Master Thesis

Bielefeld University

Faculty of Technology

Memory-based exploratory online learning of simple object manipulation

In Partial Fulfillment of the Requirements for the Degree of

Master of Science (M.Sc.)

Jan Pöppel

September 2015

Referee: Apl. Prof. Dr.-Ing. Stefan Kopp

Referee: Dipl.-Inf. Maximilian Panzner

Abstract

We also translate the abstract to English.

Contents

Chapter 1

Introduction

Robotic research is both fueled by the desire to better understand the human cognition as well as the desire to create tools that improve our everyday life. Starting from industry machines that have become more and more capable in their abilities to perform their designated tasks autonomously, robots and virtual agents are becoming more popular in our everyday life. Lawn mower or vacuum cleaning robots are just two inventions that have become increasingly popular in recent years [**?**]. In addition to such specialized machines, researches are trying to develop general purpose robots that are capable of performing a multitude of tasks, including object manipulation and interaction with other agents. Although a complete system meeting such standards has not been developed yet, the researches around the world are making progress in distinct subproblems such as object recognition [**?**] or trajectory learning [**?**].

Currently, robots are mostly trained to perform one specific set of tasks offline with previously recorded data [**?**]. In most cases the performance improves with the amount of data available for training. However, there are two major problems with the current approach. Firstly, especially in robotics, gathering data is usually very expensive. A lot research is being performed to try to reduce the required amount of data (e.g [**?**]). Secondly, such an approach limits the system to the abilities that it learned from the presented training data. Although modern learning approaches can show good generalization performances [**?**], there is currently no learning approach that can for example learn to open a bottle from training data that was recorded moving blocks. Acquiring training data for all possible tasks the robot is supposed to perform during its lifetime is infeasible at best. For that reason, online learning has become very important in recent robotics research [**?**]. When performing online learning, especially when it is performed life-long, i.e. during the entire lifetime of the robot, additional challenges arise, such as meeting performance criteria and the stability-plasticity-dilemma [**?**]. Modern robot architecture consist of a multitude of

separate components. Online learning can be used in different parts of a complete architecture, potentially even improving the performance of other components since they can adapt to changes dynamically [**?**].

Object manipulation is a very important task for many robots since it allows them to interact with their environment. Human children are capable of learning simple object manipulation at a very young age with only a relatively limited amount of training. Furthermore, humans are able to generalize what they have learned to quickly acquire the knowledge about how to manipulate never before seen objects. Understanding how the human brain achieves these learning performances as well as enabling this tasks to robots has become the goal for many researchers [**?**].

This thesis presents an online learning model to learn simple object interaction through self-driven exploration.

TODO: Mention

- reasons/motivation for Memory-based approach (one-shot learning), check with concept section about memory based models
- Tasks: Prediction/Forward model and planning/inverse model

Chapter 2

State of the art

Chapter 3

Concept

In order to achieve the desired results, multiple models were developed incrementally. This chapter explains the different concepts in general. Details on the concrete realizations of these models can be found in chapter 4 while their evaluation and discussion can be found in chapters 6 and 7 respectively. The first concept, explained in section 3.2, uses a pairwise interaction representation. In contrast to that, section 3.3 describes the second concept, which represents the individual objects and models the change in their dynamics.

3.1. Memory based models

TODO? Vl besser bei technologies?

The general idea of memory based learning, also called instance based learning, is to use lazy generalization. This means that the system compares a new problem instance with previously seen training examples in order to make a prediction instead of using an explicit generalization that was computed during training.

The simplest memory based approach is to simply store the training data as input-output pairs in memory. When querying the model, for example in order to make a prediction, a nearest neighbor search [**?**] is performed on the inputs for all seen training examples and the corresponding output of the most similar pair is return as the prediction.

The advantages of such an approach are that there is no catastrophic forgetting during online learning, since the model only changes locally around each newly inserted data point. Furthermore, the required amount of training data is very limited as long as it is very similar to the test situation. In fact, depending on the query task, a single trained input-output pair can be enough to provide a reasonable prediction.

Figure 3.1.: TODO Visualization of pairwise concept

The disadvantages are increasing query times as the model grows since most of the work is performed then. On top of that, the simple model does not generalize well in between training points if there are big changes. (TODO show graph to visualize this)

Multiple approaches have been proposed in order to reduce the gravity of the disadvantages such as using index structures to increase nearest neighbor search or using different methods in order to smooth the model output, for example by interpolating between the closest k points.

A memory based approach was mainly chosen due to the good online training capabilities and the one shot learning properties for this thesis.

The different concepts presented here try to structure and cluster the training data in order to guide training and improve prediction.

3.2. Pairwise interactions

The first developed model adopted the commonly used idea of modeling object interaction by concentrating on pairwise interactions, also used by [**?**]. The idea is to not model each object for itself but rather model the interaction between them by using features that describe their relative attributes (see section 4.2.1 for details about the chosen representation). When predicting the next object state, the next interaction state is predicted first and the resulting object states are extracted from there afterwards. The input for model consists of the current interaction features between two objects and the chosen action. The output represents the change from the input in the next state.

In order to reduce the query costs the model tries to cluster similar interactions in more abstract collections, that are called “abstract episode collections” here. The general idea is visualized in figure 3.1.

3.2.1. Abstract episode collections

In order to avoid the nearest neighbor search on all recorded episodes, this model tries to cluster the experiences it made into more abstract collections. The clustering is performed based on the set of attributes that changed within each episode. The set S of attributes that changed is defined as follows

$$S = \{f | f \in F \ \wedge \ ||Pre(f) - Post(f)|| > \epsilon_{Noise}\} \tag{3.1}$$

where F denotes the set of all features, $Pre(f), Post(f)$, the value of feature f in the initial and resulting state of the episode respectively and ϵ_{Noise} a threshold to cancel out small noise.

For each different set the model creates a new abstract collection AC_i. The idea is that these collections correspond to different interaction scenarios. The set $\{spos\}$ for example corresponds to the interaction of pushing the object directly in the center, since the distance and relative velocities do not change between the two objects. The set $\{dpos, dist\}$ on the other hand describes the interaction of the actuator moving without contact to the object. The abstract collection effectively split the interaction space in subregions.

3.2.2. Prediction

Prediction is performed by following a winner-takes-all approach: When predicting, the model first estimates the collection that is most likely responsible for the next episode and uses the collection to make the prediction.

By training a classifier, e.g. a decision tree [**?**], on all the input-collection pairs, that were already seen, this selection becomes independent of the number of training examples already stored. The total number of collections is limited to the size of the superset of F, although in practice, not all possibilities are likely and only those collections are considered, that are already made up of more than ϵ_{min} training examples. This threshold is used to reduce the number of outliers when training the classifier.

After the suitable collection has been selected, one could perform nearest neighbor search on the episodes of that collection in order to make the prediction. However, since the number of episodes can grow quite rapidly, each collection trains it's own local predictor. This also allows the model to remove stored episodes when the local predictors are already trained sufficiently well.

In order to reduce the burden on the manual feature and metric selection, an automatic feature selection can be performed periodically in each abstract collection in order to improve the performance of the local predictors.

3.2.3. Planning

(TODO consider redoing the planning algorithm more similar to the way it is done in the most recent model, which works, maybe the same ideas can be employed here)

In order to get a suitable action given a target configuration, the model first computes the difference set between the current situation and the target configuration. The model then

searches for an abstract episode collection that corresponds to the same set of attributes and delays the action computation to that collection. In case the model does not yet now of such an collection, the most similar collection is chosen instead, meaning the collection that covers most of the changed attributes in the computed difference set.

The abstract collection queries it's own inverse model, which can be the same as their predictor, if that model supports bidirectional queries, for the action that produces the output most similar to the desired target.

[TODO FORMEL]

The model can then use it's forward model in order to check if the selected action is applicable in the current situation and if it indeed reduces the distance to the target. In case no suitable action could be selected, the model can choose to perform a random action, in order to experience new interactions, that will improve the following predictions and planning results.

3.2.4. Theoretical problems

The biggest problem with the pairwise interaction based approach is that it does not model the interactions between multiple objects. In a scenario with two objects alongside the actuator, it is not trivial to decide which pairwise interaction should be used to predict the next state of each of the objects, even if only two entities interact. In interaction scenarios where all three objects are touching at least one of the other objects, the pairwise representation fails completely since it can only represent effects between two of them at any given time.

- AC selection
- Number of samples increases a lot (even without interaction/touch, the interaction states differ -> one AC gets trained)
  - The most simple AC (no interaction, just actuator movement) becomes the slowest

3.3. Object states and global changes

(TODO Maybe actually drop this concept or briefly mention it as intermediate step to the current, best working model?)

Figure 3.2.: TODO Visualization of object states + global change

In order to avoid the problem of multiple objects, a second concept was developed. Instead of pairwise interactions the objects are modeled individually by object states (see section 4.3.1 for details about the used representation and features). Predictions are made solely based on "local" features for each object. Local means here, that an object predicts e.g. it's next position only based on it's current features, such as position and velocity. In order to be able to predict the effects of interactions such as collisions, another layer of predictors is introduced that predict the "global" changes on the dynamics of each object. Here global means, that other objects are used to compute the used features.

Overall the main difference is that instead of predicting the next state directly, by predicting the next interaction state, first only the dynamic features are predicted and later based on that, the remaining features are computed. This concept is visualized in figure 3.2.

3.3.1. Local prediction

As mentioned above, instead of predicting an interaction and extracting the object states, the object states are predicted directly based on their "local" features. In order to avoid strong differences between different object types, one predictor is trained for each type from the corresponding episodes.

3.3.2. Global change prediction

During an interaction, such as a collision, the local attributes of each involved object change. Since these changes depend on the "global" properties of the objects relative to each other, they are called global change here. In order to be able to predict these changes, these relative attributes need to be considered. This model uses pairwise interactions again, similar to the previous concept. Since the actual object states do not need to be recovered from these interaction states different features can be used.

One can now train one predictor that predicts the changes to the local features of each object based on a given interaction vector. This predictor would need to be able to generalize over the entire interaction space and over all objects. Alternatively, one can train one of these predictors for each object.

Another approach yet again, is to employ a similar strategy as before: By splitting the interaction space into subregions and training a local predictor for each region, these local

predictors can be simpler and more specialized. This however, entails the problem of actually splitting the interaction space....

3.3.3. Prediction

In order to predict the next object states, the model first considers each object pair and checks for global changes based on these interaction states. Selecting the appropriate change is done similar to the abstract collection section mentioned in the previous model. A classifier is trained using the interaction state and change pairs that were experienced before. That way an object's attributes can be changed by multiple interactions which allows also interaction scenarios with more than two objects involved. Afterwards each object predicts it's next state based on it's local, potentially modified, features.

(There is a serious flaw in this argumentation, that kind of negates the purpose of this concept. By still using only interaction states, the changes might come from multiple objects but it makes a big difference if only one objects pushes one other, or multiple...)

3.3.4. Planning

(TODO if at all, this has not been tried yet with this concept. Either make this one to the current one or omit?)

3.3.5. Theoretical problems

- Approach is not really correct for multiple objects since dynamics change differently depending on if there is only one object involved or multiple
- Action selection is not trivial
- Action separation (gold standard) is quite hard to do.
- Predicting changes is not really suitable, since dynamics also change from untracked interactions (friction)

Figure 3.3.: TODO Visualization of object state + gate

3.4. Object states with gate function

[TODO only rough draft]

Both previous models separated the space of possible interactions into different cluster. The first concept by introducing the abstract collections and the second one by (potentially) introducing different change predictors. In both cases, is it necessary to train the model on training data from all possible situations, including those, were only the actuator moves without interacting with any object. This does not only reduce the performance of the model since there are more references to search through when making predictions, but it can also hinder the selection classifier, since it will usually see a lot more training examples for this "no interaction" scenario. The evaluations (see chapter 6 for details) show that the accuracy of the predictions greatly depend on the accuracy of the selection mechanism.

In order to avoid this problem, a "gate function" is added in this concept. This gate function is used to determine for every object besides the actuator if in a given situation a selected action will have an effect on that object. If the gate function predicts that there will be an effect, the change is predicted by a predictor.

This again creates two layers for prediction, as the other concepts did (compare figure 3.3 with 3.1 and 3.2). The main difference here is however, that this gate function can be a lot simpler than the AC selection since it is only a binary classifier. In fact, this gate function can even be provided to the model for specific domains, where such knowledge is available since it does not depend on internal structuring as the AC selection does.

The predictor predicts the objects directly as in the second concept but not only on local features, but uses features from the interaction of the actuator and the object that is predicted to change.

The gate function can also be used between objects in case multiple objects are present in the scene. In that case interactions between objects that do not involve the actuator can be handled as well.

(STILL TO TRY) In order to predict multi object interactions, the gate function can be used to first collect all involved objects before invoking the appropriate predictor. (e.g. by interpolating the different input features).

3.4.1. Planning

Planning is performed fairly similar to the idea mentioned in the pairwise concept: First a check if the target has already been reached is made. In case the target has not been reached the predictor is ask to provide the required conditions that need to be met in order to move the target objects towards the target configuration. These conditions are provided by the inverse model of the predictor. In this case however, a special inverse model is used instead of using the inverse model of a bidirectional regression model. Searching for specific target changes in the trained regression model has several disadvantages: Firstly, the distance to the target configuration can potentially be a lot greater than any changes the model as seen during training. In fact this will be the case in most cases when the can not reach the target by executing a single action. This would require the inverse model to generalize greatly to unknown regions. Secondly, the target configuration can consist of different attributes, such as position and orientation. Depending on the used regression model, the accuracy of the model might depend on the quality of a metric combining such different domains.

Therefore, instead of relying on the quantity of the target change, only the direction of each attribute is considered in this special inverse model. For each output change direction, the inverse model trains a prototype during training. This prototype computes the average preconditions for its given change. This means that there might be a prototype responsible for a positive change in the local x position of the object. This prototype will collect preconditions for this change, that the model has experienced during training, in order to construct an average precondition that produces such a change.

When asked for preconditions in order to reach a certain target configuration, the inverse model uses a greedy strategy to provide the preconditions that reduces the attribute with the biggest difference.

The returned preconditions are then analyzed if they can be executed given the current situation. If they can be executed, the contained actuator action simply returned. Otherwise, the model tries to find an action that moves the actuator to a configuration closer to the given preconditions without interacting with the object on the way.

Chapter 4

Model realization

This chapter explains how the different concepts were implemented. Starting with some commonalities that are shared by the different model implementations in section 4.1, the pairwise interaction model is described in 4.2 and...

4.1. Commonalities

TODO: REWRITE to fit into this chapter!!! The concepts presented here share some common aspects. Firstly, these models work on single timestep intervals or episodes which is further explained in section 4.1.1. Furthermore, the actuator is treated just as any other object which is detailed in section 4.1.2.

4.1.1. Single timestep prediction

The developed models all share the same idea of working with single timesteps. That means that instead of predicting the final result of an interaction, the models make small predictions at each timestep. The reason for this decision is mainly due to the fact, that different interactions take varying amounts of time. While it is possible to train prediction models, that can reproduce trajectories with varying speeds, such as a Hidden Markov Model [**?**], it is very hard to define the endpoint of an interaction automatically. Since these models are supposed to learn incrementally by self exploration or at least in an online manner without being provided labeled data from the outside, the models would need to derive the start and endpoints of an interaction themselves.

By using single timestep predictions, the models do not need such a segmentation. This approach does however assume that the interactions are first order Markov processes, i.e.

the next state only depends on the current state. This assumption is justified, as long as the state contains dynamic features.

(TODO Nice text but does not make sense here-> move elsewhere but where?) By storing enough of such cases, prediction as well as one step planning can be reduced to a nearest neighbor search. The problems with this approach for the given setting of incremental learning of interaction prediction are the following: Firstly, the given feature space poses some problems. Nearest neighbor search heavily relies on the used metric and finding an appropriate distance measure for the given features or constructing a representation suitable for a given metric is a research challenge in itself. Secondly, even more problematic is the amount of episodes that are required to cover most of the interaction space sufficiently well. Since all the stored episodes need to be compared when searching for the most suitable one, performance deteriorates over time when more and more episodes are recorded in order to cover the entire space. In order for the system to be able to operate in real time, improvements need to be made.

4.1.2. Actuator as object

For all the concepts presented here, the actuator that the model can control via actions, is regarded just like any other object in the scene. The reasons for this is that these concepts do not have a specific control model for the actuator, i.e. they initially do not know the effects of the actions they can do. In this sense, the concepts try to estimate a forward model of their own actions and try to learn about interaction between objects at the same time.

4.2. Pairwise interactions

As mentioned in section 3.2 the pairwise interaction model considers the interactions between object pairs directly and derives the actual object properties from these interactions. That means that the objects attributes that are published by the simulation, are first transformed into interaction states, which are further explained in section 4.2.1.

Each timestep the model is provided with the current new interaction states for all object pairs. Furthermore, the model knows about the last state, the last action and it's last prediction in case it has performed one. This information is used to update the model, which is further explained in section 4.2.2.

In order to predict the next state, the current interaction state as well as the current action are used to first select the most appropriate abstract collection (see section 4.2.3 for details about how the selection is performed). This basically already determines what attributes

will change by performing the selected action. Afterwards, the selected collection uses the same input to predict how much the attributes actually change by querying the trained predictor.

4.2.1. States and Features

Short version: sid, oid, spos, sori, slinVel, sangVel, dist, oid, dpos, dori, dlinVel, dangVel Attributes relative to the block object were used. By doing this, the model was invariant to different rotations in the world.

4.2.2. Model update

Updating the model for predictions means to update the predictors for the abstract collections. In order to do so, the difference set from the last and current state is computed and the corresponding abstract collection (AC) is selected, if it already exists. If a new combination of changed attributes is discovered, a new AC object is created and added to the model's list of ACs. The last state, the last action and the resulting current state are then passed to the responsible AC as reference where it can be used to improve the predictor. The model itself uses the last state and the last action in order to improve the AC selection.

The abstract collections can use any suitable regression model for their predictors, in this case Instantaneous Topological Maps with additional output (ITM see section 5.2.2 for details) were used. The last state and the last action are used as input while the differences to the resulting interaction state were used as outputs. One can train a single ITM for all outputs together, however, in order to reduce the dimensionality an individual ITM was trained for each changing output feature. So if an AC is responsible for changes in spos and dist, the AC trains two ITMs on the received training data.

For the AC selection a decision tree is used as already mentioned when describing the concept. This decision tree is trained with the last interaction state and last action as input and the id of the AC that was actually responsible for the occurring change. So far this training is a batch training, using the input vectors of all previously seen references that are stored in each AC, since the training is quick enough. This should however be changed to an incremental update to that the references do not need to be stored indefinitely.

4.2.3. Abstract collection selection

The naive approach of selection the most probable responsible AC is to compare the current interaction state and the current action to all the references of all ACs. The most probable AC is then the one that contained to most similar reference. This however negates the purpose of abstract collections since it requires to test against all previously seen examples and by doing so one could simple use the result of the reference instead of the prediction of the abstract case. Although this approach is unfeasible due to the computational costs, it can be used when the classifier has not been trained sufficiently well. Furthermore, it can be used as baseline to evaluate the classifier.

The normal way however, is to use a trained classifier, in this thesis a decision tree. Using the current interaction state and the current action as input, this classifier returns the id of the abstract collection that should be responsible for the resulting change.

4.3. Object state

TODO

4.3.1. States and Features

Short version: The object states are represented by following features: id, posX, posY, ori, linVelX, linVelY, angVel For the global predictions, features that represent the interactions are needed. Here pairwise interactions are used: sid, oid, dist, closing, relPosX, relPosY, relVelX, relVelY (Still subject to a lot of change)

Chapter 5

Environment

This chapter describes the software environment (section 5.1) the different models were developed and tested in. Furthermore section 5.2 describes the important technologies that were used.

5.1. Simulation

The model is supposed to learn object manipulation through interaction. Instead of on a real robot the model has been developed and tested in a simplified simulation. For this work gazebo [KH] V.2.2.3 was used with the physics engine Open Dynamics Engine (ODE) [S+05]. This simulation and version were chosen because this thesis was initially planned as a part of a bigger project in the Citec of Bielefeld University where this software was used as well.

5.1.1. World and robot actions

The model was developed with a toy-world consisting of only a sphere, which is regarded as actuator, and a rectangular block as can be seen in figure **??**. Since the focus of this work is to learn simple object interactions, the possible actions of the robot were limited to moving the actuator freely in two dimensions.

Figure 5.1.: Overview of the used world and objects. The sphere represents the robot's actuator and the rectangular block represents the primary object.

Command	Meaning
Move Command	Sets the velocity for the actuator
Pause	Pauses the simulation
Unpause	Continues the simulation
Reset	Resets the world to starting configuration
Set Pose	Places a specified object at a certain position

Table 5.1.: Overview of all the commands implemented in the model.

5.1.2. Communication

In order to communicate with the simulation, a plug-in was written, that publishes the properties of all objects using google protobuf [**?**]. These messages are received and unpacked by a custom python interface that handles the communication with the simulation from the model's side. The plug-in also interprets the commands coming from the interface. A list of all implemented commands can be found in table 5.1

5.2. Technologies

This section briefly describes the technologies that were used or adapted to build the model.

5.2.1. Case based Reasoning

Case based reasoning (CBR) is a delayed... In CBR a case, in this case an episode, consists of an initial state, an optional action or event, and the resulting state after the action or event has taken place. In CBR these cases are simply stored. Whenever a new situation is to be handled, the case whose initial state and action are most similar to the current situation and available actions is retrieved and the resulting state is used to make a prediction about the effect of each action. Inversely, by searching for the resulting states, one can also decide what kind of actions should be taken in order to achieve a certain desired state.

5.2.2. Instantaneous Topological Map

Talk about :

- Similarity/difference to GNG
- Extension to provide output
- How is it trained
- How does it make predictions
- How does it return an action/preconditions

5.2.3. Decision Tree

5.2.4. Learning Vector Quantization

Chapter 6

Evaluation

This chapter describes the different evaluation scenarios that were used to test the trained model as well as present the achieved results in each of these scenarios. The presented tests were designed to measure the forwards models as well as the inverse models performances. The first scenario, called the "Push Task Simulation", tests the forward models of the different concept, while the second test scenario, the "Move to target", tests the inverse models.

6.1. Push Task Simulation

This task is designed to test the accuracy of the forward models by decoupling them from the open world during successive predictions.

6.1.1. Scenario description

In this test, the actuator uses a constant action in order to first move towards the block and then push it. The actuator starts at different starting positions on a line parallel to the block so that the distance along the action axis always starts at 25cm between the two centers. The block is orientated so that it's main axis is perpendicular to the action axis.

The model is first trained using the open loop. This means that the model receives all information about the world at each timestep before making the prediction for the next timestep. This training is done for a set amount of "runs". A run starts when the actuator performs the first action from the starting position and ends after a fixed number of iterations or if the actuator travels a predefined distance. An example starting and end configuration can be seen in figure 6.1.

Figure 6.1.: TODO Push Task Simulation. The left side shows one exemplary start configuration, while on the right the corresponding end configuration can be seen. The darker objects (dark blue and black) represent the actual block and actuator, while the transparent objects symbolize the predicted objects.

All starting positions except the first three are chosen randomly, but using the same seed for all models and configurations. The first three are chosen so that the actuator touches the object on the outer left, the outer right side and directly at the center. This has been done to ensure that the models have seen all general interactions for when they are evaluated with only 3 runs. The random start positions are chosen in such a range, that the actuator can also pass the object without touching it, by sampling starting positions between -0.35 and 0.35. Another seed is set after the training is done, so that all models are tested on the same starting positions, regardless of the number of training examples.

The number of iterations used here is 150 when using the model at 100Hz. During these iterations the actuator moves approximately 75cm. The maximal distance is set to 1m when using the model with 10Hz. The actual action is the same for both frequencies and is that to 0.5m/s upwards towards the block from the starting position.

6.1.2. Evaluation criteria

This task tries to evaluate the precision of the different models. In order to measure this precision, the predicted actuator position is compared with the actual actuator position. Since at least the block can also rotate around the third axis, the orientation needs to be considered as well when measuring the prediction performance. Since it is hard to find a metric that adequately combines point differences with angular differences, the orientation is indirectly measured. In addition to the center position of each object, two key points are defined that are located at the edges of the block as can be seen in figure 6.2. This way the euclidean distance can be used to measure the prediction accuracy of both the object's position as well as their orientation. The three distances are then averaged to compute the final difference score s:

$$s = \frac{1}{3} \sum_{i=1}^{3} ||p_i^{pred} - p_i^{actual}||$$

where $p_i^{pred}, pred_i^{actual}$ represents the i's predicted and actual reference point, including the center, respectively.

These scores are computed at each timestep and accumulated. Furthermore, the final difference, at the last timestep of each run is recorded separately. The results show the averaged results for the final difference, the accumulated differences and the mean difference over 20 test runs. The mean difference is computed by dividing the accumulated difference

Figure 6.2.: TODO Image of the reference points of the block object. The two selected edges, along with the center of the block, make up the three points that are used to compare the predicted position with the actual position.

Model configuration	Actuator final dif	Actuator cumDif	Actuator mean dif
Pairwise AC DT 3 Train	0.001	0.095	0.001
Pairwise AC DT 10 Train	0.053	3.929	0.026
Gate HG HA 3 Train	0.001	0.128	0.001
Gate HG HA 10 Train ITM η 0.0	0.001	0.127	0.001
Gate HG HA 10 Train ITM η 0.1	0.001	0.122	0.001
Gate HG HA 10 Train ITM η 0.2	0.001	0.122	0.001
Gate HG HA 10 Train NoAV	0.001	0.135	0.001
Gate HG HA 10 Train NoDyns	0.001	0.130	0.001

Table 6.1.: Test1

by the number of predictions, the model made in each run, which is equal to the number of iterations-1. For the first iteration, there has not been a prediction that can be compared yet.

6.1.3. Results

[TODOSome first results of the interaction model. Not perfectly tuned yet, as the DT often selects strange things]

6.2. Move to target

6.2.1. Scenario description

6.2.2. Evaluation criteria

6.2.3. Results

Model configuration	Block final dif	Block cumDif	Block mean dif
Pairwise AC DT 3 Train	0.209	14.051	0.094
Pairwise AC DT 10 Train	0.142	10.841	0.073
Gate HG HA 3 Train	0.048	3.061	0.021
Gate HG HA 10 Train ITM η 0.0	0.052	3.189	0.021
Gate HG HA 10 Train ITM η 0.1	0.046	2.968	0.020
Gate HG HA 10 Train ITM η 0.2	0.061	3.538	0.024
Gate HG HA 10 Train NoAV	0.050	2.859	0.019
Gate HG HA 10 Train NoDyns	0.068	4.670	0.031

Table 6.2.: Test2

Chapter 7

Conclusion

- Discuss single timestep decision
- Discuss results/different concepts/models
- Reference state of the art and compare if possible
- discuss limitations/possible improvements (of concept/implementation separately)

Chapter 8

Bibliography

[KH] Koenig, Nathan ; Howard, Andrew: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* Bd. 3 IEEE, S. 2149–2154

[S+05] Smith, Russell u. a.: Open dynamics engine. (2005)

Appendix A

Eidesstattliche Erklrung

Ich versichere hiermit, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Die Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Bielefeld, den XX. Monat 20XX

Nachname, Vorname