represented as two separate objects. The base of the box is a movable container and the lid can be opened and closed.

3.3 Dataset

We recorded a total of 132 minutes of data for training and validation of our prediction algorithms. Two different actors interact with the objects in the scene. While both actors are right-handed, the actors do not always use the same hand for a given task, and sometimes use both hands for tasks where this is reasonably possible. For example, two hands may be used to move a box, or when holding the cup while scrubbing it.

The interaction with the objects includes a set of seven different types of interaction: *reaching* for an object, *placing* an object on another object, *pouring* into another object, *drinking* from an object, *opening* or *closing* an object, and *scrubbing* an object. In order to get a balanced selection of interaction, actions and matching target objects are chosen at random by a script running on the recording laptop. The *reaching* action is not explicitly included in the instructions, but has to happen implicitly in order to interact with any movable object in the scene. The instructions are passed to the actor via speech synthesis. In order to filter actions that are impossible under the current state of the objects in the scene, and to control the timing of actions, a human manually screens actions and triggers sound playback.

3.3.1 Labeling

In order to apply learning algorithms to the recorded data, ground-truth activity labels are required. Thus, we segmented and labeled the entire recorded data manually.

For the labels, we use the seven interaction types described above, i.e. *reaching*, *placing*, *pouring*, *drinking*, *opening*, *closing*, and *scrubbing*. In addition to that, we add a *null* label for segments where the actor does not interact with any object, for example because of waiting for the next instruction. Thus, our data is labeled with a very similar set of activities compared to the work of Koppula et al. [KGS13], except that we do not have the *eating* activity and do not label *moving* segments.

For the *reaching* activity, it is not inherently clear when the corresponding segment should start or end. We define the start point of *reaching* segments as the time when the hand starts to move into the direction of the object or the actor's gaze moves towards the object. The segment ends when the user has taken hold of the target object. Segments of any other label always directly follow either a *reaching* segment or any other non-*null* segment. The time during which an object is being moved to its target location is included within the segment of the following activity.

Sometimes, the actor starts performing a new activity while the current activity is not completed. An example for this is when the actor is about to pour into the openable box while it is still closed. While the box is still being opened with one hand, the actor may already start to pour with the other hand. Similarly, the actor may start to place the cup while closing the box. Since our architecture only supports a single label for each segment, such cases cannot be represented perfectly. For each of these cases, we try to find the best split point by looking at the angles and distances of the involved objects.

After tagging the data from all recording sessions, we remove the *null* segments at the beginning and end of each session. The resulting dataset consists of 3042 segments with a total duration of 125 minutes. Table 3.2 shows the distribution of labels within the dataset. Despite randomized selection of actions during recording, the number of segments per label is not entirely balanced. This is especially visible for *null*, *reaching* and *placing*, which all occur in many more segments than the other labels. However, this is difficult to avoid, as for all non-*null* activities it is necessary to reach for the object first, and for most activities the object has to be placed afterwards. Due to differences in the average segment durations, the imbalance increases when looking at the fraction of time per label, which is the relevant metric for our activity recognition and anticipation methods.

Label	Number of segments	Fraction of time	Average duration
null	26.4 %	31.7 %	3.0 s
reaching	30.6 %	22.7 %	1.8 s
placing	24.6 %	18.4 %	1.8 s
pouring	3.8 %	4.1 %	2.7 s
drinking	2.7 %	2.3 %	2.1 s
opening	3.5 %	2.0 %	1.4 s
closing	3.5 %	1.7 %	1.2 s
scrubbing	4.9 %	17.0 %	8.6 s

Table 3.2: Distribution of labeled segments within the recorded dataset. The shown values are after removal of leading and trailing *null* segments for each recording session.

3.4 Features

In order to have suitable input data for activity recognition and anticipation, we calculate a variety of features from the raw recorded data. These consist mainly of distance and orientation features. Our implementation allows to calculate all of the features for recorded data files as well as for live real-time data read from ROS.

The features are structured in different parts for the gaze direction, the skeleton, and the objects. As a basis for calculating these features, we use the recorded data as described in Section 3.1 and Section 3.2.

For human gaze, there is an origin coordinate and a point in the distance, which defines the gaze vector. For the skeleton, we consider only joint data from the body parts that interact directly with an object. Thus, we include the head for the *drinking* activity, and both hands.

For every object, the centroid and orientation are included in the recorded data. In addition, the object properties which are mapped to every object are partially used for feature calculation: We take into account the object size for all calculated features. However, in order to simplify the implementation, all objects are treated as cubes. This does not result in a significant amount of error, as all cylindrical objects in the scene are small in size.

For objects, we calculate the angle relative to its canonical orientation. This is done by taking the normal of the top surface and calculating the angle from the world z axis. The resulting angle tells how much an object is tilted, which happens when *drinking* or *pouring* from an object.

Distance features are included for objects with respect to gaze as well as the skeleton. The gaze distance is calculated as the smallest distance between the object and the gaze vector. To do this, first the nearest point on the object surface is calculated and then the distance from there to the gaze vector. The gaze distance is undefined when the gaze vector points in the opposite direction than the object. For the skeleton, the distances between the object surface and the head as well as the hand joints are calculated.

For objects that have a plane where other objects can be placed, an additional gaze feature is calculated. The 2D gaze coordinates of the plane are given by the intersection of the gaze vector with the plane surface. No truncation to the actual size of the plane is performed, thus also coordinates outside the area of the plane are possible.

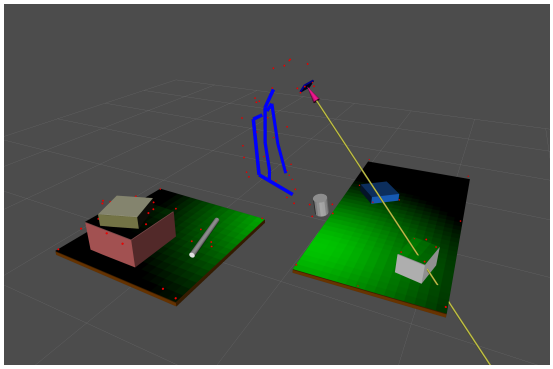
In addition to these object features, we calculate some features between different objects. For this, one object is treated as the primary object, and the second as the reference object.

If the reference object is a plane, the 3D coordinate of the primary object relative to the plane is calculated. For this, a local plane coordinate system with its origin in the center of the plane is used. The x and y coordinate is then given by the position of the primary object's centroid in the plane coordinate system. For the z coordinate, the lowest point of the primary object in its current orientation is used. Thus, the z coordinate represents the vertical distance of the object to above plane.

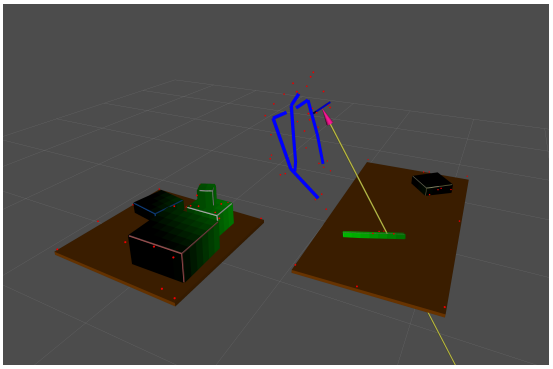
For all object pairs, the distance between the objects is given. In this case, a simplification is used to improve performance and simplify the code. For every corner of the primary object, the distance to the reference object is calculated, and the smallest value is taken as result. This is not correct for cases where the nearest point to the reference object would be on an edge or surface of the primary object. This does not matter much due to the object size, except for the two tables. Since the object-object features are designed to be used with the primary object being an object the user interacts with, this does not cause any issues.

The last of the object-object features is the orientation difference, which is useful to determine whether the lid of an openable object is currently opened. It is defined as the orientation of the primary object relative to the reference object. Additionally, the angle between the objects is calculated in a similar ways as the object angle, by using the angle between the two top surface normals of the objects.

For all of the features, the temporal changes in the near past are calculated. This allows to tell if and how an object is moving. We use the three time steps of 250 ms, 500 ms, and 1000 ms. For each of these time steps, the change from the past value towards the current value is calculated.



(a) Plane heatmaps for the cup while the actor is placing it on the small box.



(b) Object heatmaps for the actor's right hand while they are reaching for the scrubber.

Figure 3.3: Generated heatmaps for planes and objects, as visualized in RViz.

Heatmaps

In addition to calculating features, it is possible to generate different heatmaps. These heatmaps are calculated for surfaces using a grid with a target size of 5 cm. The values are based on distances. Since small distances should have a high value, all distances are clamped to the range between 0 and 1 m, and the value is then inverted. As calculation of heatmap values can be an expensive operation for large areas, they are not calculated by default. Instead, calculation is done explicitly where it is required.

Heatmaps are supported for two cases. For objects it is possible to generate a heatmap for the distance to a given point. In this case, the object is always treated as a cube. The heatmap may be calculated for all surfaces of the cube, or only some of them.

It is also possible to generate heatmaps for the distance of an object to a plane surface. In this case, the value is calculated from the distance between the heatmap area and the object's plane coordinates.

For both types of heatmaps, an optional occlusion check can be enabled. This filters out areas where an object is placed on a plane surface. However, in practice we found that the performance overhead for this check is too high to be useful.

Our code allows to publish heatmaps for visualization in RViz, as shown in Figure 3.3. This is possible for either plane heatmaps with a selected object, or object heatmaps for either the left or right hand of the skeleton.

4 Activity Recognition

4.1 SVM

For activity recognition, we implemented a simple SVM model. This model uses a single feature vector with hand-selected and scaled features, consisting of a subset of the calculated features described in Section 3.4. In addition, object properties from the object model are added.

An overview of the SVM features is shown in Table 4.1. No raw position or orientation data from the motion capture or gaze data is included in the feature vector. Instead, distance and angle features for the objects in the scene are used. The feature vector is constructed in a way so that it is independent of the actual number of objects in the scene. This is realized by only including data for the three nearest objects based on gaze distance, as well as the distances to the left and right hands of the actor.

For each of these objects, the object properties are included. These are encoded in a vector with the length corresponding to the number of possible properties. Each property maps to one of the values which is set to either zero or one, depending on whether the object has the property. From the calculated features, each object has the angle, gaze distance, as well as the distances to the head and both hands of the actor. For each of the three nearest objects, also object-object features are included.

Description	**Length**
Object features for 3 nearest objects to gaze, left hand, and right hand	
Object properties	8
Angle	1 + 3
Gaze distance	1 + 3
Head distance	1 + 3
Left hand distance	1 + 3
Right hand distance	1 + 3
Object-object features for 3 nearest other objects	
Object properties	8
Distance	1 + 3
Angle	1 + 3
Total	684

Table 4.1: Overview of SVM features. This shows the length and structure of the feature vector. The number after the plus sign stands for temporal difference features.

Within these object-object features, the object properties of the reference object are encoded in the same way as described previously. From the calculated features, the distance to the reference object, and the angle between the two objects are included.

For all calculated features, the temporal changes for three time steps are included. No scaling is performed for the majority of features. However, temporal changes of angle features are scaled by a factor of 0.5. All temporal changes are normalized to a duration of 2.5 s, so they effectively represent the speed of movement averaged over different durations. This way, the values for different time steps are comparable. Some features, in particular the gaze distance, do not always have a valid value. These missing values are replaced by zeros in the final feature vector.

In total, the feature vector for the SVM model has a length of 684. When removing all gaze-related features, the length is reduced to 432. This reduction mainly comes from removing the three nearest object relative to the actor's gaze.

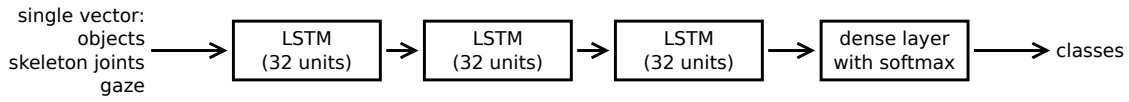
Our model is implemented in Python with scikit-learn [PVG+11]. We use the SVC classifier with the default RBF kernel. In this setup, the two parameters C and γ need to be chosen. C is the penalty parameter in the error term and γ is the kernel coefficient for the RBF kernel. Both parameters can be determined in a grid search. In order to compensate for the varying number of samples for different labels, we use balanced class weights. The weights are inversely proportional to the number of samples in a class. In the model, these are used by multiplication with the parameter C for each class.

As a SVM is a binary classifier, it cannot be used directly for a multi-class problem. A mapping of the problem to a binary classification, as well as the reverse mapping of the results is required. There are two different possibilities to implement this. In a one-vs-the-rest (OvR) scheme, a binary classifier is built for each class, deciding between samples being in the respective class or in any other class. Each of these binary classifiers produces a score, allowing to determine the final label by finding the one with the highest score. The other possibility is a one-vs-one (OvO) scheme, where a classifier is built for every pair of two classes. In this case, a voting scheme can be used to determine the final class.

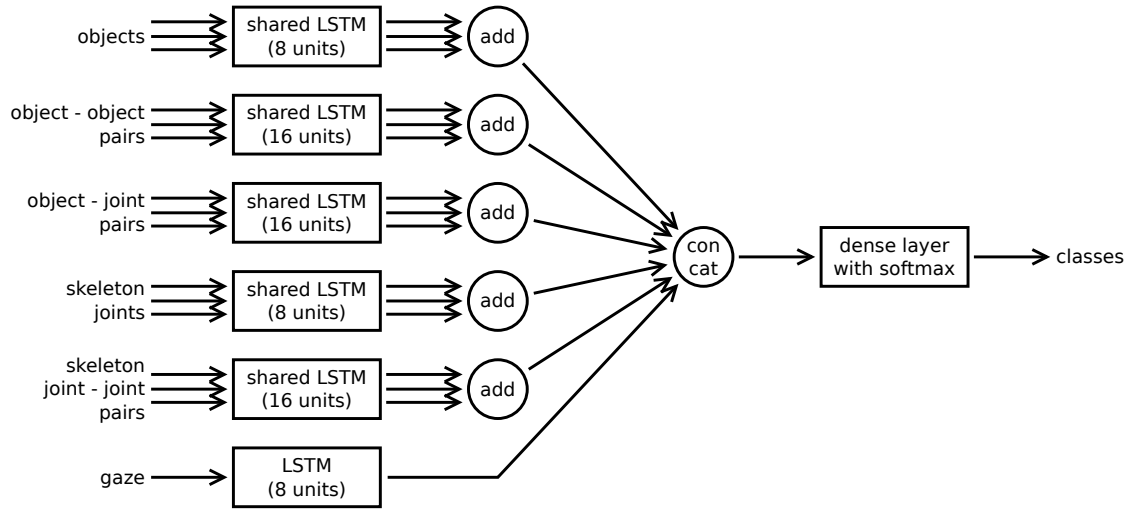
The SVC classifier of scikit-learn uses the OvO scheme internally. The scores are then converted to the shape of a OvR scheme. Thus, the model provides scores for all classes, which are then used to determine the predicted class. In practice, the score for the predicted class is almost always 7.5 for our model. This is not an issue when using the model to predict a label, but the scores are important for our activity anticipation method, as described in Chapter 5.

Then, to get proper scores, we implement two additional variants of the SVM model. The first one is to use an actual OvR scheme for the mapping to binary classification. With scikit-learn, this is possible using a separate OvR classifier that can be wrapped around any binary classifier. Thus, we train a separate model which is constructed from the SVC classifier in binary mode with the OvR wrapper.

Alternatively, scikit-learn provides the option to calculate per-class probabilities for a SVM model. This is internally implemented as a separate process via Platt scaling. It involves doing a regression on the scores, which requires an additional cross-validation on the training set. In addition to the computing overhead, this has the potential issue that the probabilities may be inconsistent with the



(a) Simple LSTM model



(b) Structural LSTM model

Figure 4.1: Structure of our two different LSTM-based neural networks.

prediction. In particular, the class with the highest probability value may not match the predicted class. This also means that the actual prediction model is not changed when probabilities are enabled, so the prediction performance stays the same as for the standard model.

4.2 LSTM

Another possible approach for activity recognition is to use unprocessed raw data using a more powerful model. Thus, we try a recurrent neural network implementation with only unprocessed data as input. As our dataset contains sequential data, an LSTM-based model is a natural choice.

As input data for the model, we use only unmodified raw data from the recording. There is no processing of position or orientation data, and it is instead fed into the model in the existing form of 3D positions and 4D quaternions. In addition to the recorded motion and gaze data, we add some additional information from the object models:

- For **objects**, the recording contains the centroid and orientation for each body. We extend this with the corresponding body properties. The encoding is identical to the one used for the SVM model, thus a vector with a length of the number of properties, with zero and one as values.

- **Skeleton joints** are also represented by a centroid and orientation in the recorded data. For these, the joint identifier is added, so different joints can be distinguished by the model. Here, a vector with the number of joints as length is used. Each value corresponds to a specific joint, and is set to one only for that joint and zero otherwise.
- The **gaze** data is used as is, without any extension. This means the origin point of the gaze vector and a distant point to define its direction.

In order to represent temporal aspects of interaction, we use time sequences. Each time sequence has a length of 11 time steps which are each 100 ms apart. Thus, the time sequence covers a duration of one second. Each time sequence is fed into the LSTM model sequentially in the order of recording, and the output after the last time step is used.

We implemented two models with different complexity:

- The first model has a simpler structure, which is shown in Figure 4.1a. The entire set of input data from objects, the skeleton, and gaze data is concatenated into a single vector. This data is then passed into a network of three LSTM layers with a unit size of 32. In order to get label predictions from the LSTM output, we use a fully-connected layer with softmax activation. It has a size of 8 units, matching the number of labels. Thus, the output is a set of scores where each corresponds to a specific label. In order to train this model, the labels are converted to a vector where only the corresponding item is set to one. For prediction, the index with the highest score tells the label.
- The second model uses a more complex network structure as shown in Figure 4.1b. Different parts of the data and combinations of it are processed separately and the merged afterwards. The basic types of data items are objects, skeleton joints, and gaze. In addition, we add object-object pairs, object-joint pairs, and joint-joint pairs. Each individual data item is passed through a single LSTM layer. For the three basic types, layers with 8 units are used, and for the combined types, layers with 16 units are used. All items of the same type are passed through shared LSTM layers respectively. These shared layers share the same weights, thus only a single layer is thus trained for each data type. For each shared layer, the outputs are merged by adding the individual values. Then, all results are merged into a single vector through concatenation. Finally, the same way as for the simple model, a fully-connected layer with softmax activation is used to get label scores.

For our implementation of these neural networks, we use Python with Keras [Cho+15] and the TensorFlow [AAB+15] backend. As the model performs a classification, we use the categorical cross entropy loss function. As optimizer, we use RMSProp. In this configuration of the model, there are two parameters which affect the training process, the batch size and the learning rate. The batch size is the number of samples which are used for each update of the model during training. We use a batch size of 512, as larger batch sizes provide better generalization. For the learning rate, we keep the default value of 0.001.

5 Anticipation Of Future Activities

In order to anticipate the next activity, we use a method similar to the one of Koppula et al. [KS13; KS16]. First, future states and trajectories are sampled based on the current state. The sampled trajectories are then scored to find the three best ones. By taking the labels of the selected samples, an activity can be provided as output.

5.1 Trajectory Sampling

In order to sample trajectories, first all the features for the current frame are calculated. Then, possible follow-up activities are determined based on the distance features. For all activities that are possible from the current state, multiple trajectories are randomly sampled. Each trajectory encodes the movement of an object or a skeleton joint within the next second.

Since most of the activities are related to an object, we first need to determine whether the actor is currently holding an object. For this, only objects with the movable or openable properties are considered, as those are the ones the actor interacts with. For each hand, the nearest movable and openable object are determined. If the object is within a distance threshold of 20 cm, it is chosen as a candidate object to sample trajectories for. For all candidate objects, trajectories are sampled depending on the object properties: For movable objects, *placing* are sampled, for drinkable objects *drinking*, for pourable objects *pouring*, for the scrubber *scrubbing*, and for an openable object trajectories for *opening* as well as *closing* are sampled. In the case where no object is within the threshold distance of a hand, *reaching* trajectories are sampled. In all cases, a single *null* trajectory is generated. The way how trajectories for different activities are sampled is described below.

For some activities, heatmaps are taken into account for sampling of trajectories. First, the values for all heatmap areas on possible target objects are calculated. Then, areas can be sampled by using importance sampling, so that areas with a higher score have a proportionally higher chance of being selected. For that, first an area is sampled uniformly at random. Then, another uniform random value is chosen, and the area is selected, if its score is higher than that value. Otherwise, another area will be sampled until the condition is fulfilled and an area is selected.

The shape of the trajectory movement is different for each activity. For some of them, splines are used. These are implemented as cubic Bézier curves. The first and last control points are the start and end point of the trajectory. The first in-between control point is defined by adding the movement within the last 250 ms to the start point. This provides trajectories that follow the previous movement of the hand or object. For the second in-between control point the normal of the target surface with the length normalized to 10 cm is added to the end point if it is generated from a heatmap area. In cases where the end point is not located on an object surface, it is used unchanged.

- **reaching**: First, heatmaps are generated for the sides of all *movable* objects in the scenes. The top and bottom are not considered, as objects are typically held by their side. From all the heatmaps a total of 5 areas are sampled. To get the target location for the *reaching* movement, the center of each sampled area is used with a distance offset of 5 cm. The movement of the resulting trajectory is defined by a spline curve.
- **placing**: Heatmaps for all other objects with the *plane* property are generated and used to sample 5 areas. The target location and orientation are set so that the object sits directly on the plane in its canonical orientation with the center above the sampled area. For the *scrubber*, we take into account that it has to be placed on its side instead of the bottom surface. For the trajectory movement, a spline interpolation is used. The orientation change is linearly interpolated.
- **drinking**: As target position, a point in front of the skeleton head is defined. Then, 5 target positions for trajectories are sampled via a normal distribution with standard deviation of 3 cm. The target orientation is sampled from a normal distribution with a mean of 90° and standard deviation of 5°. The trajectory movement is interpolated using a spline and the orientation change is interpolated linearly.
- **pouring**: All *pourto* objects within a distance of at most 1 m are considered as possible pouring targets. From these, 5 targets are sampled via importance sampling based on a distance-based score. For each target object, the actual target position is sampled from a normal distribution around a point above the center of the container. The standard deviation is derived from the size of the target container. For the target orientation, a normal distribution with mean of 145° and standard deviation of 10° is sampled. The resulting trajectory is interpolated using a spline for the movement and linearly for the change in orientation.
- **scrubbing**: As candidates for target objects, all *scrubbable* objects within a distance of 1 m are considered. Similar to the *pouring* activity, out of these candidates 5 targets are sampled with importance sampling. However, the way of constructing the actual trajectory is very different for *scrubbing*: It distinguishes itself from the other activities in that is is defined mostly by the characteristics of the actual movement, and not just by the target location. Thus, for each target, multiple points are sampled to build the trajectory from. For the *plane* targets, these points are sampled via importance sampling from the surface heatmap. For containers, a uniform sampling of points within the object is used. The orientation of the scrubber is sampled from a normal distribution. The parameters are chosen based on the *scrubbable* object size in a way so that the scrubber does not intersect the walls of containers. As an example, the angle of the scrubber needs to be steeper for a cup than for a bowl to fit into the opening. The resulting *scrubbing* trajectories are finally generated by linear interpolation between each pair of points in the trajectory.
- **opening** and **closing**: Whether *opening* or *closing* trajectories are sampled depends on the current angle of the lid relative to the base. If the current lid angle is below 180°, 5 different target opening angles are sampled. As long as the lid angle is greater than 5°, a single closing target angle of 0° is sampled. Trajectories are created from the sampled angles by linear interpolation of a rotation around the hinge axis.
- **null**: The *null* trajectory is always sampled and represents no change at all.

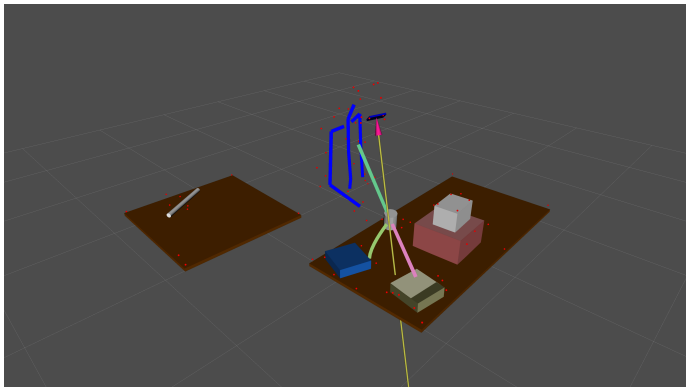


Figure 5.1: Visualization of anticipated trajectories for the cup. Shown here are trajectories for *placing*, *pouring*, and *drinking*.

5.2 Scoring

In order to score the sampled trajectories, we need to apply the trajectory on top of the current state of the scene. To to this, we copy the current frame and then transform the raw object and skeleton data for multiple time steps, up to one second into the future. For *reaching* trajectories only the corresponding hand joint is transformed based on the trajectory. For any other activity, the matching object is transformed according to the trajectory and the nearest hand joint is also transformed to follow the trajectory.

For evaluation of scores for the trajectory, we apply our existing activity recognition models. At the last time step one second in the future, the required input data for each model is extracted and features are calculated, if necessary. In this case, we do not use the predicted label from the model, but instead make use of the per-label scores. Specifically, for each trajectory, the score of its matching label is used. Then, the trajectories are ordered based on their scores and the three highest-scored ones are picked as output together with their labels.

The entire activity anticipation process including sampling and scoring can be run as a batch process on an existing recording, as well as on a playback or live recorded data. It provides the three best trajectories and their labels as output. The trajectories can also be visualized in RViz as colored lines, where the color matches the color scheme used for the original labeling process. An example for this is shown in Figure 5.1.

The process is not suitable for real-time prediction on a typical laptop computer. Depending on the used scoring model, every prediction can take multiple seconds, due to the expensive process of feature calculation for the current and future frames.

6 Evaluation

In order to evaluate the different activity recognition models, we split the recorded data into a training and validation set. The first 80 % of each recorded session are used for training and the remaining 20 % for validation. For each recorded session, the leading and trailing *null* segments are removed. Then, all sessions are merged into a large dataset with a duration of 125 minutes. For all models, a sampling rate of 10 frames per second is used to extract a total of 74 890 samples of input data. After splitting into training and validation set, there are 59 912 samples for training and 14 978 samples for validation.

6.1 Metrics

The results of our activity recognition and anticipation methods are presented in confusion matrices. In a table, the true and predicted labels are compared, where each cell shows the number of samples of a one class that are predicted as another one. This allows to see how the model works for each class, including correct and wrong predictions. Since the number of samples per class is not balanced in the used dataset, a normalization is performed to make the matrix easier to read. For this, the values in each row are normalized to the size of the corresponding class, so that the shown values are fractions of 1. For a perfect model, all the values on the diagonal are ones and all others are zero.

While a confusion matrix shows a good summary of the prediction behavior of a model, an overall metric is desirable to more easily compare different models. The simplest summary metric for a classification problem is the accuracy, which is just the fraction samples that are classified correctly. However, this does not take into account the type of errors, and also is problematic for unbalanced class sizes. As an example, if the majority of a dataset is in the same class, an algorithm that always predicts this class would give a high accuracy, although none of the other classes is every predicted.

Thus, it makes sense to look at per-class metrics and then perform an aggregation into an overall metric. For classification problems, precision and recall are common per-class metrics. The precision tells how many of the samples predicted as a class actually belong to that class:

$$P = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

The recall shows how many of the samples with a true label are correctly predicted:

$$R = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

In a confusion matrix that is normalized per row, the values on the diagonal represent the recall. To get an overall metric, we need to combine precision and recall. The per-class F_1 score is the harmonic mean of precision and recall, defined as:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

As an overall metric we calculate the macro F_1 score as the mean of all per-class F_1 scores. Using the mean value makes the metric independent of unbalanced classes, as each class gets the same influence over the overall score.

6.2 SVM Parameters

For the SVM, the model parameters need to be chosen before evaluation on the recorded data can be done. To do this, we perform a grid search of the two parameters C and γ. In scikit-learn, the default value for the regularization parameter C is 1.0. For γ, the default value is determined as the inverse of the number of features, which is $1/684$ for our model.

We check all power-of-two values in the ranges of $C = 2^{-1} \dots 2^{4}$ and $\gamma = 2^{-13} \dots 2^{-6}$. These ranges were dermined by an initial run of the grid search with a larger value range on a subset of the data. For each combination of these values, our SVM model is trained and validated. A 2-fold cross-validation with the data split into half for training and validation set is used to get more reliable results.

As a metric of performance, the F_1 score is used. The results of the grid search are shown as a table in Figure 6.1. Based on these results we chose the parameters as $C = 2^3$ and $\gamma = 2^{-8}$.

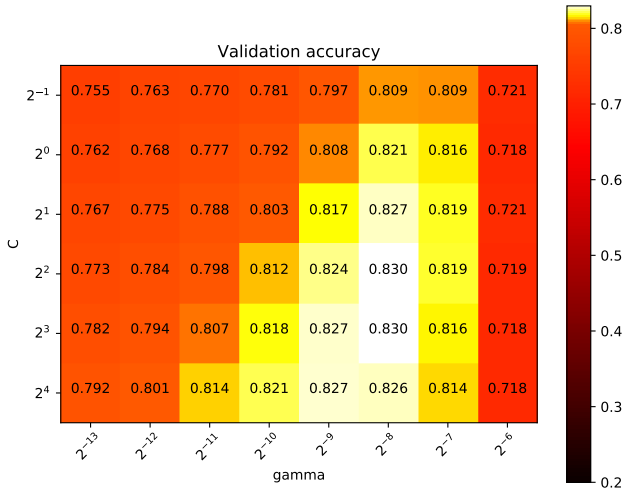


Figure 6.1: Results of SVM parameter search. The table shows the F_1 score for different combinations of the parameters C and γ.

Model	Macro F1 score	Per-label F1 score							
		null	reaching	placing	pouring	drinking	opening	closing	scrubbing
SVM	0.84	0.84	0.76	0.83	0.80	0.88	0.90	0.82	0.93
SVM without gaze	0.81	0.81	0.71	0.82	0.74	0.85	0.86	0.78	0.94
SVM OvR	0.84	0.84	0.76	0.83	0.79	0.88	0.90	0.81	0.92
LSTM simple	0.52	0.74	0.61	0.54	0.41	0.73	0.27	0.26	0.66
LSTM structural	0.73	0.79	0.66	0.70	0.69	0.80	0.65	0.72	0.86

Table 6.1: Results of activity recognition, comparing different models.

6.3 Activity Recognition

We trained all of our models on the training set and compare their results. Table 6.1 shows a overview of the overall macro F_1 scores for each model, as well as the individual per-class F_1 scores. A more detailed discussion of the results follows in the next two sub-sections.

6.3.1 SVM

As described in Section 4.1, we implemented three different SVM models. The standard SVM model uses the default implementation from scikit-learn with an OvO scheme internally. In addition we added a version that uses the OvR mode and another version that provides class probabilities on top of the standard model. The last version produces identical classification results than the standard SVM model, so we do not show results for that variant in this section. However, we add an additional variant for the standard SVM without any gaze-related features. This gives a baseline model to evaluate the influence of adding gaze features. For this, the model definition itself is not changed, but the model is trained on a reduced feature vector.

To know how well the model converges, we generate a learning curve showing the training progress. There is no output of intermediate results during training of a SVM model in our architecture. Instead, a separate process is used to generate a learning curve for the SVM model. We perform this for our standard SVM model.

Calculation of the learning curve works by training the model on training data subsets of different sizes and checking the F_1 score on the validation set. In order to know how much variance there is between different training data, multiple subsets of the same size are used. Then, the mean and standard deviation can be calculated for each subset size. For small subset sizes up to 12.5 % of the total training data, 15 separate subsets are used. With increasing subset size, smaller subset sizes are used, as the difference between subset gets smaller due to overlapping. Above 50 % of the training data, only a single subset is evaluated.

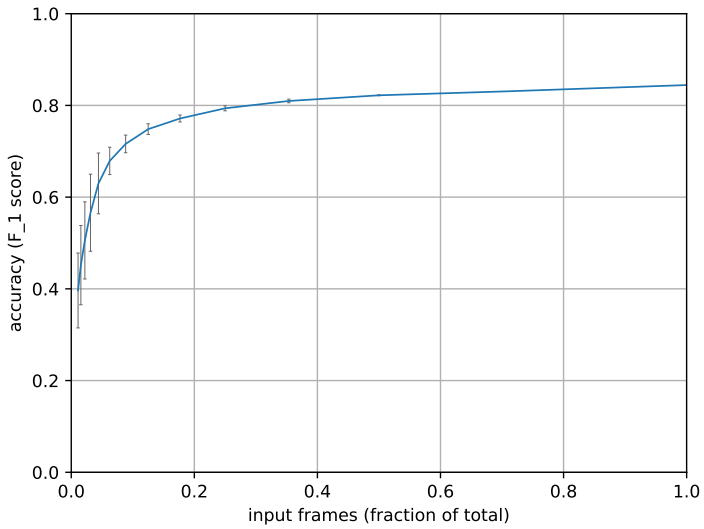


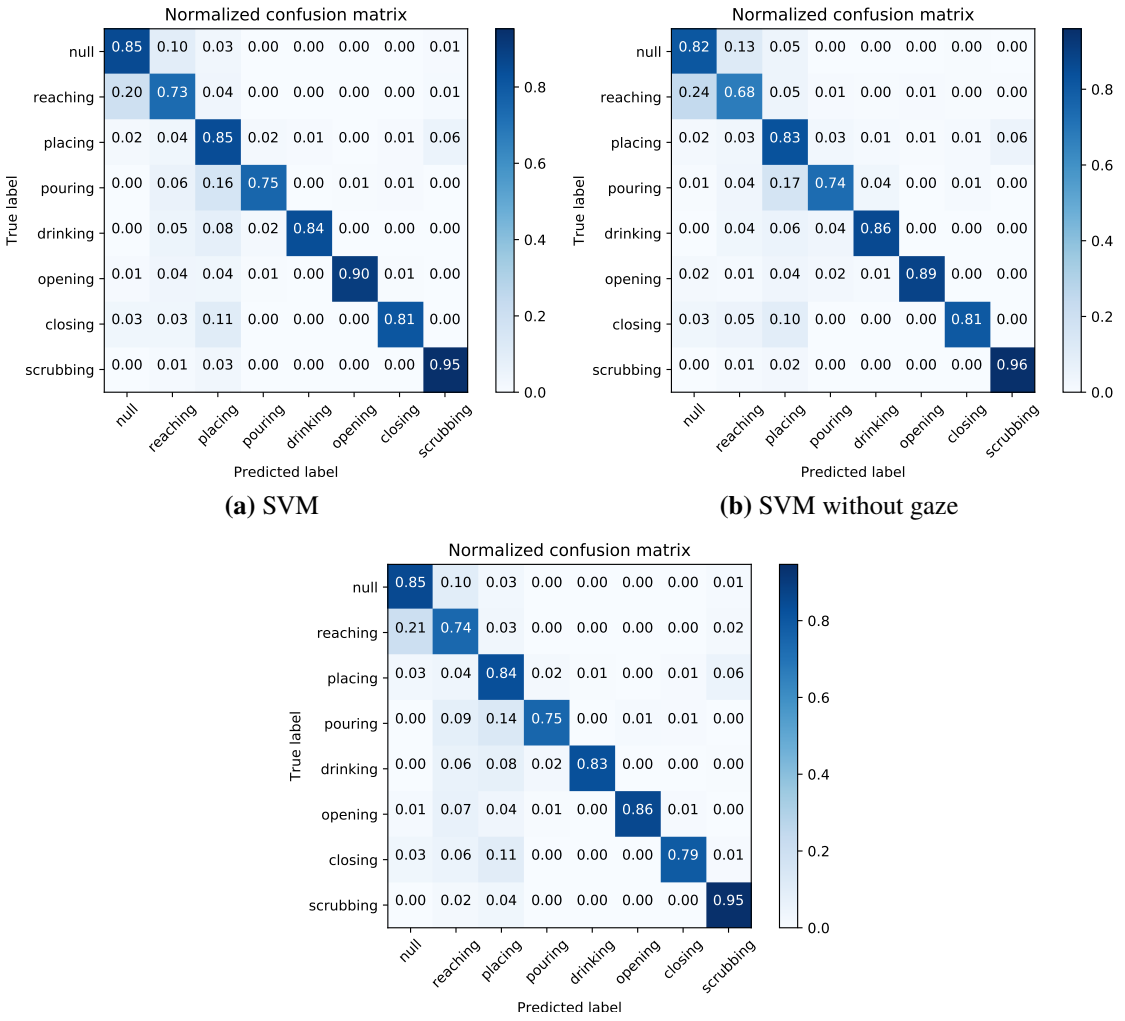
Figure 6.2: SVM learning curve generated by training the model in subsets of increasing size. The plot shows the F_1 score for each subset size and the standard deviation for multiple subsets of the same size.

The resulting learning curve is shown in Figure 6.2. The plot shows the expected learning curve with a steep increase at the beginning followed by convergence. It also shows that the variance between different training subsets decreases quickly.

Confusion matrices for the different SVM activity recognition models are shown in Figure 6.3. By looking at the detailed results for the standard SVM model, we can see that there are some cases where mislabeling occurs more often. One such case is labeling of *reaching* samples as *null*. It may be difficult to discern *reaching* from *null*, as it is just movement of the actor, which may also occur within *null* segments. In contrast to the other labels, no objects are near the actor until the end of the segment. In addition, the point in time where a *reaching* segment starts is not clear, as already mentioned in Section 3.3.

Another example for frequent mislabeling that is visible in the confusion matrix is prediction of *placing* instead of another class. All activities except of *null* and *reaching* involve the actor holding an object. Thus confusion within this set of activities is plausible. As the *placing* activity is the one with the largest number samples out of these, it is the most likely in the case that no other one fits and the actor is holding an object. In the case of *placing* instead of *pouring* or *drinking*, all of these activities start with the actor moving an object, where the only difference is the target location. The similarity may thus lead to confusions. Another special case is *placing* instead of *closing*. Here, the lid of the openable box is moved downwards during closing, which is a similar motion than placing an object. Also, as described in Section 3.3, there are some cases where the actor starts placing the cup after pouring while still closing the lid, so there may be confusion between different objects.

A different case is a prediction of *scrubbing* instead of *placing*. This may be because of the difficulty to identify the point when the actor stops to scrub and starts to place the scrubber. The scrubber is also near a plane while it is being scrubbed, so the difference between *scrubbing* and *placing* is only defined by movement and orientation.



(a) SVM

(b) SVM without gaze

(c) SVM OvR

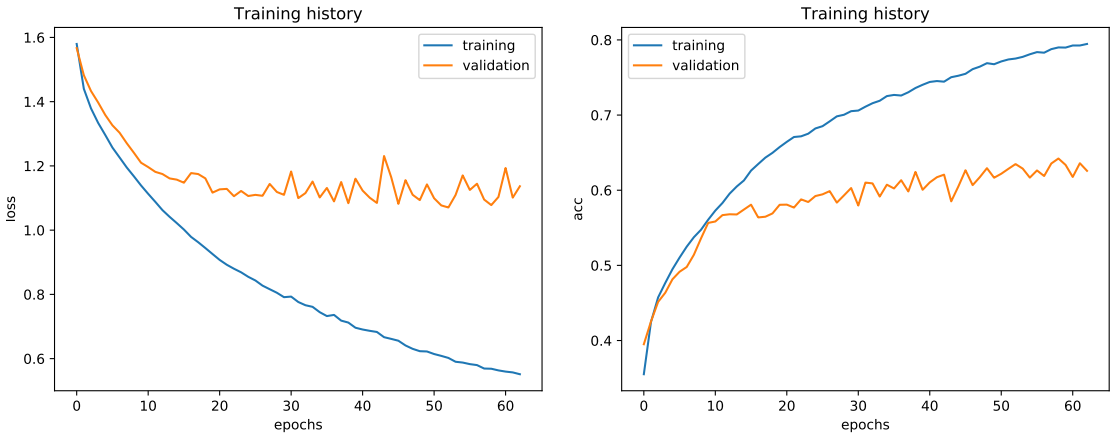
Figure 6.3: Activity recognition confusion matrices for three different SVM models, showing how each model labels samples with different labels.

When comparing the standard model and the OvR model, both have a overall macro F_1 score of 0.84. Looking at the per-label F_1 scores, only small differences are visible for some of the classes: For *pouring*, *closing*, and *scrubbing* the OvR model is slightly worse by 0.01. Thus, for activity recognition, the OvR model offers no advantage, but is practically equal in performance to the standard model.

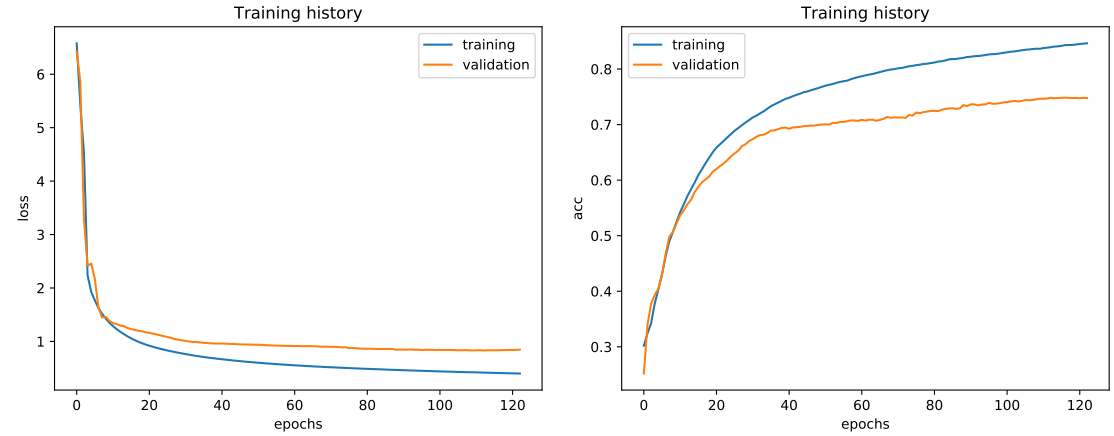
Comparing the standard model to the model without gaze features shows overall better performance for the standard model with a macro F_1 score of 0.84 compared to 0.81. In the per-class scores, the largest improvement is for the *pouring* label by 0.06. It is not immediately clear why adding gaze features improves results for this class. Possibly the additional gaze features help for cases where multiple objects are involved. This would fit with the increase of the F_1 scores for *opening* and *closing*, as both activities also involve two objects. The second largest change of 0.05 is for

reaching. This is plausible, as shifting of the actor's gaze was used as definition of the start point for *reaching* segments. The only label where the model with gaze data reduces performance is *scrubbing*, and even there it is only by 0.01.

Overall, our SVM model provides acceptable performance for human activity recognition. In addition, the results show that adding gaze information in addition to motion tracking data improves the performance of activity recognition.



(a) Simple LSTM model



(b) Structural LSTM model

Figure 6.4: Learning curves for different LSTM models. The progress of the loss is shown on the left and the accuracy on the right.

6.3.2 LSTM

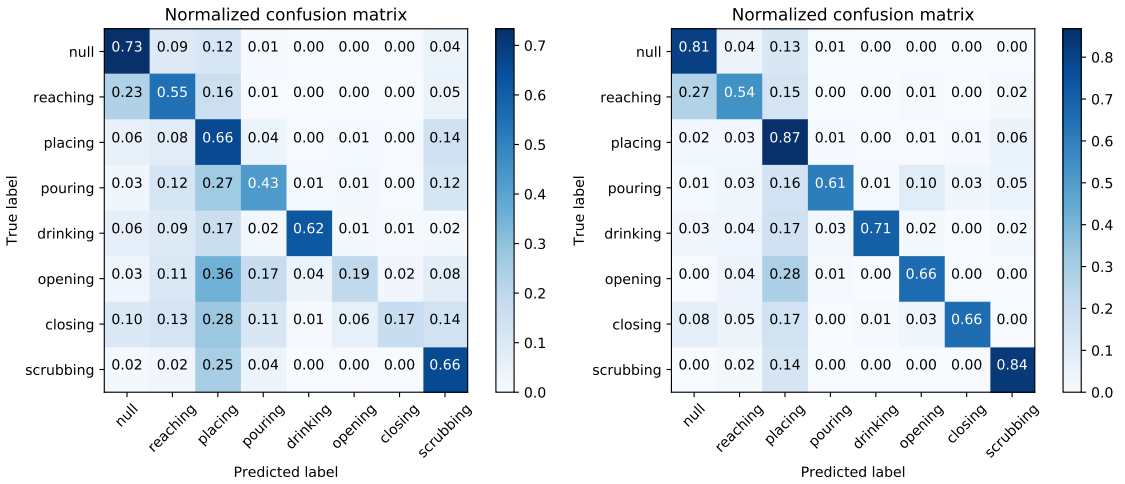
For the LSTM models, a learning curve is generated during training of each model. The training of the network is divided into epochs, which correspond to an pass over the entire input data. After every epoch, the current state of the model is used to evaluate both the training set and validation set. Two different metrics are computed, the value of the loss function, and the accuracy as the fraction of correctly labeled samples. This information is also used to interrupt the training early, if the validation loss no longer decreases within a succession of 10 epochs. By having history data on both the training set and validation set, it is possible to determine how well the model works, and whether there is overfitting or underfitting.

The learning curve of the simple LSTM model is shown in Figure 6.4a. The plot shows a quick decrease in training loss. However, at the same time, the validation loss and accuracy start to stagnate after about 15 epochs, which is a sign of overfitting. There is also some oscillation in the progress for the validation set. In general, the learning curve is already a sign that this model is not able to extract the required information from the input data to perform a good labeling.

In order to verify whether the simple model can work in principle, we apply it to a synthetic dataset. This dataset is based on 2D trajectories starting from a single point and moving forward in y-direction, with a randomized x-position as target. The classes are defined by 8 bins of the target x-coordinate. Only the first half of each trajectory is fed to the LSTM model as a time series. In order to get comparable results to the recorded activity data, we use a total of 75 000 trajectories and the same split ratio. The result shows a macro F_1 score of 0.96 and closely matching curves for validation set and training set in the learning curve. Thus, there is no general overfitting problem with the model, but only related to the input data.

In Figure 6.4b, the learning curve of the structural LSTM model is shown. The graph looks as expected, quickly improving after start, with slower improvement later on. However, the results of the structural model seem to depend heavily on random initialization for the training run. The presented results are actually the best of multiple runs. Often this model gets stuck at bad local minima. A closer look at the results shows that in these cases, some labels are missing entirely from the predictions, even on the training set. An extreme case of this is that every input gets the same label, with the results not changing at all in further epochs. Often there are also sudden jumps during training, where the loss and accuracy improve a lot. Experiments with different training parameters did not help to stabilize this model, the model also often gets stuck for smaller learning rates. We also recorded additional data with special focus on underrepresented labels, which did cause a small improvement in the results. Despite the issues of this model, we still present the prediction results for the best run.

Figure 6.5a shows the confusion matrix for the simple LSTM model. With a overall F_1 score of 0.52, this model has much lower performance compared to the existing SVM models. In general, it seems to have problems discerning activities where orientations and angles or relations between objects are required. For *opening* and *closing*, the F_1 scores are very bad with 0.27 and 0.26 respectively. Both activities require the angle, or possibly the distance, between the base and the lid of the openable box. The model seems unable to extract this information. The activity *pouring* also has a low score of 0.41. Similarly, it requires knowing the angle of the cup, and also involves multiple objects, namely the cup and a container. Other than that, many samples are mislabeled as



(a) Simple LSTM model **(b)** Structural LSTM model

Figure 6.5: Activity recognition confusion matrices for different LSTM models, showing how each model labels samples with different labels.

placing. The possible reasons are similar as described for the SVM results. Since *placing* has the largest number of samples from all activities involving an object, the model may tend to predict it instead of the correct label.

For the structural LSTM model, Figure 6.5b shows the confusion matrix. Compared to the simple model, it has better per-label F_1 scores for every activity and an improved overall macro F_1 score of 0.73. Especially for *opening* and *closing*, there are large improvements with scores of 0.65 and 0.72. This shows that splitting the model into separate parts, including object pairs, improves results. Specifically, relationships between objects can be detected from the object-object pairs. The differences of per-label scores between different labels have decreased compared to the simple model, and are now on par with the SVM models. However, there are still misclassifications of activities as *placing*, including of *null* and *reaching* which do not involve any objects.

Compared with the SVM models, the performance of the structural LSTM model is still significantly lower. In addition, the convergence in training is also an issue for this model. Nonetheless, the results are promising, considering that the model works without any hand-crafted features and uses only a combination of raw data. There are also still unexplored possibilities to improve the results. For example, the sizes of objects which are used in the SVM model for calculating features, are not used for the LSTM model. The gaze data is also just included on its own, without any combination with objects.

6.4 Activity Anticipation

We evaluate our activity anticipation implementation on the last 20 % of recorded data, which was also used as validation set for activity recognition. The data is sampled at one frame every second, yielding a total of 1492 samples. This is a bit less than a tenth of the sample count in the activity recognition validation set, as the last sample per recording session cannot be used due to missing

Model	Accuracy (any of 3)	Macro F1 score	Per-label F1 score							
			null	reaching	placing	pouring	drinking	opening	closing	scrubbing
SVM probabilities	0.62	0.37	0.37	0.42	0.39	0.35	0.13	0.19	0.37	0.76
SVM OvR	0.64	0.37	0.38	0.42	0.48	0.38	0.09	0.13	0.35	0.77
LSTM simple	0.59	0.31	0.47	0.26	0.40	0.27	0.08	0.04	0.23	0.73
LSTM structural	0.59	0.36	0.48	0.31	0.38	0.30	0.12	0.23	0.36	0.70

Table 6.2: Results of activity anticipation, comparing effect of different models for scoring.

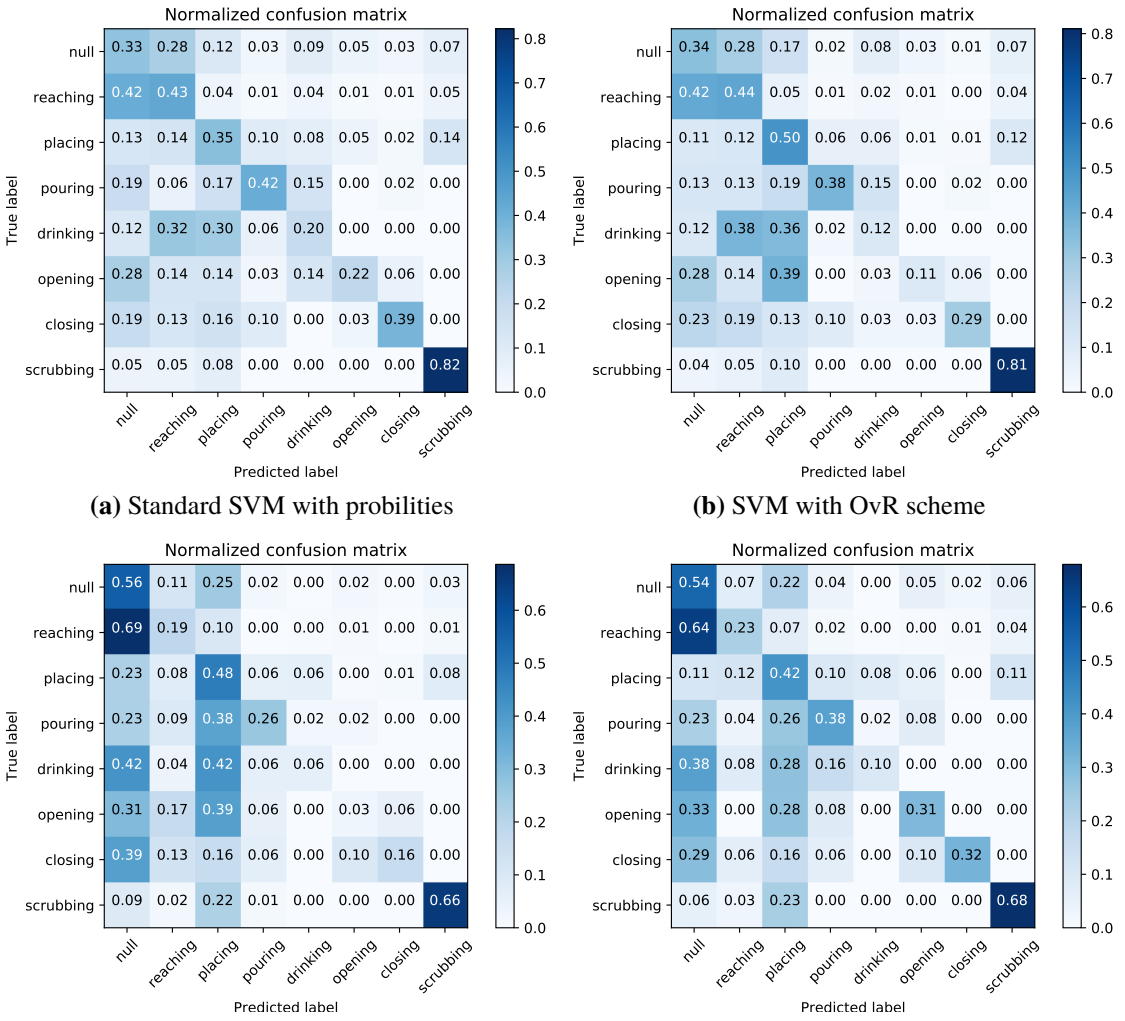
ground truth for the prediction. For each sampled frame, future trajectories are sampled and scored using the different models. The seed for the random number generator is derived from the recording time for each sample. This gives a consistent set of sampled trajectories, so that every model can be evaluated with the same data.

From the trajectories scored by an activity recognition model, the three highest-scored are used as output. To measure the performance, the labels of these trajectories are then compared to the true label one second in the future. For activity anticipation, we define the accuracy as the fraction of samples where at least one of the three labels matches the true label. In addition, we calculate the F_1 scores for the label of the top-rated trajectory.

Out of our activity recognition models, we compare the four models that provide the required scores. From the SVM model, these are the standard SVM model with additional probabilities, and the SVM model with OvR structure. In addition, both the simple LSTM model and the structural LSTM model are used for anticipation.

Table 6.2 shows an overview of the activity anticipation results. In general, the overall macro F_1 scores are very similar with a range of 0.36 to 0.37 for all models except the simple LSTM model at 0.31. The difference in accuracy is larger, with the best result being the SVM OvR model with a score of 0.64, followed by the SVM model with probabilities at 0.62. For the case of LSTM models, both have the same accuracy score of 0.59. More differences are visible by looking at the per-label behavior of the models and the confusion matrices in Figure 6.6.

For *scrubbing*, all models give good results with a F_1 score of at least 0.70. Actually, this label has the best F_1 score compared to the other labels for all of the models. An explanation for this is that *scrubbing* segments are usually much longer than a second. Thus when the actor is currently *scrubbing*, there is a high likelihood that they continue to scrub. In this case, a scrubbing trajectory will always be sampled, so if the model gives it a high score it has a good chance of being predicted. As shown in Section 6.3, all models are good at recognizing the scrubbing activity, so this seems likely.



(a) Standard SVM with probilities

(b) SVM with OvR scheme

(c) Simple LSTM model

(d) Structural LSTM model

Figure 6.6: Activity anticipation confusion matrices for different activity recognition models for scoring of trajectories, showing how each model affects the prediction results for different labels.

In constrast to *scrubbing*, all models show bad anticipation results for *drinking*. Except for the simple LSTM model, where *opening* is worse, it is always the activity with the lowest F_1 score. This contradicts with the activity recognition results, as all models are able to detect *drinking* acceptably. Possibly this indicates an issue with the generated trajectories, such as a non-natural movement which in turn causes low scores.

For the LSTM models, some issues that were already apparent for activity recognition appear again. Both models often mispredict activities as *placing*, which was also a problem for activity recognition. In addition, the *null* activity is often predicted instead of the true activity. Since the *null* trajectory is always generated by the sampling process, it can be predicted for every frame if the score is high enough. It is possible that the LSTM models produce a high score for the generated *null* trajectory, as there is no movement at all, which is easy to detect.

To summarize, the overall difference from using different models to score sampled trajectories is not very large. This suggests, that the hand-coded sampling algorithm already prevents very bad predictions by not sampling any trajectories that would be impossible under the current state. However, some differences between the model exist when looking at their behavior more closely. These differences are in line with the previous results from activity recognition. This shows that the principle of using activity recognition models to score trajectories works, and better activity recognition models can be used to also get better anticipation results. Overall, the SVM models provide the best result, which matches the results from activity recognition. There is a small overall difference between the two SVM models, but the variant with probabilities has slightly less difference in performance between different labels.

7 Discussion

In the preceding chapters, we presented our methods for recognition and anticipation of human activities. Our system is based on motion and gaze tracking of a human and objects in the environment. We implemented different variants for activity recognition. The first kind uses a SVM with hand-crafted features as input data. The other variants use LSTM-based neural networks and work directly on the recorded raw data in combination with object descriptions. For anticipation we sample future states and make use of the existing activity recognition models for scoring.

For training and evaluation of our models, we recorded a dataset and manually segmented and labeled it. An actor performs seven different activities involving eight objects in the scene. In addition to previous work, we also recorded eye gaze data. Gaze can be an indicator of future movements, as objects are typically fixated before an actual interaction is performed.

The evaluation of our activity recognition system showed good results for the SVM, with an overall F_1 score of 0.84. A comparison of variants with and without gaze-based features shows a positive impact from using gaze data. The model including gaze features improves the F_1 score by 0.03.

Our LSTM-based models did not reach the same performance as the SVM, with a F_1 score difference of 0.11 for the structural model. There are also issues in the training behavior of the structural LSTM model, as the result heavily depends on random initialization. The LSTM-based models also tend to mispredict activities as *placing*. Nonetheless, using an end-to-end LSTM-based model without having to calculate any features is still a promising approach.

The activity anticipation of our system turned out to be significantly less accurate than activity recognition, with a F_1 score of 0.37 for the highest-ranked trajectory. However, the results also show characteristics of the individual models which have been seen previously for activity recognition. This shows the influence of the used recognition model on the anticipation results, meaning that better recognition methods can be used to also improve the results of anticipation.

Future Work

There are still open possibilities for improving the training behavior and overall results of the structural LSTM model. One way could be to modify the design of the fusion layer, which currently uses summation and concatenation of the output from the LSTM layer. Another possible change could be to increase the number of LSTM layers. Currently, the structural model only uses a single layer whereas the simple model uses three.

Our anticipation scheme currently requires features, as those are used for sampling of trajectories. Thus, even with the LSTM model, it is not a full end-to-end solution. A possible alternative would be to build a generative model for trajectories using a learned model. As a side effect, this could also improve the quality of the returned trajectories to represent a more natural shape of movement.

For use of the system in a real world application, the current capture system is not practical, as it requires markers on the body and objects, as well as a headset. The motion tracking could be replaced by a system based on depth cameras. However, gaze tracking remains an issue, as it requires a clear view on the pupils of the tracked person.

A Appendix

A.1 Tracking Setup

For motion tracking, an OptiTrack[1] system together with the Motive:Body software is used. The OptiTrack system works with infrared cameras that at the tracked scene. Reflective markers in the shape of small spheres are placed in the scene and can be detected by the cameras. For this, each camera illuminates the scene with infrared light. We use a set of Flex 13 cameras that record images with a frame rate of 120 fps. The Motive software detects the position of the markers in each individual 2D image. By combining the images of all cameras, it calculates the actual 3D position in space for each marker.

A fixed arrangement of markers can be grouped as a rigid body. This can be used to track objects, where markers are placed in a way so that their orientation towards each other is fixed. Thus, not only the position of an object, but also its orientation can be determined.

In addition to rigid bodies, the version of the Motive software we use also supports human skeletons. The Motive software includes some skeleton templates which specify the positions on the human body where markers need to be placed. We use a template for the upper body that requires placing 25 markers. This template includes the upper body including arms and hands, as well as the head of a person. We do not track the movement of individual fingers. The resulting tracked skeleton is modeled by the body joints, which are represented by their position and orientation.

Our setup has a usable area of around 3×4 meters, where objects and humans can be tracked with high precision. The system consists of a total of 12 cameras. Of those, ten cameras are mounted on a truss system that spans the long sides of the area in a height of around 3 meters. The other two cameras are placed on tripods on the short sides of the area, at a height of circa 1.5 meters. All the cameras have been carefully aligned to ensure that the entire area is covered, and there is no interference between cameras.

For human gaze tracking, we use a Pupil[2] headset together with the Pupil Capture software. Our version of the headset contains cameras pointed at each eye, as well as a 3D world camera. For tracking of the gaze direction, only the eye cameras are used. The cameras include infrared illumination of the eye and capture images at a frequency of 200 fps. From the camera images, the Pupil software calculates for each eye the position of the pupil and the size and position of the eyeball. Before each use, the gaze tracking has to be calibrated by looking at target areas on a computer screen. In the end, the software provides the direction of the gaze point, at varying time intervals.

[1] https://www.optitrack.com/
[2] https://pupil-labs.com/

The coordinates of the gaze point provided by the tracking with the headset are relative from the location of the headset. In order to map the gaze direction into the world coordinate system of the motion tracking, we place five markers on the headset and set them up as a rigid body. This allows to later transform the gaze coordinates to the world coordinate system.

However, adding markers in proximity to the skeleton head markers can cause tracking errors, as the OptiTrack software confuses skeleton markers with rigid body markers. This can lead to false tracking of the head orientation, as well as the headset rigid body stopping to be tracked. We try to avoid these issues by placing the head and headset markers as far away as possible, but changes to the skeleton markers are limited due to the existing body template. It is possible to add more markers to individual body segments, so we manually add two more markers to the head in order to make the tracking more reliable. Thus, a total of 27 markers are used for the skeleton instead of the default number of 25 markers.

This is not the only way that interferences can be caused in motion or gaze tracking. Since both tracking systems use infrared light, the are sensitive to both external infared light sources and each other. By default, the OptiTrack system uses strobed infrared illumination in order to output more light at equivalent power levels. This can cause issues with the eye cameras. Thus, the OptiTrack system needs to be set to use continuous illumination.

Similarly, the OptiTrack system can get confused by infrared light originating from the headset. The eye illumination of the headset is required for proper working and cannot be disabled. However, the 3D camera also uses infrared light by projecting a light pattern into the environment. As we do not use the data from the 3D camera, we disable it after the calibration process is completed. It is also useful to make sure that no other similar devices are used in vincinity of the motion capture scene to reduce the likelihood of interference.

Another source of possible tracking issues are reflective surfaces within the scene. Bright spots in reflections can be detected as markers and thus cause false tracking results. In our case, the floor and the tables turned out to have high reflectance. To avoid these issues, we place a black carpet on the floor. We also reduce the light intensity to a level where reflections are minimized while markers are still clearly detectable.

A.2 Processing Architecture

We use the Robot Operating System (ROS) to develop our software. ROS is a framework that provides various functionality for robot software. The most important parts for us are the communication system and the RViz visualization tool. A program using ROS can consist of multiple nodes, which are in our case implemented as Python scripts. These nodes can communicate through a publish-subscribe system.

The RViz visualization tool is a node that can subscribe to topics where special marker messages are sent. These marker messages specify the geometry and 3D pose of objects, which are then visualized in an interactive 3D view. Markers are interpreted relative to a reference frame. Relationships between frames are defined by transformations, which can also be published via the communication system. RViz can also visualize these transformation vectors in the 3D view.

In our architecture, there are two nodes that are responsible for reading data from the tracking systems and publishing them to ROS. For recording the data, both nodes run on the same laptop. For timestamping, we use the local ROS time, although the motion tracking data already includes accurate timestamps. This way, we have the same time reference for both motion and gaze data, although a small amount of jitter is introduced compared to the original motion tracking timestamps.

The skeleton publisher node is responsible for motion tracking. The Motive software runs on a PC next to the tracked scene, where the OptiTrack cameras are connected. Motive publishes all marker, rigid body, and skeleton data via IP multicast for every captured frame (at 120 fps). This data is formatted using the OptiTrack NatNet protocol. OptiTrack provide a Python client for this protocol that we modified slightly for compatibility with the used version of the Motive software.

The gaze publisher node reads and publishes the data from the Pupil headset. The Pupil software runs on the same laptop as the publisher node, where the headset is also connected via USB. The node connects to Pupil Capture locally via the ZeroMQ publish-subscribe system, and publishes the received data to ROS.

Motion data is published in the form RViz markers for visualization, and additionally in a custom message format that contains the raw data for further processing in other nodes. In addition, the 3D pose of the gaze tracking headset is also published as a transformation, so that the actual gaze direction can use it as a reference frame. The gaze direction is published as a transformation from the headset to a point at large distance. This transformation is visualized in RViz and also contains all required data for further processing of the gaze information.

At the same time as data is published to ROS, the nodes also save it into HDF5[3] files. HDF5 is a file format for structured data, that allows to efficiently store multi-dimensional tables. We use it via the h5py[4] Python package. The stored data can be played back via a playback node, which publishes it in the same way as the two publisher nodes. This allows our architecture to work the same way on live data as on recorded data.

A.3 Labeling Tool

In order to label the recorded data, we developed a custom graphical tool using Python and the Qt5 UI framework. It is implemented as a ROS node that publishes data for visualization in RViz and at the same time provides an interface to label the data. In addition to the graphical user interface, faster input is possible using the keyboard. The interface is shown in Figure A.1.

Before labeling the data, it is first split into segments of varying length. When a new recording session is opened, it starts with a single segment which can be modified by split and merge operations. Each segment can then be assigned a different label. The segments and labels are stored in a HDF5 file for each recording session.

[3] https://www.hdfgroup.org/
[4] https://www.h5py.org/

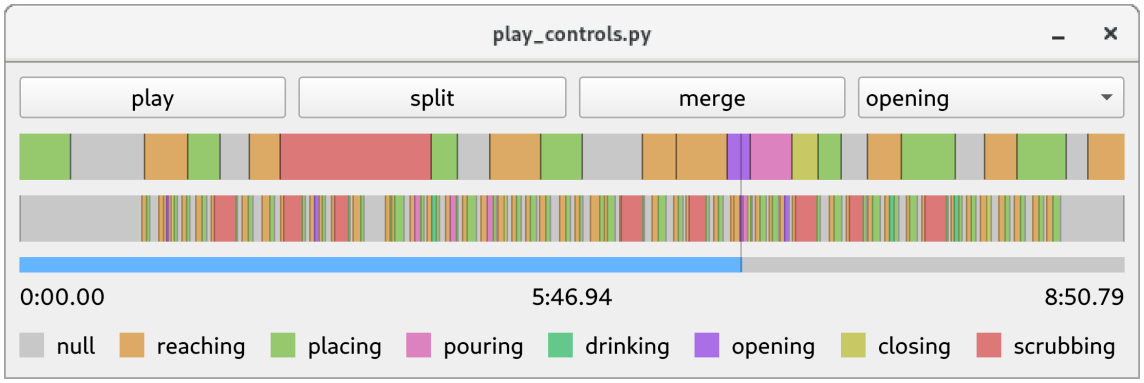


Figure A.1: Playback and labeling tool shown for the shortest of our recording sessions. An overview of the labeled segments with color coding is shown. Individual segments can be edited within the graphical interface or using keyboard controls.

A bar graph visualizes the segments in the file separated by lines, and also shows the current position within the recording. The label of each segment is represented by a unique color which corresponds to the label. Below the seek bar the segments in a time window of 60 seconds around the current position are shown in more detail. This is useful, as most of the recording sessions have a duration greater than 30 minutes, so the individual segments cannot be distinguished in the overview.

The user can navigate within the recording by using the segment overview as a seek bar, or by seeking at varying speed using the keyboard. In addition, it is possible to play the recording at normal speed.

Bibliography

[AAB+15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/ (cit. on p. 30).

[AS17] H. Admoni, B. Scassellati. "Social eye gaze in human-robot interaction: a review". In: *Journal of Human-Robot Interaction* 6.1 (2017), pp. 25–63. DOI: 10.5898/JHRI.6.1.Admoni (cit. on p. 15).

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, 2006. ISBN: 978-0-387-31073-2 (cit. on p. 16).

[Cho+15] F. Chollet et al. *Keras*. 2015. URL: https://keras.io/ (cit. on p. 30).

[DTC+18] N. Duarte, J. Tasevski, M. Coco, M. Raković, J. Santos-Victor. "Action Anticipation: Reading the Intentions of Humans and Robots". In: *arXiv preprint arXiv:1802.02788* (2018). URL: http://arxiv.org/abs/1802.02788 (cit. on p. 15).

[GSK+17] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, J. Schmidhuber. "LSTM: A search space odyssey". In: *IEEE transactions on neural networks and learning systems* 28.10 (2017), pp. 2222–2232. DOI: 10.1109/TNNLS.2016.2582924 (cit. on p. 17).

[GSS02] F. A. Gers, N. N. Schraudolph, J. Schmidhuber. "Learning precise timing with LSTM recurrent networks". In: *Journal of machine learning research* 3.Aug (2002), pp. 115–143 (cit. on p. 17).

[HM16] C.-M. Huang, B. Mutlu. "Anticipatory robot control for efficient human-robot collaboration". In: *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press. 2016, pp. 83–90. DOI: 10.1109/HRI.2016.7451737 (cit. on p. 15).

[HS97] S. Hochreiter, J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. on p. 17).

[JSK+16] A. Jain, A. Singh, H. S. Koppula, S. Soh, A. Saxena. "Recurrent neural networks for driver activity anticipation via sensory-fusion architecture". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 3118–3125. DOI: 10.1109/ICRA.2016.7487478 (cit. on p. 14).

[JZSS16] A. Jain, A. R. Zamir, S. Savarese, A. Saxena. “Structural-RNN: Deep learning on spatio-temporal graphs”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5308–5317. DOI: 10.1109/CVPR.2016.573 (cit. on p. 14).

[KGS13] H. S. Koppula, R. Gupta, A. Saxena. “Learning human activities and object affordances from rgb-d videos”. In: *The International Journal of Robotics Research* 32.8 (2013), pp. 951–970. DOI: 10.1177/0278364913478446 (cit. on pp. 11, 13, 22).

[KS13] H. S. Koppula, A. Saxena. “Anticipating Human Activities using Object Affordances for Reactive Robotic Response”. In: *Robotics: Science and Systems (RSS)*. 2013 (cit. on pp. 13, 14, 31).

[KS16] H. S. Koppula, A. Saxena. “Anticipating human activities using object affordances for reactive robotic response”. In: *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2016), pp. 14–29. DOI: 10.1109/TPAMI.2015.2430335 (cit. on pp. 13, 31).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, É. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (Nov. 2011), pp. 2825–2830. URL: http://scikit-learn.org/ (cit. on p. 28).

[Sch15] J. Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003 (cit. on pp. 16, 17).

[SGR05] A. P. Shon, K. Grochow, R. P. Rao. “Robotic imitation from human motion capture using gaussian processes”. In: *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*. IEEE. 2005, pp. 129–134. DOI: 10.1109/ICHR.2005.1573557 (cit. on p. 15).

[SLNW16] A. Shahroudy, J. Liu, T.-T. Ng, G. Wang. “NTU RGB+ D: A Large Scale Dataset for 3D Human Activity Analysis”. In: *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. IEEE. 2016, pp. 1010–1019. DOI: 10.1109/CVPR.2016.115 (cit. on p. 14).

[TY17] A. M. Truong, A. Yoshitaka. “Structured LSTM for human-object interaction detection and anticipation”. In: *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on*. IEEE. 2017, pp. 1–6. DOI: 10.1109/AVSS.2017.8078543 (cit. on p. 14).

All links were last followed on June 19, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature