Object Grasping with the NAO

Egbert van der Wal

April 3, 2012

Master Thesis
Artificial Intelligence
University of Groningen, The Netherlands

Primary supervisor:
Dr. M.A. Wiering (Artificial Intelligence, University of Groningen)

Secondary supervisor:
Dr. C.M. van der Zant (Artificial Intelligence, University of Groningen)

Abstract

With autonomous robots becoming more and more common, the interest in applications of mobile robotics increases. Many applications of robotics include the grasping and manipulation of objects. As many robotic manipulators have several degrees of freedom, controlling these manipulators is not a trivial task. The actuator needs to be guided along a proper trajectory towards the object to grasp, avoiding collisions with other objects and the surface supporting the object. In this project, the problem of learning a proper trajectory towards an object to grasp, located in front of a humanoid robot, the Aldebaran NAO, is solved by using machine learning. Three algorithms were evaluated. Learning from demonstration using a neural network trained on a training set of recorded demonstrations was not capable of learning this task. Using Nearest Neighbor on the same training set yielded much better results in simulation but had more problems picking up objects on the real robot. A form of Reinforcement Learning (RL) tailored to continuous state and action spaces, the Continuous Actor Critic Learning Automaton (CACLA), proved to be an effective way to learn to solve the problem by exploring the action space to obtain a good trajectory in a reasonable amount of time. This algorithm also proved to be robust against the additional complexity of operating on the real robot after being trained in simulation, bridging the reality gap.

Table of Contents

Chapter 1

Introduction

Autonomous robots have an increasing importance in society. As the possibilities increase, they become more and more useful in our lives, being able to assist us with tasks of everyday live. Some commercial robots that perform useful tasks in daily live have already been put on the market, such as the Roomba vacuum cleaning robot (Tribelhorn & Dodds, 2007) or the Lawn-mowing-robot (Hagedon et al., 2009). While these robots perform a valuable service to their owners, their abilities are very limited: they are fully targeted at one specific task. The reasons for this are two-fold. Firstly, developing the software to control these robots is much simpler if the task to be performed is simple and well-constrained. Secondly, the hardware costs can be reduced to select all the actuators and sensors in a robot for one specific task instead of attempting to account for all configurations.

Research on more general purpose robots has also seen much progress, although few commercial products have been launched because most solutions are not perfect. However, many tasks can already be performed by autonomous robots. Progress in the research field of robots that can assist people in their daily living has been stimulated over the last couple of years by the launch of a new league in the RoboCup (Kitano et al., 1997) competitions: the RoboCup@Home (Wisspeintner et al., 2009, 2010), being organized since 2006. In these competitions, participating teams compete with self-built robots that have to perform a selected set of tasks and are scored based on the performance of their robots. Each year one RoboCup World Cup is organized but several countries also organize local competitions allowing teams to benchmark before participating in the World Cup. These competitions focus the research of participating teams on the relevant tasks required to score points.

Many tasks require that the robot is able to pick up and safely transport objects from one place to another autonomously. To achieve this, the robot needs to have some kind of manipulator mounted on it. Many forms of manipulators are available, such as the

LMS Mechanical Hand[1] and Barrett Arm and Hand[2]. Most manipulators are aimed at providing the best manoeuvrability to the end-effector. The downside is that usually these agile robotic manipulators have been designed for research or industrial tasks, making them large and unattractive to have on a robot meant to assist in a domestic environment. The RoboCup@Home team of the University of Groningen[3] instead uses a combination of a wheeled robot, the ActivMedia Pioneer 2[4] with the humanoid robot of Aldebaran, the NAO, mounted on top of it. The NAO has a cute appearance, and is immediately appealing to non-technical people. One drawback of the NAO robot is that severe constraints have been put on the size, configuration and strength of the actuators. To keep in proportion with the rest of its body, the NAO only has short arms. Its hands are controlled by one motor and therefore the fingers cannot be operated independently. This strongly limits the abilities of the platform to pick up objects of different sizes and shapes from different positions.

Controlling this kind of robot requires a whole new approach to be able to successfully grasp objects. Firstly and most importantly, the robot needs to plan a proper trajectory for its arms to actually reach the object, avoiding the surface supporting it. Then, it needs to clamp the object between its hands and lift it. Since the fingers of the NAO have low strength and a moderately smooth surface, the range of objects that can be picked up is small: only light, small objects with surfaces that are not too smooth can be picked up as other objects will slip from the NAO's fingers. Also, control has to be very fine grained and specific to the exact situation. Discretizing the action space therefore severely limits the possibility of any learning system to obtain a proper solution to the problem. Any learning system attempting to solve this problem must therefore be able to cope with both continuous state and continuous action spaces. Also, the problem can be approached in two ways: estimating the correct angles for the joints directly or estimating the angular difference between two successive states. The first approach will be referred to as 'absolute angles' in the rest of this thesis, while the second approach will be referred to as 'relative angles'.

This master thesis reports on the attempt to solve this problem using three forms of machine learning. The first one is learning from demonstrations (Schaal, 1997) where the controller of the robot is trained on demonstrations recorded when the hands of the NAO were guided by a human towards the object. The second one uses the same demonstrations in a much more direct way: K-Nearest Neighbor regression (Cover & Hart, 1967). This compares the current state with all the states in the training set and selects the best matching examples to generate the action. The third algorithm is a form

[1] www-lms.univ-poitiers.fr/article167.html?lang=en
[2] www.barrett.com/robot/products-arm.htm
[3] www.ai.rug.nl/crl/
[4] www.mobilerobots.com/

of reinforcement learning that is able to cope with both continuous state space and continuous action space: the Continuous Actor Critic Learning Automaton (Van Hasselt & Wiering, 2007). This algorithm is an actor-critic system adapted for continuous state and action spaces. Both reinforcement learning in general and the CACLA algorithm will be discussed in-depth in section 2.5. There, a variant on CACLA that uses the variance of the TD-error to determine the number of updates to the actor called CACLA+Var will also be discussed.

Success is determined by the robot itself during training: when the hands are in an appropriate location relative to the object this is considered a success. This feedback is then used to continue training the system and increasing its performance. In the final experiments on the real robot a human decided whether an attempt was successful or not.

Results show that the artificial neural network trained on demonstrated trajectories is unable to learn to perform the correct behavior, but that nearest neighbor regression on the dataset does show excellent results in simulation. This method performs a lot worse on the real robot. The results also show that Reinforcement Learning using CACLA+Var is able to learn the correct trajectory from the starting position to the location of the object. On the robot, a success rate even higher than the success rate in simulation was measured.

1.1 Related Work

There are few publications on object manipulation on the NAO as of yet, but object manipulation in particular and robotics in general has been subject to a lot of research. Some of this research will be discussed in this section.

1.1.1 Domestic Service Robots

Robotics performing service tasks in a domestic environment have been in people's minds for a long time, as can be seen in many science fiction stories and movies. Over the past few decades the feasibility of service robots has increased and more research groups have started to do research in the field of robotics. The RoboCup is an international robotics competition founded in 1997 (Kitano et al., 1997), that aims to speed up research and cooperation in robotics. The competitions are divided into multiple leagues, each focusing on different applications such as soccer, rescue and simulated robotics. In 2006, a new league was introduced, the RoboCup@Home (Wisspeintner et al., 2009, 2010). This league aims at the application of many different applications

in robotics to construct an autonomous *domestic service robot* that is able to assist its users in a domestic environment. The competition is formed by a set of tests that each participating robot can perform to score points. Points are rewarded for performing parts of each test to stimulate teams to participate even when their robot cannot complete the full test yet. Tasks include welcoming guests, finding and retrieving objects and recognizing people. Also, to stimulate any research relevant to the research field there is a test in which the teams can showcase any interesting project they have been working on in the Open Challenge. Since 2006, many teams have participated and also published reports on their scientific contributions, e.g. Holz et al. (2009); Graf et al. (2004); Chacón et al. (2011).

1.1.2 Object Grasping and Manipulation

Numerous researches have focused on object grasping and manipulation, benchmarking grasps, grasp synthesis and object rotation. In the following sections, some of these researches will be discussed.

Motor Control

While controlling motors of robots can be modeled explicitly, for example by recording trajectories and executing these at a later time, this requires a lot of manual labor and the result will only be applicable in situations much like the one in which the trajectories were recorded. A report on attempting to solve these problems using machine learning is presented by Peters & Schaal (2008b). An approach to generate the building blocks of movenent, *motor primitives*, is presented in Peters & Schaal (2006).

Grasping Novel Objects

An article by Saxena et al. (2007) reports on a research to grasp novel objects in a cluttered environment. While previous researches relied on detailed 3D-models of the objects to grasp, this research made no such assumptions. In their approach a 3D model was built from vision. Using this model, possible grasping points were identified and the points that were best reachable from the robot's position were selected. Based on this information and perception of the environment, a path was calculated for the robot arm to successfully reach the object without hitting obstacles. While their approach gave some good results, they acknowledge that their algorithm failed to find a path when there was much clutter around the object to grasp. Still, they report 80 percent success rate in a test where the robot had to unload a dishwasher. They further investigated the subject in a follow up paper (Saxena et al., 2008) where they accounted for more degrees of freedom, for example a robot hand with multiple fingers. In this case, not only the grasping points on the object need to be selected, but also the appropriate position

for all the fingers while grasping the object. Using their new approach, they performed several trials on grasping a set of objects of varying sizes from cluttered and uncluttered environments. They report success rates from 90 to 100 percent for medium sized objects.

Opening Unseen Doors

Another research investigated the opening of unseen doors by a robot (Klingbeil et al., 2008). In this case, a robot is moving through an unknown environment. In order to access new locations, it is able to detect door handles or elevator buttons and recognize how to manipulate those objects. They did impose a few constraints because their initial approach gave many false positives. They incorporated, for example, the knowledge that doors have at least one and at most two door handles. If there are two, they are probably close to each other. Based on these heuristics, the robot was able to successfully perceive the location of door handles. They used PCA on the 3D point cloud generated from the image to determine the way to manipulate the door handle: whether it is a right-turning or left-turning handle. Their robot was able to open the door in 31 out of 34 experiments.

Properties of Objects

For manipulation tasks, certain properties of objects are very useful to increase performance. For example, statistics such as weight, size and structure information are useful to select the appropriate amount of force to exert and the location where the object can be grasped. The friction between the object and the surface it is placed on is also an important factor. The force required to displace an object on various surfaces can be a useful statistic, which is what was measured in a paper by Matheus & Dollar (2010). They measured the force required for the displacement of a set of objects occurring frequently in daily live when placed on a set of common surfaces such as glass, granite, stainless steel and others. Properties of objects can also be deduced by dynamic touch, e.g. by shaking the object. In Takamuku et al. (2008) a research is presented that extracts additional information about the object by shaking it with different speeds. By recording the sound the object makes while shaking it, they were able to distinguish between a bottle of water, paper materials and rigid objects. Intra-category differences were small while inter-category differences were large. However, the classification will become harder when more types of object categories are added.

Trajectory Planning

In order to successfully grab an object, the manipulator must first be brought close to the object and objects need to be avoided. A research by Hsiao et al. (2011) reports on an

attempt to do this using *World-Relative Trajectories* (WRTs). They model the state and action spaces into discrete belief states and end-effector trajectories in Cartesian space. Using continuous updates of the belief states, they managed to increase the robustness of the system. However, they did have to provide planned trajectories to work with and also used inverse kinematics to execute the motions, requiring a precise kinematic model of the actuators.

Grasp Synthesis

When the object has been approached, the correct locations to actually pick up or manipulate the object need to be selected. Different approaches are usable, such as learning from demonstration or automatic selection. In Daoud et al. (2011) an approach using a genetic algorithm to optimize a grasp for certain objects is discussed. They were able to synthesize correct grasping poses to pick up a set of objects using three of the four available fingers on their manipulator, an LMS mechanical hand. Control of multi-fingered robot hands has been studied in more detail in a reviewing paper by Yoshikawa (2010). Different kinds of grasp synthesis methods are discussed, both for soft and hard fingers. Soft fingers are harder to control as they can be deformed and can thus be controlled less precisely. However, the deformation capability allows for more firm grasping of certain objects by forming more around the object and by providing more friction. A grasp pose can be tested by attempting to pick up the actual object. However, estimating the quality of a grasp beforehand can lead to improved results. In Cheraghpour et al. (2010), a method to estimate the quality of a grasp using a *multiple aspect grasp performance index* is discussed.

Inverse Kinematics

Positioning the hands correctly to pick up an object has a strong relation with inverse kinematics: the joint configuration of the arm to reach the object must be such that the manipulator ends up near the correct coordinates in Cartesian space. While this problem can be modeled and solved using equations, there are usually more than one way to reach the same position and in that situation a decision must be made which solution is the best. An attempt to solve this problem without a model but by approximating it by learning directly on the position level is presented by Bocsi et al. (2011). An approach to learn the building blocks of movement, *motor primitives*, using reinforcement learning is discussed in a paper by Peters & Schaal (2006).

1.1.3 Behavior Selection

The researches by Saxena et al. (2007) and Saxena et al. (2008) use a different strategy than the research by Klingbeil et al. (2008). They first use little prior knowledge: they

estimate proper grasping points which are then used to move the hand to the proper location to grab the object. The latter research uses several trained strategies for opening doors, where the optimal strategy was selected based on visual input. In the proposed research the system needs to do a combination of both: it needs to select the proper grabbing strategy based on recognition of object types. A method for selecting behaviors is reported by Van der Zant et al. (2005). This method implements exploration and exploitation in a natural fashion. The success and failure rates of behaviors are stored for each target. When a behavior selection is required, the system looks at the confidence interval of each behavior for the selected target. By selecting the behavior with the highest upper bound the system will explore when too little data is available for the confidence interval to be small but will naturally switch to exploitation when confidence intervals become smaller. This method was also applied in a bachelor thesis by Oost & Jansen (2011), that reports on an effort to train the NAO to mimic grabbing behaviors. The grabbing behaviors were selected inspired by how humans grab objects and the NAO was trained to perform these behaviors on command. Using interval estimation, the best behavior for each situation was determined.

Another approach is presented by Van Hasselt & Wiering (2007). In this paper, a Continuous Action Critic Learning Automaton (CACLA) is used to map continuous input onto continuous output. This algorithm is an actor-critic system well-suited for continuous state and action spaces by using a function approximator to learn both the value function and the policy. By exploring the action space sufficiently, CACLA can be used to optimize a policy to achieve the goal.

1.1.4 Object Recognition and Pose Estimation

Interpreting the data obtained from cameras is not an easy task. Many factors influence the output such as lighting conditions and camera parameters such as exposure, gain, white balance and resolution. Humans are able to recognize objects under extremely varying circumstances robustly and much research has been devoted to achieving the same level of performance of object recognition in machine vision. Because lighting conditions vary, color values are usually not a robust indicator of object properties. So far, the most robust properties of objects in camera images have proven to be descriptors that describe the spatial organization of salient features of the images, which are usually the edges in an image, as these are the easiest to detect and provide much information about the structure of the object. One algorithm that uses this information is SIFT (Lowe, 2004), which detects the most stable features of an image under various scalings and stores the direction of the edges as a descriptor of 128 values. While this approach is reasonably robust and copes with rotations and scaling rather well, it is relatively expensive to compute and the length of the descriptor results in long match-

ing times when there are many features in the database to compare with. A different algorithm, also using spatial information, is SURF (Bay et al., 2006, 2008). This algorithm results in descriptors of 64 values, using the Haar Wavelet responses. Also, the features are all based on the same scaling by generating the Integral Image from the original image to begin with (Viola & Jones, 2001). As a result, less time is required to calculate the descriptors. Matching the resulting descriptors with a database of known feature vectors is also more efficient because the feature vector is only half the size of the feature vector used in SIFT.

Another method for object recognition is presented by Malmir & Shiry (2009). The method described in this paper is inspired by the *primate visual cortex*. They implemented a system performing roughly the same functions that the V2 and V4 areas in the primate brain perform. In addition, the already established alternative for the V1 area, Gabor filters (Jones & Palmer, 1987) is used. While they do report optimistic results, they present results on just 6 images from a dataset which does not seem enough to establish the quality of the method. Earlier however, Van der Zant et al. (2008) reports on using biologically inspired feature detectors for recognizing handwritten text in a handwriting recognition system called Monk. They used a model presented based on Gabor functions, local pooling and radial basis functions, described in Serre et al. (2007). They report an accuracy of up to 90% on a large dataset of 37,811 word zones.

Any object recognition method will benefit from better images. Instead of attempting to work with bad images, Lu et al. (2009) attempt to improve image quality by optimizing the entropy of the image, as entropy is a good measure of the amount of information available in the image. By adjusting certain camera parameters such as gain and exposure time, they were able to improve the image quality significantly in several hundred milliseconds.

Naturally, the processing speed of these algorithms automatically increases over time by technological advancements resulting in faster hardware. However, the features computed by the methods mentioned above are largely independent and only depend on the direct surroundings of each pixel. This makes the features independent and thus easily parallelizable. Therefore, implementations of both SIFT and SURF for use on the Graphical Processing Unit (GPU) have been made. GPUs are extremely suitable for highly parallel computations and can thus increase the processing speed tremendously. Using these approaches it is often feasible to process complex scenes with many objects with almost real-time performance, making them extremely suitable for use in domestic service robots.

While SIFT and SURF are reasonably robust against rotations in the image plane and scaling, other rotations pose a problem. Therefore, the algorithms need to be trained

on several images of the object in several different poses. A 3D model of the object can help to improve the training. Furthermore, because the descriptors of the objects are independent, problems often occur when multiple instances of the same type of object appear in the same image. An attempt to unite several algorithms to integrate images from multiple cameras of the same scene and 3D models generated from images to recognize objects and estimate their pose in the real world is made in MOPED (Collet et al., 2011). Their results indicate reliable recognition, even of many different instances of the same object and in highly cluttered environments.

A different approach was taken by Kouskouridas et al. (2011). They detect objects using their features as detected by SIFT or SURF. From this information, they form an outline of the image, resulting in a binary image containing the general shape of the object in the image plane. Using this information and a training set, they were able to estimate the pose of objects successfully with good accuracy: a mean error $\approx 3cm$ when using SIFT and a mean error $\approx 5cm$ when using SURF.

1.2 Research Questions

In this thesis, one main research question and two sub-questions will answered:

1. "Can machine learning algorithms be used to control the joints of a humanoid robot in order to grasp an object?"
   (a) "Which of the evaluated algorithms, learning from demonstration, nearest neighbor or CACLA, performs best on the task of grasping an object?"
   (b) "Which form of control, the target angular values for the joints or the angular difference relative to the current state of the joints, is better suited for machine learning?"

These questions will be evaluated on the results obtained from the experiments and they will be answered in the conclusion of this thesis.

1.3 Outline

The outline of the remainder of this thesis is as follows. Chapter 2 will discuss the various machine learning algorithms, such as learning from demonstration, reinforcement learning and K-Nearest Neighbor. It will also discuss the object recognition algorithms used in this research. Chapter 3 will give insight into the hard- and software architecture used to perform the research. It gives details about the geometry of the NAO

humanoid robot and about the software used to control it. Chapter 4 will present the details about the experiments performed for this research and their results. The implications of these results will also be discussed in this chapter. Chapter 5 will conclude the thesis and answer the research questions posed in the previous section. It will also discuss what ends were left open and give suggestions for further research into the field of robotic machine learning for motor control.

Chapter 2

Robot Learning and Object Recognition

In this chapter, the setup of the project will be discussed. It has several sections, describing the methods utilized in the corresponding parts of the project. First, a generic overview of the project is given, followed by the detailed overview of the individual parts.

2.1 Grabbing System

Building on the hardware of the NAO and the robot architecture in use, the project is naturally split into two parts: the object recognition as an external module, and a higher-level behavior to grab an object. This higher-level behavior is split into three sub-behaviors. The first sub-behavior finds out where the object to grab is located. The second sub-behavior actuates the motors of the NAO to pick up the object. The third sub-behavior validates that the object has indeed been picked up. The behavior architecture is shown schematically in figure 2.1. Each of these systems will be described in more detail in the following subsections.

2.1.1 Object Recognition

In order to successfully grasp an object, some of its features must be known. Essential features are its location and dimensions. Other information could also be useful. If a model of the object is available, the Generalized Hough Transform (Ballard, 1981) is able to find transformation parameters that give a best match in mapping the model onto an actual image. This is much harder when there is no model available, which will be the case when grasping unknown objects. If unknown objects should be recognized

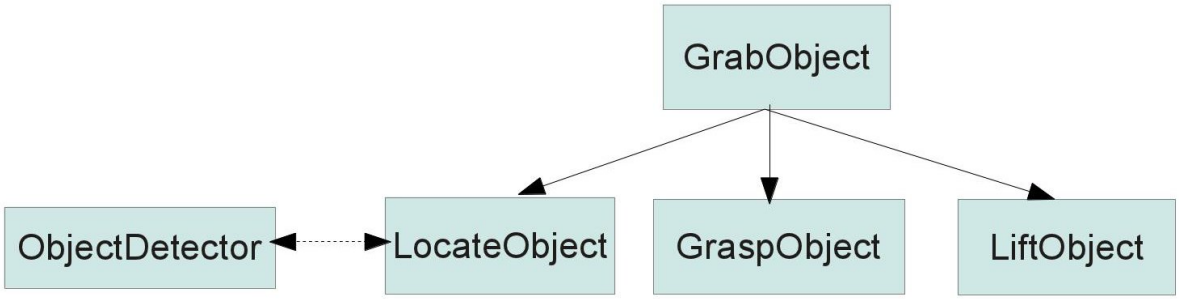


Figure 2.1: The Behavioral Architecture of the Grabbing System

and labeled, feature detectors such as SURF (Bay et al., 2006) or SIFT (Lowe, 2004) can be used. These methods result in a set of features which can be stored in an object database. New observations can then be compared with this database to see if the object can be recognized. Using this method, the system will be able to learn to recognize new objects without user intervention, which is an appealing feature for this project. It will be assumed that the objects are located in an uncluttered environment: for example without any distractions on a table or in an organized, open closet. These limitations are implied by the design of the NAO: it has relatively short arms making it harder to avoid lots of obstacles. This lowers the demands on the object recognition algorithm.

For the initial experiments, a basic approach was taken, reducing the dependency on the vision system in order to evaluate and optimize the grasping system first. The system did not use any object recognition at all, but instead requires a human to locate the object in the camera image and select it. The system then calculates the position of the object in the real world based on the assumption that the object is always located at 16.5 cm in front of the robot. The arms of the robot are little over 20 cm so the robot cannot reach further than around 18 cm in front of it. However, objects are usually placed lower than the shoulders and this reduces the reaching distance of the NAO's arms because they also have to reach down. At around 34 centimeters height, around its waist, the robot is able to comfortably grab objects that are 16.5cm in front of it. Therefore, this distance was used for all experiments, even though the vertical position of the object was varied.

In the final experiments on the robot it was attempted to actually recognize the object using SURF descriptors and using these descriptors to estimate the position of the object.

2.1.2 Grabbing the Object

Once an object has been detected and selected for grabbing, the second sub-behavior obtains the dimension and position of the object from the first sub-behavior. The second sub-behavior will interpret these data and try to find the right joint angles required to position two hands at appropriate positions on the object.

Three methods will be used to generate the proper sequence of actions: learning from demonstrations (Schaal, 1997) with artificial neural networks, learning from demonstrations using Nearest Neighbor regression (Cover & Hart, 1967) and reinforcement learning using the CACLA algorithm (Van Hasselt & Wiering, 2007). For the first two methods, a large dataset has been formed containing 1000 demonstrations where the NAO's arms were guided towards the object, avoiding the surface supporting the object in the process. For these demonstrations, much of the available data about the current state of the NAO was recorded, such as the angles of all the joints, measurements of the accelerometers and the camera image from the active camera. The dataset was formed by demonstrating how to grasp an object at four different heights. For each of those four heights, roughly ten demonstrations were recorded where the object was moved a small distance from the right side of the scene to the left side of the scene after each demonstration. This results in roughly 40 demonstrations per object. Because the position of the object varied, the object was not always equally visible in each camera. The camera whose field of view was closest to the object was used to look at the object. For the lowest placed objects this was always the bottom camera while for the highest placed objects this was always the top camera. Since both cameras are the same and also share the same parameter settings, impact on performance by switching cameras is minimal. The position of the object was always calculated in *NAO Space*, one of the three spaces defined in NAOqi, the API for controlling the NAO. The other two spaces are *Torso Space* and *World Space*. Torso Space is the space with the origin in the center of NAO's torso, with the Z-axis extending upwards along the spine. NAO space is the space centered between NAO's legs with the Z-axis pointing upwards. World Space is initially equivalent to NAO space when the robot boots up. However, the World Space has a fixed origin and orientation while NAO Space moves with the NAO. Because the NAO's feet did not move during the experiments in this project, NAO Space and World Space were equivalent.

The second method, Nearest Neighbor regression, was implemented using the Fast Library for Approximate Nearest Neighbor (Muja & Lowe, 2009), a fast implementation of the nearest neighbor algorithm (Cover & Hart, 1967) which can be used for regression on datasets.

The third method that was evaluated is the Continuous Actor Critic Learning Au-

tomaton, CACLA (Van Hasselt & Wiering, 2007). This method can be used with untrained, randomly initialized networks. Alternatively, the actor can be bootstrapped with a network trained on the pre-recorded demonstrations. Using the trained network can speed up training significantly but will also bias the results more towards this initial solution. An untrained, randomly initialized network makes sure that there is no bias and the action space is explored to obtain the best solution. In this research, both a randomly initialized actor and a pre-trained actor were evaluated to compare their performance. Also, not the default CACLA version described in Van Hasselt & Wiering (2007) was used, but instead a variant on it presented in the same paper, CACLA+Var was used. CACLA+Var differs from CACLA in that it uses the variance of the size of the TD-error to determine the number of updates to the actor.

All training algorithms were used with the same set of outputs: the joint values for the relevant joints. Each arm of the NAO has 6 Degrees Of Freedom (DOF), but of these, only five are relevant for grabbing objects with two hands. The last one is the joint controlling the opening and closing of the hand. This joint does not add to the possibility to solve the problem but it does extend the action space, making it harder to find a solution. Therefore, that joint was ignored during training. The hand joints allow closing of the hands but this is not useful for the objects used in this project as NAO's hands are too small to fit around these objects.

To reduce the search space even further, only one hand can be explicitly controlled. The other hand can then mirror the movement of the controlled hand. As grabbing usually involves a lot of similarity between the two hands, this is a sensible simplification. The only additional limitation that this imposes is that the object must be centered in front of the NAO before attempting to grab it. In practice, this is not a limitation, because the robot can solve this problem by performing a rotation or a few steps to the side to change its pose relative to the object so that the object will be centered in front of the robot.

2.1.3 Parameter Optimization

From this information, the next task is to find the best suited representation of each state and the optimal set of outputs of the system that result in the best performance. Because this kind of optimization is quite hard and tedious to do by hand and will also take up a long time, a parameter optimization algorithm was used that was able to automatically generate new sets of parameters based on the performance of previous sets of parameters and evaluate those sets. Given enough time to run, this method will find nearly optimal parameters, much better than possible by hand. The algorithm used by parameter optimization software was the Covariance Matrix Adaptation Evolution

Strategy (CMA-ES), presented in Hansen et al. (2003). This algorithm was combined with a bandit using *upper confidence bounds* (Kocsis & Szepesvári, 2006) that always evaluates the most promising offspring first to avoid evaluating bad performing offspring repeatedly to save time. The program evaluating the algorithms was developed internally and tests were run to find the best methods to optimize sets of parameters based on several experiments (Van der Wal, 2011). The algorithm was set up to vary the input to the algorithm and the parameters of the algorithm such as the size of the hidden layer, learning rate, etc. From each configuration, the performance was evaluated.

To evaluate the performance, the dataset of 1000 demonstrations as discussed in section 2.1.2 was divided into a training set of 70% of the demonstrations and a test set of 30% of the demonstrations. The training set was used for training the algorithm and then performance was evaluated on the test set. Afterwards, the set of inputs that provided the best performance was used in further training. The parameters that were optimized include the parameters of the artificial neural network used for learning from demonstration and CACLA+Var, the inputs to provide to the system and the units of those inputs (e.g. meters or centimeters).

2.1.4 Self Evaluation

To be able to learn and improve the grasping skills, the system needs to know whether it was successful. The major part of training was performed while running in a robot simulator, so approaches using the strain on the engines cannot be used. Therefore, the algorithm evaluates the pose of the hands by calculating their position in Cartesian space and the distance to the object. The goal state was defined as having the hands close to the object, facing each other. When the robot reached this state, the attempt was considered a success. For experiments on the real robot, the evaluation of the grasping attempt consisted of a human monitoring the trials and deciding when the hands are in an appropriate position to pick up the object. Picking up the object then occurs by moving the hands closer together and then moving the arms upwards. Success was evaluated by checking if the object had actually been lifted.

2.2 Input and Output

This section will describe the input and output used in all the machine learning algorithms evaluated in this research. The input represents the state the robot is currently in and the output represents the action to take in the current state.

2.2.1 Input

The input must represent the state of the robot and the environment sufficiently to be able to select the appropriate action in each state. It should therefore incorporate information about the dimensions and position of the object that must be grasped. Without this information, the system would not be able to find the correct location. Also important is the current position of the arms of the robot. This information can be represented in various forms, for example the current angles for each joint in the arms of the NAO or the Cartesian coordinates of the hands of the NAO. The first form provides the most information because multiple configurations of joint angles can lead to the same position. However, the position of the hands has a more direct relation with the problem of moving the hands towards the object of which the position is known, because the units and dimensions for these numbers are equal. On the other hand, multiple configurations of the joints can lead to the same position of the hands, so that possibly relevant information is lost.

For the experiments in this project, the following set of inputs was used as a state representation: the current angles of all 10 arm joints, the coordinate tuple (x, y, z) describing the position of the center of the object in meters, relative to the robot, the dimension tuple (w, h) of the width w and the height h of the object to grab in meters, and finally the distance in meters from the left hand to the object and from the right hand to the object, resulting in a total state representation of 17 inputs. These features were selected because the empirical research using parameter optimization techniques described in 2.1.3 suggested that the best results could be obtained by using these features.

2.2.2 Output

The output must represent the suggested action for the robot to take in the current state. Again, multiple formats can be used for this. The algorithm could output either angles for the joints or Cartesian coordinates to which to move each hand. The problem can be approached in a local or in a global way. In the global approach, the algorithm outputs the next exact angle configuration or position to move to. In the local approach, only the difference with the current state or the direction to move in is output by the system. The advantage by the local approach is that the meaning of the values are the same in each state while they result in different state transitions. Because the relative movement is limited to a small area the NAO is able to reach in one time step from the current state, the system can much better exploit the available output range of the function approximator being used, giving it more opportunity to learn. To obtain a valid value for the maximum movement of each joint, the 1000 grabbing demonstrations recorded for this project, discussed in section 2.1.2, were analyzed. The difference in joint angles

between each time step was calculated. For each joint, the standard deviation of these differences was calculated. To accommodate the majority of these differences the standard deviation σ was used. To standardize this over all joints, the largest value for σ for all arm joints was used, which is the shoulder pitch. For the shoulder pitch, $\sigma = 0.0574$ was obtained, so a maximum change of angles of 0.0574 was set for all joints. This was then scaled to the interval $(-1, 1)$ to match the output space of sigmoid function of the ANN.

2.3 Learning from Demonstration

Learning from demonstration (LFD) (Schaal, 1997) is a method that can be used to train a function approximator to perform some robot task. Recordings are made of demonstrations by the human. These demonstrations should be performed with the same hardware the algorithm should work on, except that now the control lies with the human. All relevant data is recorded that might influence the decisions the human makes. When enough data is collected, a training set can be formed that formalizes the exact input for each situation and the correct output for that situation. This training set can then be used to train the function approximator. For this research, an Artificial Neural Network (ANN) was used. Specifically, the open source software FANN (Nissen, 2003) was used. This program is a highly optimized implementation of an ANN with support for various training algorithms such as back propagation (Rumelhart et al., 1986), RPROP (Riedmiller & Braun, 1993), quickprop (Fahlman, 1988) and completely different approaches to train neural networks such as cascade-correlation training (Fahlman, 1990) that dynamically adds new units to an already-trained ANN to improve the quality. It also has partial support for recurrent ANNs and adapted strategies to train them (Pineda, 1987).

One additional way to optimize the performance of ANNs is to use ensembles (Hansen & Salamon, 1990). By using a set of similar networks having the same outputs, the generalization of the networks can be improved by averaging their outputs. This will reduce bias of any of the networks towards any training set, assuming that the networks were initialized with different random weights, and optionally have differing structures. In the experiments for this project, both single ANNs and ensembles of ANNs were used to test the performance.

The robot control module receives a set of inputs from the behavior system of the architecture as described in section 2.1.2. These inputs consist of properties of the object to grab and the current state of the motors. See section 2.2.1 for more information about the inputs. The ANN is run on these inputs and produces a set of outputs. The outputs represent the new position for the arms to bring it closer to the object. See sec-

tion 2.2.2 for more details about the output of the system. ANNs perform best when their inputs and outputs are scaled to some limited interval, usually $(-1, 1)$, the value range of the symmetric sigmoid function usually used as activation function of ANNs. The inputs and outputs are scaled to match this range. The output of the algorithm is then used to control the NAO's joints.

Using the parameter optimization program discussed in section 2.1.3, the parameters for the ANN were determined. The best results were obtained with batch training using back-propagation with a learning rate of 0.0001. The networks consist of three layers, one input layer, one hidden layer with 200 hidden units and one output layer. The networks were trained on the dataset for 30,000 epochs, at which point the training error stagnated.

2.4 K-Nearest Neighbor Regression

The problem described is to obtain an action based on the current state of the system: at least involving the current angles. This output is continuous and can therefore be regarded as a regression problem. One non-parametric way to solve a regression problem with a dataset is the Nearest Neighbor (NN) algorithm (Cover & Hart, 1967). This algorithm relies on the fact that similar inputs will usually lead to similar outputs. So, when input is fed into the system, it compares this input to all the known samples and finds the closest example using a distance measure, for example the Euclidean distance. The output of this trained example is then used as the answer. The algorithm can be generalized to K-Nearest Neighbor (KNN) where not just the closest neighbor but the K nearest neighbors are considered. The output of the algorithm can then be interpolated between these nearest neighbors. Since a dataset was collected for this research to train the ANN described in the previous section, a natural option to solve this problem is to use KNN on the dataset. Since finding the nearest neighbor in a large dataset such as this one can take a long time, one can settle for the *approximate nearest neighbor*, also sometimes confusingly referred to as ANN. For this research, an approximate nearest neighbor algorithm was used, FLANN[1]. FLANN is a highly optimized implementation of this algorithm, described in Muja & Lowe (2009). For this implementation, the "*autotuning*" setting was used for FLANN, meaning that it automatically tries to find the best possible parameters for the database while building an index. Once the index was built, the nearest neighbors can quickly be obtained by matching the current state with the dataset, and an interpolation based on the distance to each neighbor can be made. Empirical research showed best results for $K = 3$. Using more neighbors increased the

[1] www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN

performance only slightly but added strongly to the processing requirements of the algorithm and therefore $K = 3$ was used. However, for completeness, $K = 1$, where the output is determined entirely by the nearest neighbor, was also evaluated. For $K > 1$, the output was determined using a weighted average of the nearest neighbors. If d_x is the Euclidean distance from the actual input state to training sample x, the weight w_i for each of the nearest neighbors was calculated as follows:

$$w_{0<x\leq K} = \left(\sum_{i=1}^{K} d_i\right) - d_x \tag{2.1}$$

After all the weights have been calculated, they are normalized to sum up to 1. The resulting weights are then used to calculate the output at time t, X_t by the outputs of each sample, Y_i for $0 < x \leq K$:

$$X_t = \sum_{i=1}^{K} w_i Y_i \tag{2.2}$$

2.5 Reinforcement Learning

Reinforcement Learning (RL) is an online training method that can be used to teach an agent to perform a certain task. The main requirement is that the task can be formulated in terms of states, actions and rewards, and that the total rewards received are maximized when the agent performs the target behavior. The following sections will introduce the RL methodology and provide an overview of the various available algorithms of RL. For a detailed overview of reinforcement learning algorithms, see e.g. Sutton & Barto (1998).

2.5.1 Methodology

Reinforcement Learning formulates a problem using three elements: states, actions and rewards. An agent needs to act in a certain environment. The agent is the entity that must make decisions. The agent does not necessarily, and most often does not, equal the physical agent for which reinforcement learning is implemented. In reinforcement learning, the agent solely consists of the decision making system. All other factors, such as sensors and actuators are considered part of the environment. In essence, the agent consists of all elements of the problem which it can directly influence. The environment consists of all other factors. The agent can influence the environment only by means of the selected actions.

The task of a reinforcement learning algorithm is to select an action that is to be performed in the environment. The agent bases its decision on a summary of the environment: a set of features that contains the most relevant available features about the environment. This set is called the *state* or state description. The state description can contain low level features such as raw readings from sensors or higher level features such as interpreted readings or external information about the problem. It does not necessarily need to contain all relevant information. The state could be partially hidden in a card game for example, where the agent does not know which cards the other players have even though this information would be very useful for the decision. The agent always is forced to deal with the available information and make the best decisions given this limited set of information.

Based on this state representation, the agent selects an action from a possible set of actions. It then performs this action in the environment. As a result, the environment changes, and thus the state representation of that environment. The transition from one state to another state resulting from an action occurs with a certain probability. In stochastic settings one action executed in a certain state can lead to multiple following states, each with their own probability. To accommodate for this, the agent needs to maintain the set of transition probabilities for each action in each state. A specific case of problem settings is the deterministic setting where the transition probability of one state to another is always 1, and 0 to all other states. Executing an action and reaching a new state can provide the agent with a certain reward. In reinforcement learning, the reward is the single most important instrument to instruct the agent what to do. Usually, a reward is given for reaching the goal state. Rewards can also be used to instruct the agent what *not* to do. For example, in board games, a negative reward can be given each time a piece is captured by the opponent, or when the game is lost. If the agent has to drive a car, collisions should be awarded with a negative reward.

Based on the above information, the agent selects an action. Because a reinforcement learning agent is never instructed with the correct action to take, there are two strategies for selecting an action that must be alternated sufficiently to allow the agent to learn a proper strategy: exploitation and exploration. Exploitation is using the agent's knowledge of which action is good in the current state. If an agent exploits its knowledge, it will take an action which is known to lead to high rewards. But because the agent usually has not tried all possible actions it cannot know the expected rewards of all possible actions. Therefore it needs to explore regularly by performing an action which is not the current best known action. By trying this action the agent will learn the expected rewards of this action and if it is better, it can adjust its strategy to increase the probability of selecting this action in the future.

2.5.2 State Representations

As mentioned, state representation of the environment must convey as much relevant information as possible but it should be as concise as possible. Furthermore, reinforcement learning algorithms assume that the state has the Markov property: the current state contains all the information necessary to select the next action. If knowledge of past states is required to make a decision, this information should be summarized in the current state in the most optimal form. Usually, the way the current state has been reached is not relevant, only the effect it has on the current state is important. For example, if the car driving agent must avoid collisions, it does not need to know all the changes of speed and heading in the past, but only the current speed and heading. Formulating the state representation appropriately is essential for the performance of the agent. When the state representation does not contain enough relevant information the agent will not be able to make the best possible decision. Having many irrelevant details in the state representation increases the number of possible states and therefore the complexity of the problem.

Of course, in this way, the state representation is a snapshot of the environment at a certain moment. In a reinforcement learning system, time is usually sliced into discrete elements, time steps. At each time step, the state representation is formulated again from the environment. The state representation at time step t is usually indicated by $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states.

While RL assumes the Markov property, this does not necessarily need to be the case exactly. A near-Markov state representation is good enough for RL algorithms to perform satisfactory.

2.5.3 Actions and Policies

Actions are the one way for a reinforcement learning agent to influence the environment. The output of the algorithm can be a decision on the type of action to take in a certain situation, a value from a discrete set of numbers appropriate for the problem or a continuous function. Some reinforcement learning algorithms only handle discrete actions where the set of possible actions is limited, while others also handle continuous action spaces. For the first type of algorithms, continuous functions have to be discretized at a set level that gives enough flexibility in the actions to take without making the action space too large to handle. The action selected at time t is usually indicated by $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of possible actions in state s_t.

The set of actions to take in each state together form the so-called policy, usually indicated by π. The task of a reinforcement learning system is to optimize the policy to

select the best possible action in each state, which truly returns the highest reward. The policy that accomplishes this is called the optimal policy, usually indicated by π^*.

2.5.4 Exploration Strategies

Exploiting the best known action is called *greedy*. A possible approach to mix exploitation and exploration is ϵ-*greedy*. In this approach, the agent selects an action at random from the set of possible actions in each state with probability ϵ as exploration, and the best known action, the greedy choice, with probability $1 - \epsilon$. While this allows for exploration, the action is selected completely random which might not always be the best approach. Another possibility is to arrange the possible actions by the current estimate of the expected future rewards received after performing each action. The probability of each action is then based upon their rank meaning that the best action has the highest probability to be selected, while the action with the least expected rewards has the lowest probability to be selected. This method is called the softmax method (Bridle, 1990).

A more sophisticated but mathematically more complex method is to use interval estimation to select an action (Kaelbling, 1993). For this method, not just the expected rewards for each action must be kept track of, but also the confidence interval for a set percentage, usually the 95% confidence interval indicating that the rewards of this action will lie between the lower and the upper bounds of the confidence interval with a probability of 0.95. If the action has only been attempted a few times, the confidence interval will be large, while for actions that have been tried numerous times, the confidence interval will be small. To select an action, the agent could then select not the action with the highest expected rewards, but the action with the highest upper bound of the expected rewards. The action with the highest upper bound has a chance to be more rewarding than the action with the highest expected reward. By performing this action, the agent can update the confidence interval and the expected rewards of this action and thus explores the action space.

If the action space is continuous, meaning that there is an unlimited set of possible actions, another strategy for exploration is *Gaussian exploration* (e.g. Van Hasselt & Wiering, 2007). Because continuous actions are usually numeric, exploration can be achieved by selecting the action from a Gaussian distribution with a mean equal to the action that is deemed best by the agent in the current state. The rate of exploration is then determined by the standard deviation $\sigma_{exploration}$ of this Gaussian distribution: the larger the standard deviation is, the more the agent will explore. Also, the standard deviation could be gradually decreased during training to reduce exploration after the agent has had sufficient training. Of course, this same method could also be used for the other exploration strategies.

2.5.5 Rewards and State Values

As mentioned, one of the most important elements in a reinforcement learning system is the reward function, as it informs the system what situations are desirable. The task of the system is to improve the policy so that the agent obtains the highest reward. The reward received in state s_{t+1} after executing action a_t in state s_t is indicated by r_{t+1}. The best situation for the agent is not to obtain the highest reward in any single state. Instead, the best situation is to obtain the highest cumulative reward in all future states. However, it is not optimal to keep all the future rewards in mind when selecting an action. Rewards in the near future should be valued higher than equal rewards in the distant future to make sure the agent performs optimally. To accomplish this, future rewards have to be *discounted* based on how far in the future the reward is expected to be received. For each time step the reward is multiplied by a certain *discount factor*, indicated by γ. So, a reward received 10 steps in the future would be valued in state s_t as $r_{t+10} * \gamma^{10}$. This is the basic notion of a value: the way for a reinforcement learning to look into the future. The state of each value represents the discounted expected future rewards. By selecting the action that most likely leads to the state with the highest expected rewards, the agent performs a greedy action.

With this information, the value of the state can be formalized as $V(s_t)$, giving the value of each state. During training, the value of each state is updated to match the true value of that state. Because the expected rewards received depend on the policy π, the value function also depends on π. The optimal value function that gives the expected rewards when acting according to the optimal policy π^* is given by $V^*(s_t)$. Because the value of each state represents the cumulative discounted rewards of all future states, $V(s_t)$ can be formulated recursively as follows, where E is the expectation operator:

$$V(s_t) = E\left\{r_{t+1} + \gamma V(s_{t+1})\right\} \tag{2.3}$$

2.5.6 State Transitions and the Q-function

As each action a_t results in a change of state from $s_t \rightarrow s_{t+1}$ and yields a reward r_{t+1}, each state-action pair can be given a value representing the future rewards obtained. The function assigning this value to a state-action pair is called the Q-Function $Q(s_t, a_t)$. Where $V(s_t)$ emphasizes the reward obtained executing any action according to a policy π, $Q(s_t, a_t)$ focuses on the value of the action.

2.5.7 Temporal Difference Learning

An approach to learning the state values iteratively is Temporal Difference Learning, introduced in Sutton (1988). The idea of Temporal Difference learning is that at each

time step a Temporal Difference Error (TD-error) is calculated which is the difference between the current value of a function and the new estimate of the current value of that function at a certain time step. For reinforcement learning, this could be applied to the learning of the value function $V(s_t)$, by updating the function with the newly calculated value function according to equation 2.3. If, in state s_t action a_t is selected which yields the reward r_{t+1} and leaves the agent in state s_{t+1}, a new estimate of the value of state s_t can be made by calculating the TD-error $\delta_{V(s_t)}$ as follows.

$$\delta_{V(s_t)} = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.4}$$

The TD-error can then be used to update the value of state s_t:

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_{V(s_t)} \tag{2.5}$$

Here, α refers to the learning rate used to control the size of the updates performed. This value should not be too large, because updating towards one sample usually means updating away from another sample, and the system needs to generalize between all the samples. The value should not be too small either because then learning will be slow. Equation 2.5 is called the $TD(0)$ update rule (Sutton, 1988). This rule updates the state values in place, assuming that the value s_{t+1} is a good estimate of the value upon which the new value for state s_t can be based. By using Temporal Difference Learning, the value function $V(s_t)$ gets updated iteratively and will converge to its true value when the number of iterations approaches ∞.

2.5.8 Q-Learning

An algorithm that aims to learn the Q-function of a certain problem is Q-learning, presented in Watkins (1989). This method defines the Q-value in terms of the reward obtained by executing the action and the future reward obtained by executing the action with the highest Q-value in future states. Using the $TD(0)$ rule, the Q-value can be trained iteratively according to the following function for the TD-error:

$$\delta_{Q(s_t,a_t)} = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \tag{2.6}$$

This TD-error can then be used to update the value of taking action a_t in state s_t:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_{Q(s_t,a_t)} \tag{2.7}$$

Where α is once again a learning rate to control the size of the updates performed to the Q-function.

2.5.9 SARSA

A modification of Q-learning was presented in Rummery & Niranjan (1994) and later dubbed SARSA in Sutton (1996). SARSA focuses on the transition from a state-action pair to the next state-action pair while obtaining a reward in the process. The resulting sequence results in the name SARSA: $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$. The difference between SARSA and Q-learning is that SARSA does not build on the action in state s_{t+1} with the highest Q-value, but instead uses the action in state s_{t+1} that is selected by π. This could either be an explorative or a greedy action.

2.5.10 Actor-Critic Systems

The methods described above depend on either the values obtained from the value function $V(s_t)$ or on the Q-values obtained from the Q-function $Q(s_t, a_t)$ to select the next action to take. By updating the value of the state or state-action pair, the policy may or may not select a different action when it encounters the same state again. A different approach is explored in Barto et al. (1983) that has a separate structure implementing the policy π, the so-called *actor*. The actions taken by the actor are evaluated by the *critic*, which represents the value function. This means that while the actor should be influenced by the values generated by the critic, it does not necessarily use the value assigned by the critic to select the next action. This separation has advantages in situations where the action space is extremely large or continuous as selecting an action does not necessarily involve evaluating the expected rewards of all possible actions but instead depends on the precise implementation of the actor. A paper by Peters & Schaal (2008a) describes an actor-critic system using a gradient update method called *natural actor critic*. They show that the traditional actor-critic is a form of a natural actor-critic.

2.5.11 Continuous Spaces

Discrete state and action spaces allow to implement the value and transition functions as lookup tables from which the state or state-action values can be obtained and updated. Many problems are not discrete however, and the number of states is usually extremely large. The value function can be regarded a function that maps a set of numerical features to a value for that state. The true structure of this function is almost always unknown. Any function can be approximated using a function approximator (FA) that attempts to learn the patterns in the input data to correctly predict the corresponding output. Examples of function approximators are Artificial Neural Networks and decision trees. These FAs take the continuous input and use it to generate an estimate of the correct output. The TD-error can then be used to update the function approximator after each action.

2.5.12 CACLA

Actor-critic systems can also be implemented for continuous state and action spaces. A function approximator can be used for both the actor and the critic, where the critic outputs a single value evaluating the current state, while the actor yields a set of values representing the action to take. One implementation of an actor-critic system in continuous spaces is CACLA (Van Hasselt & Wiering, 2007), the Continuous Actor Critic Learning Automaton. The main feature that distinguishes CACLA from conventional actor-critic systems is that the update to the actor is equal regardless of the size of the TD-error. Instead, the TD-error is used only for the decision whether to reinforce the actor or not. If $\delta_{V(s_t)} > 0$ then the action generated by the actor is reinforced, otherwise it is not. The claim is that this makes the algorithm less sensitive to scaling of the value landscape and also makes it easier to overcome plateaus in the value function: if the value only increases very slightly, the actor will still be updated to select that action. One variant of CACLA, also presented in Van Hasselt & Wiering (2007), does take the TD-error into account to some extent. This variant is called CACLA+Var. A running average is maintained of the TD-error over all states:

$$var_{t+1} = (1 - \beta)var_t + \beta\delta_t^2 \tag{2.8}$$

After each TD-error of an action has been calculated, the number of times that action is reinforced is determined by the TD-error divided by the standard deviation of the TD-error of earlier actions. This requires a starting value for var_0 and a value for β, where var_0 should not be too low to avoid updating often early in learning, when var_0 is still a large factor in var_t. In Van Hasselt & Wiering (2007), CACLA was compared to two other algorithms that deal with continuous state and action spaces: Wire Fitting (Baird et al., 1993) and Gradient Ascent on the Value (Prokhorov & Wunsch, 1997) on a tracking task where the agent has to follow a moving target and a Cart Pole Balancing task where the agent has to apply force to a cart with a pole attached to it which must remain upright. For both problems, CACLA and CACLA+Var performed well, and CACLA+Var showed quicker converging than regular CACLA. When Gaussian noise was used as exploration strategy, it outperformed all other algorithms while it performed roughly equal to Wire Fitting in the case of ϵ-*greedy* exploration. These good results motivated the application of CACLA+Var to the higher dimensional problem of controlling a robot arm to grasp an object.

2.6 Parameters

The CACLA algorithm has a number of parameters. The first one is the discount factor $0 \leq \gamma \leq 1$ which indicates how important the future is. The larger this value, the more the system looks ahead to future rewards. If this value is small, only rewards in the very

near future are considered relevant. In practice, varying this parameter does not affect the outcome much, but since the goal of the project is to reach a goal situation, the future is deemed rather important. An estimate of a good value for γ can be calculated based on the average time expected to reach the goal. Based on the demonstrations of grabbing poses that were recorded as part of this project, the average duration of each demonstration was 25 seconds. While the demonstrations were performed rather slowly to generate more fine-grained data, learning algorithms may need more time to explore the action space. Therefore, an upper limit of 60 seconds was used. Because a frequency of 10Hz was used in the robot architecture, 60 seconds correspond to 600 time steps, so a reward received 600 steps in the future must still be relevant in the current state. As a baseline a factor of 0.1 for a state 600 time steps in the future is used, resulting in $\gamma^{600} \approx 0.1$, or $\gamma = .1^{1/600} \approx 0.9962$.

Also, two learning rates are required, one for the critic, α, and one for the actor β. These values should be small to avoid overshooting on the correct values when updating the networks. Therefore, an initial value of 0.05 was used for both networks. The learning rates were updated by multiplying with a factor 0.999 after each episode to make them decay over time. This allows the system to learn quickly in the beginning but converge to an optimal solution over time. These values were determined by many intermediate experiments with various values for these parameters. Larger initial values for the learning rate or lower decay of the learning rates often resulted in failure to reach a working solution to the problem, while lower initial learning rates require longer to train the system. A smaller decay factor will result in the learning rates converging to 0 too soon, stopping learning completely in an early phase.

When using Gaussian noise, a standard deviation $\sigma_{exploration}$ is required to sample from the Gaussian distribution. To decrease exploration when training progresses, a large initial value of $\sigma_{exploration} = 2$ was used, which decayed over time. After each grasping attempt, $\sigma_{exploration}$ was reduced:

$$\sigma_{exploration} \leftarrow \sigma_{exploration} * 0.999 \tag{2.9}$$

After several hundred episodes of training this value will slowly decrease to 0, but since the output of an ANN is deterministic, it is useful to keep some noise on the output to keep the system dynamic. Therefore, a lower bound of 0.2 was enforced on $\sigma_{exploration}$.

Two more parameters must be set for the CACLA+Var algorithm: the initial value for the moving variance var_0 and the factor β with which new values for the TD-error are incorporated into the moving variance. Since the system will receive the same reward in most non-final states, the size of the TD-error will be small. An initial value of $var_0 = 1.0$ makes sure that initially most TD-errors will be smaller than the moving

variance and will result in just one update of the actor. A value of $\beta = 0.001$ was used, as this will result in approximately halving the influence of any value for the moving variance in around 600 time steps, or 60 seconds, the maximum duration of time steps enforced in the final experiments. Higher values of β led to too much fluctuation in the moving variance and lower variance makes the moving variance stabilize too much over time, resulting in hardly any additional updates at all.

To stimulate the robustness of the system, an additional layer of noise was added at the final output of the algorithm. The algorithm never knows about this noise and therefore never knows the actual command executed. This noise was also Gaussian noise with a standard deviation of $\sigma_{noise} = 0.01$ during training. The experiments were performed with two different values for σ_{noise}: 0.01 and 0.05, to test the robustness against noise of the system.

2.7 Reward Function

The value function given in equation 2.3 can be rewritten as the sum of a geometric series:

$$V(t) = \gamma^0 r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + ... \tag{2.10}$$

FANN, the function approximator that was used for the experiments has a linear output function that is limited to the interval $(-300, 300)$. This information can be used to set the maximum reward that can be given in each state to avoid saturating the value function by calculating the sum of the infinite geometric series:

$$\sum_{k=0}^{\infty} \gamma^k a = \frac{a}{1-\gamma} \tag{2.11}$$

with a as the maximum reward. For this project, $\gamma = 0.9962$ was used as is explained in section 2.6. Using this formula and the value for γ, a can be calculated:

$$a = 300 * (1 - \gamma) = 1.14 \tag{2.12}$$

So, any reward in a non-final state should not lie outside the interval $(-1.14, 1.14)$.

To keep the implementation simple, a simple reward function is required that formulates the desired behavior with as few separate cases as possible. For the problem at hand, negative situations are collisions with the environment or the robot itself and positive situations are reaching the goal as quickly as possible without hitting anything. Therefore, a large positive reward is awarded when the robot arms reach their goal. A

negative reward is given when the robot collides with its environment or its own body. This is decided by performing collision detection between the bounding box of the robot itself, the surface supporting the object and the position of the hands of the robot.

Furthermore, for each time step a negative reward of $-0.3a$ is given, with a the maximum reward at each time step, motivating the system to find the shortest possible path to the goal. Finally, a reward of 300 is given for reaching the goal location which is having both hands touching the object, with the hands at roughly opposing sides of the object.

After this situation has been reached, the robot attempts to actually pick up the object by moving the hands closer to each other and then moving them upwards.

2.8 Performance Evaluation

The system was evaluated by attempting to grab objects from a selected set of different objects from various locations and orientations. The evaluation will take place after the algorithms have been trained in simulation on randomly generated objects of varying dimensions and positions.

Evaluation consists of moving the hands together and upwards. If the location of the hands was correct, the object should be fixed between the two arms and the upward movement should have lifted the object. During training, the object was simulated and just having the hands in the correct position was used to determine success. In the final evaluation on the real robot, success was determined by a human observing if the robot has correctly picked up the object.

2.9 The Reality Gap - Evaluation on the Real NAO

As robots are expensive and limited in processing capacity, often researchers rely on simulations of the world and the robot to perform their experiments. However, whatever works in simulations usually will not work without modifications when using a real robot in the real world. This is called the reality gap (Jakobi et al., 1995). The tremendous advantages of simulations however can be of great help to validate new approaches to robot learning problems. This research is no exception to that rule. In order to evaluate many different parameter settings and conditions, experiments will have to be run on a simulated robot that makes it possible to evaluate the performance without deploying it on a real robot. For the experiments in this research, the best performing algorithms will be evaluated on a real robot with the best performing setups for these

Figure 2.2: The objects the algorithms were tested on: a cup, a can, a bottle, a package of Pringles, a stuffed toy, a carton of milk, a computer fan and a coffee pad tin.

algorithms. The simulator models the object to grasp as a box with a random width between 8 and 10 centimeters and a height between 8 and 10 centimeter, randomly placed in front of the robot, at a random height between 34 and 37 centimeter. Theses values agree with the capabilities of the NAO: it cannot handle much larger and thus heavier objects, and smaller objects are more effectively handled using one hand, which is not the focus of this research.

Performance on the real robot will be tested on a set of 8 different objects of varying sizes and shapes: a small disposable cardboard cup, a 330 ml soda can, a 0.5 liter plastic bottle, a package of Pringles, a stuffed toy, a carton of milk, a computer fan and a coffee pad tin. For each object, 10 attempts will be made to see if the robot is able to pick up the object. To be able to localize the object, the robot will be trained to recognize the object based on SURF features, using 5 training images per object. The set of objects is shown in figure 2.2. Using this dataset, the robot was able to easily recognize the cup, the can, the bottle, the Pringles and the milk carton. Recognizing the stuffed toy, the fan and the coffee pad tin was more difficult during testing. When it failed to recognize these objects, a small rotation of the object was sufficient to get the correct angle and lighting conditions in order for the robot to recognize the object and start its grasping attempt.

Chapter 3

Hard- and Software

The project was run on a PC and an Aldebaran NAO humanoid robot, using a software architecture designed for a robot participating in the RoboCup@Home competitions. In the following sections, details about the hard- and software used in the system will be presented.

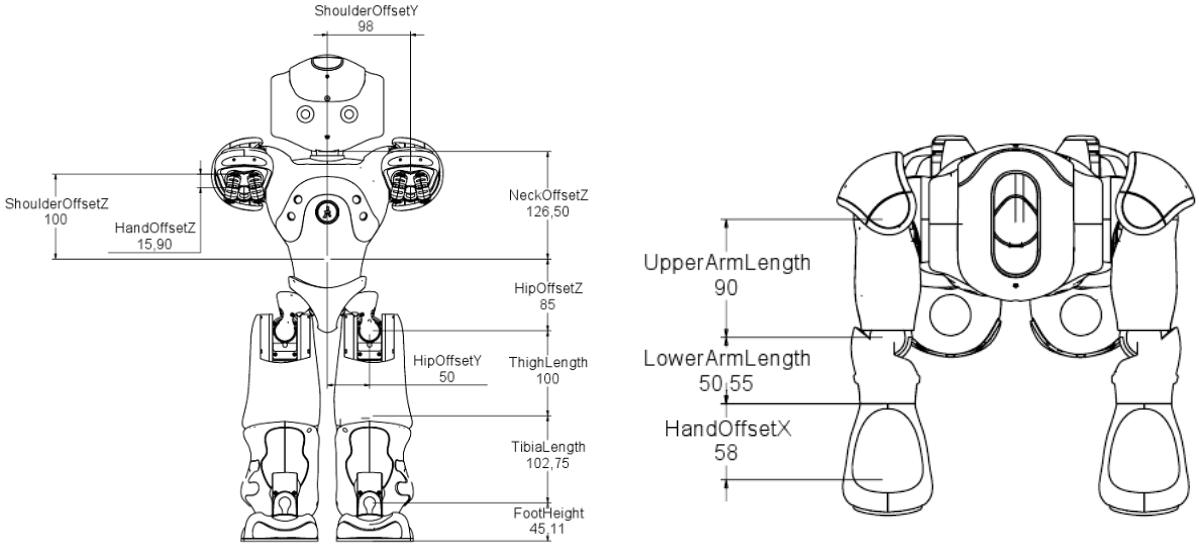
3.1 Hardware

The experiments were performed on an Aldebaran NAO humanoid robot, version 3.2. The robot is 573.2 mm high and 273.3 mm wide. From scapula to the end of the hand, the length of the arms is 290 mm. Detailed information about the geometry of the NAO can be found in image 3.1.

The training of the robot was performed using the actual dimensions of the robot. Therefore, the resulting methods will not be immediately usable on a different version of NAO or on a different humanoid robot. However, the dimensions are parameters to the training algorithms and in no way hard coded. Therefore, repeating the experiment using another robot should be fairly straightforward and require few modifications.

For image recognition, the two cameras inside NAO's head were used. These cameras are VGA cameras producing a video stream of up to 30 Hz. Both cameras have a horizontal field of view (FOV) of 47.8 degrees and a vertical FOV of 36.8 degrees and use the YUV422 color space. However, hardware and processing limitations imply that the full quality and frame rate of the cameras could not be used. Instead, the video stream was downscaled to 160x120 with 10 frames per second. The cameras have a fixed focus and have a focus range of 30 centimeters to infinity [1]. The placement of the two cameras

[1]The minimum focus range of 30 cm advertised for the camera does not not pose any problems in practice, even though the distance towards the object is usually less than 30 cm: the object remained in focus during the experiments



(a) Front view (b) Top view

Figure 3.1: Geometry of the NAO.

Source (05-03-2012): www.aldebaran-robotics.com/documentation/nao/hardware/kinematics/nao-links-32.html

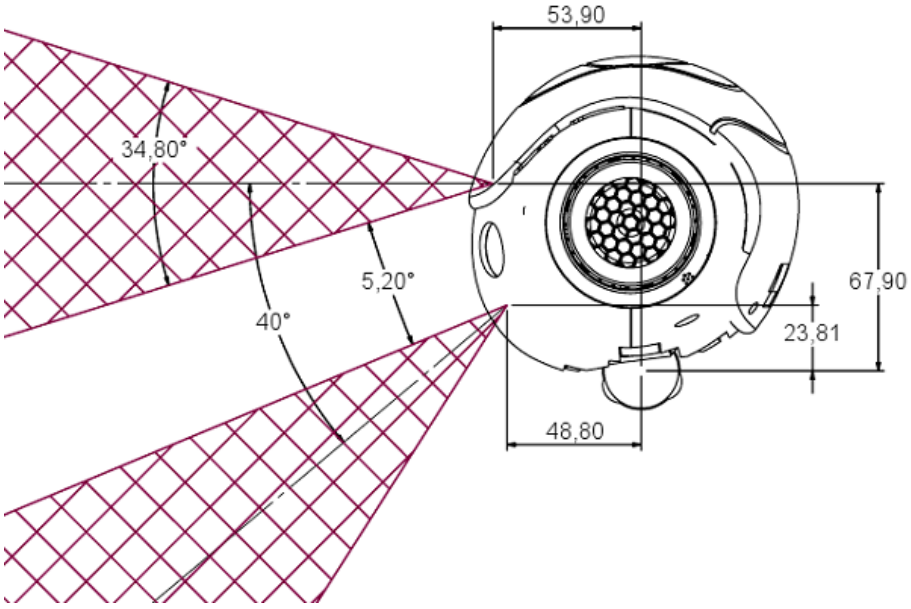


Figure 3.2: The placement of the cameras in NAO's head.

Source (05-03-2012): www.aldebaran-robotics.com/documentation/nao/hardware/video.html

is such that they do not overlap, there is a gap of 5.2 degrees between the two fields of view. The first camera is facing straight forward and the second camera faces down 40 degrees. See image 3.2 for more details.

The NAO is equipped with a 500 MHz x86 processor from Advanced Micro Devices, Inc. Because of the limited processing capacity of this processor, all data processing was performed on a separate desktop PC with an Intel Core i5 processor, running a 32-bits version of Debian GNU/Linux. The NAO collects all the data and transmits this over a wired network connection to the PC, which processes the data and sends new commands to the NAO.

3.2 Operating System and libraries

The NAO was running the Aldebaran NAO Software version 1.10.52 during the experiments.

The software architecture used runs on a 32 bits version of Debian GNU/Linux. The architecture itself was written in Python and is targeted at Python version 2.6, but will run in Python 2.7. For image processing the OpenCV library [2] was used. The artificial neural networks that were used in the system were implemented using FANN, Fast Artificial Neural Network [3]. For the generation of random numbers and Gaussian noise, the NumPy package [4] was used which provides many strongly optimized numerical methods for use in Python.

3.3 Robot Software Architecture

The system was deployed on a behavior based architecture (Arkin, 1998) , developed by the BORG team [5], a robotics team of the University of Groningen competing in the RoboCup@Home competitions (Wisspeintner et al., 2009). The architecture was written in Python. This section will describe the design of this architecture.

The architecture consists of one central core, the Brain, and a set of modules performing data acquisition, which can be run on machines separate from the one running

[2] opencv.willowgarage.com/
[3] leenissen.dk/fann/wp/
[4] numpy.scipy.org/
[5] www.ai.rug.nl/crl/

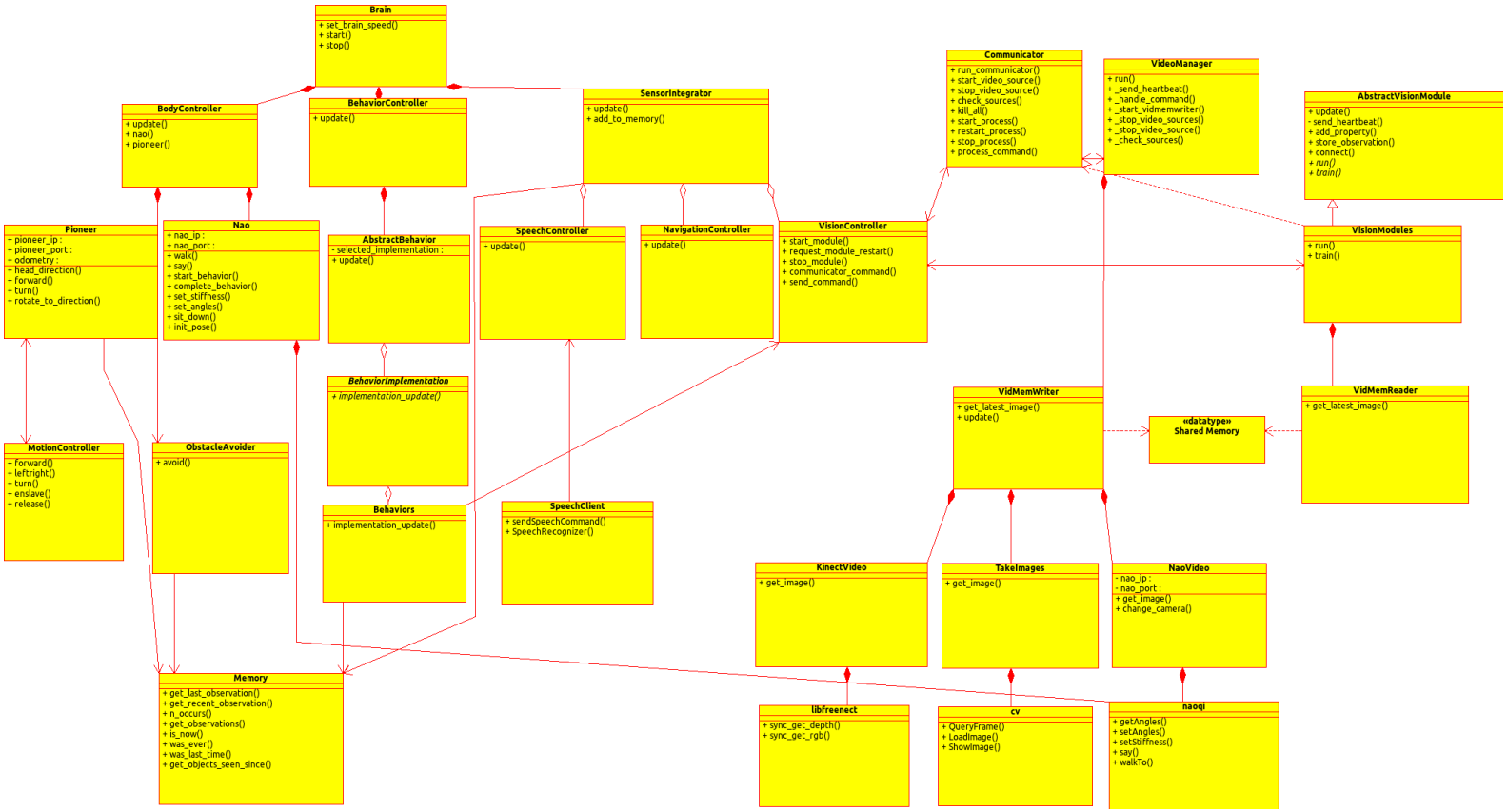


Figure 3.3: An overview of the software architecture.

the Brain. The following subsections will provide more information about how they operate and interact. An overview of the system can be seen in figure 3.3.

3.3.1 Main Software: The Brain

The Brain itself consists of a set of controllers, each controlling a part of the whole system and a central storage facility, the memory. The task of the Brain is to initialize the system, read the configuration and run the system at a set frequency. After setup, the Brain runs its main loop at a specified frequency, requesting updates from each controller sequentially. We generally use a frequency of 10Hz, and this value was also used in this research.

The Memory

The Memory is a central storage mechanism where all parts of the Brain can store and retrieve information. Each entry in the memory has a name in order to categorize the information available. Specific functions are available to retrieve the most relevant information necessary, such as obtaining the last observation of one specific category of information, or checking whether one type of observation has occurred at all or in the recent past. A typical query would be:

```
memory.is_now('person_visible', {'name': 'John'})
```

which would evaluate to True if a person is currently visible which is recognized as John. Similarly, is_now could be replaced with was_ever to check if the robot has ever seen John for as long as it has been running.

Body Controller

The Body Controller is in control of the body of the robot. In this specific project, this contains the modules connecting to and controlling the NAO. It reads out sensor values from the NAO's joints and actuates the motors. It connects to the NAO using the SDK supplied by Aldebaran, NAOqi.

Our body controller has support for other robots, such as ActivMedia's Pioneer robots, but these were not used in this project.

Sensor Integrator

The Sensor Integrator is a controller that collects data from the external sensors, such as cameras, microphones, sonars and other equipment. It mainly consists of a networking

architecture setting up communication with the external modules to send commands and receive observations from them. All the data obtained from the external modules is gathered and stored in the Memory, to be used by behaviors.

Behavior Controller

The Behavior Controller is where the behavioral logic takes place. It selects the behavior to run based on the configuration or the current state of the robot, such as the presence or absence of certain stimuli. The selection of behaviors is realized through the use of pre-conditions and post-conditions. These conditions are evaluated by the Memory to check if they are true, in the way described above. As soon as a pre-condition becomes true, the behavior controller will activate the behavior. As soon as the post-condition becomes true, the behavior controller will terminate the behavior.

There are high-level behaviors and low-level behaviors. The low level behaviors will actually actuate the robot and use the readings provided by the Sensor Integrator. An example could be a behavior to find an object in the room. Such a module would use the readings from a module recognizing objects to see if the object is currently in view, and if it is not, it could move the robot around until it is in view. High-level behaviors will not use such direct actions, but will use sub-behaviors to perform the required action. A behavior to serve drinks to a guest could, for example, consist of a behavior to find the guest, a behavior to ask the guest which drink the guest wants, a behavior to find the kitchen, a behavior to pick up the drink that the guest wants, a behavior to navigate back to the person and a behavior to give the drink to the guest. Each of these sub-behaviors would encapsulate the necessary low-level actions to accomplish the task at hand. The high-level behavior's job is then to make sure that each sub-behavior completes successfully, and if it does not, find a way to work around the problem or report back to the user that the task was impossible.

3.3.2 Data Acquisition Modules

The robot architecture was developed to obtain maximum performance on a distributed system. Therefore, each of the modules can be run on any machine that is accessible through a network by the Brain. This allows for easy scalability: if more processing power is required, one could easily add a new PC or laptop to the system. Because the data acquisition is usually very demanding on processing power while the behavior system is not, the system is able to utilize all available resources optimally.

The modules are configured in the configuration of the Brain and started by the Sensor Integrator. If any module fails for some reason, it is automatically terminated and

restarted. This greatly improves the overall stability of the system. Also, multiple instances of the same module can be run using different parameters to tune the results for some specific task. For example, two instances of an object recognizer could be run, each trained on a different set of objects.

The modules most directly interact with the sensors, such as the 3D sensor, sonars, video cameras or microphones. They also process the data to perform object or speech recognition. While most modules of this system are also written in Python this is not a strict requirement and we do have other modules written in Matlab, Java and C++. For example, our speech recognition is in JAVA. We have a person tracking module using Matlab and we handle point clouds from our 3D sensor using a C++ library.

For this project, one module was used to capture the video stream from the NAO and localize the object to grab in the image. To accomplish this, the module extracts SURF features (Bay et al., 2006) from the image and matches them with the features calculated on a set of objects trained on. If an object is recognized, the module sends the observation and the position in the image to the grabbing module which then attempts to grab it.

Chapter 4

Results and Discussions

This chapter will describe the experiments and report on their results. Various algorithms are compared. First, all algorithms are trained if necessary, and then evaluated for 250 trials on randomly generated objects in simulation. Afterwards, the algorithms that performed adequately in simulation were tested on the real robot, for 80 trials in total, or ten trials for each of a set of eight objects.

Five different algorithms are compared in the following sections. The five algorithms are:

1. 1-Nearest Neighbor on the dataset
2. 3-Nearest Neighbor on the dataset
3. Learning from Demonstration using an Artificial Neural Network
4. CACLA+Var with random initial actor
5. CACLA+Var with actor pre-trained on dataset

Each of these algorithms has parameters. For 1-NN and 3-NN, the most important factor is the value for K, which is tested at $K = 1$ and $K = 3$ as the names indicate. For the other settings, the *autotuning* option of FLANN was used which automatically tunes the behavior of the algorithm to the provided dataset. The parameters for the ANN used in Learning From Demonstration are summarized in table 4.1. The parameters for CACLA+Var algorithm are summarized in table 4.2. For CACLA+Var with a pre-trained actor, the actor network, originating from the learning from demonstration using an ANN, was updated to match the parameter settings listed in table 4.2. It is important to note that for all experiments, the system was put into testing mode. This means that no learning occurred, and all updates to the system were disabled. Since CACLA+Var is the only of the algorithms tested that uses on-line training, this does not

Parameter	Value
Inputs	17
Hidden Units	200
Outputs	5 or 10
Learning rate	0.0001
Activation function	Symmetric Sigmoid
Training method	Back-propagation
Initial weights	(-0.1, 0.1)

Table 4.1: Parameters for the Artificial Neural Network used for Learning from Demonstration. The values for these parameters have been discussed in section 2.3.

Parameter	Value
Inputs	17
Hidden units	200
Outputs Critic	1
Outputs Actor	5 or 10
Critic Activation Function for Hidden Units	Symmetric Sigmoid
Critic Activation Function for Output	Linear $\in$ (-300, 300)
Actor Activation Function for Hidden Units	Symmetric Sigmoid
Actor Activation Function for Output	Symmetric Sigmoid
Actor Learning Rate	Initial: 0.05 Min: 0.001
Critic Learning Rate	Initial: 0.05 Min: 0.001
Learning rate discount factor, applied after each episode	0.999
Discount Factor	0.9962
$\sigma_{exploration}$	Initial: 2.00 Min: 0.2
Exploration discount factor, applied after each episode	0.999
β for updating variance	0.001
var_0 for initial variance	1.0
Number of training episodes	1500
Reward for reaching goal	300
Reward per time step	-0.342
Reward for collisions	-1.14

Table 4.2: Parameters for CACLA+Var. The values mentioned for the parameters here have been discussed in section 2.6.

affect the other algorithms. For CACLA+Var this means that each subsequent trial is not dependent on the previous trial. If learning were enabled, one trial might lead to updates of the system that influence the next trial. The same holds for the experiments on the real robot.

For this research, experiments were performed using four different setups:

1. control both arms, using absolute angles
2. control both arms, using relative angles
3. control the left arm and have the right arm mirror the left arm, using absolute angles
4. control the left arm and have the right arm mirror the left arm, using relative angles

Each of the following sections will describe the results of one of the five algorithms on the four different setups. Different numbers of runs were used for the algorithms, because they differ in their approach and requirements.

For learning from demonstration, training uses the same data set for all runs but this data set is split randomly into a training set and a test set. To achieve significant results, several different distributions over the training set and the test set must be used to test all the data in many different combinations. 20 independent trainings on this dataset were performed for each of the four setups. Additionally, the twenty trained networks were combined in sets of four to form an ensemble which was also evaluated for 250 grasping trials. These five additional trained systems resulted in a total of 25 different systems trained using learning from demonstration.

For Nearest Neighbor, no training is required. Therefore, the results of each run should be nearly identical for the same settings, only depending on the randomly generated objects. Therefore, less variation in the results was expected and therefore just five runs of 250 grabbing attempts were evaluated per setup.

CACLA+Var has more random factors that are different for each run of the algorithm. These additional random factors are the initial weights of the actor and critic neural networks and the random exploration. Therefore, a larger number of runs is required to average over these factors and increase the statistical significance of the experiments. For this reason, twelve independent runs of CACLA+Var were executed and each resulting system was evaluated for 250 grasping attempts.

For CACLA+Var with a pre-trained actor, the ANNs trained for the evaluation of Learning from Demonstration were used. The four trained networks per PC were combined in an ensemble and used in that way as the actor of the system. Because the training then still relies on the random exploration, the system was started four times with the same pre-trained ensemble to average over this. Since 20 networks were available, this resulted in 20 evaluations of the pre-trained CACLA+Var.

Each trained algorithm was evaluated 250 times on a randomly generated object in simulation, between $8cm$ and $10cm$ in width and height, located on a centered position in front of the robot, between $34cm$ and $37cm$ high and $16.5cm$ in front of the robot. The histograms list the number of time steps required until the goal state was reached. Time steps correspond to actions for the robot. When running on the robot, each second is split in ten time steps. In simulation, up to 450 time steps were evaluated every second.

The statistical details of the various experiments are presented in table 4.3 for control using absolute angles and table 4.4 for control using relative angles. The table lists the mean μ, the standard deviation σ and the upper and lower bounds of the 95% confidence interval for each of the experiments. The numbers refer to the success rate of the algorithms, so that 1 means 100% success and 0 means 0% success. All algorithms listed were ran N times, each time attempting to grab an object 250 times, and the success rate listed in the table is the average of the results of each of those runs.

4.1 Nearest Neighbor Results

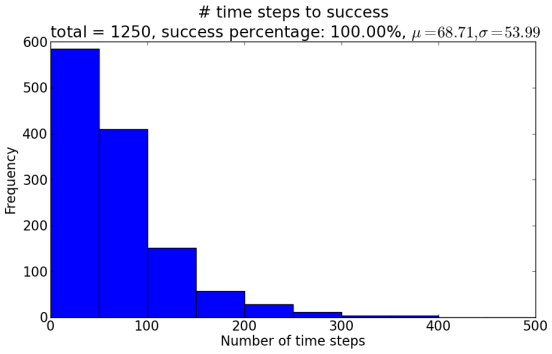
In these simulations, 3-Nearest Neighbor and 1-Nearest Neighbor give similar performance in four of the setups, while 3-Nearest Neighbor performs a lot better in three of the remaining four setups, and 1-Nearest Neighbor performs better in the last setup. Here, only the graphs for 3-NN are shown, but the graphs for 1-NN show similar shapes and numbers. The results of 3-Nearest Neighbor Regression when controlling one arm using absolute angles in simulation are shown in figure 4.1. The first histogram shows the distribution of the number of time steps required to reach the goal state with $\sigma_{noise} = 0.05$ while the second histogram shows the same information when using $\sigma_{noise} = 0.01$. In both cases, each attempt resulted in reaching the goal state. Figure 4.3 shows the results when using relative angles for the same two values for σ_{noise}. When using relative angles, this resulted in 118 successes for $\sigma_{noise} = 0.05$ and 541 successes for $\sigma_{noise} = 0.01$. Similar figures for when controlling both arms can be seen in figure 4.2 for absolute angles (885 and 976 successes for both noise levels) and figure 4.4 for relative angles (11 and 1 successes for both noise levels).

Algorithm	Arms	σ_{noise}	N	μ	σ	95% CI LB	95% CI UB
CACLA	Left	0.01	12	91.4%	27.0 pp	74.4%	100.0%
CACLA Pre-trained	Left	0.01	20	2.5%	1.1 pp	2.0%	3.0%
LFD	Left	0.01	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Left	0.01	5	100.0%	0.0 pp	100.0%	100.0%
3-NN	Left	0.01	5	100.0%	0.0 pp	100.0%	100.0%
CACLA	Left	0.05	12	91.8%	26.8 pp	74.9%	108.6%
CACLA Pre-trained	Left	0.05	20	4.5%	1.9 pp	3.6%	5.3%
LFD	Left	0.05	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Left	0.05	5	100.0%	0.0 pp	100.0%	100.0%
3-NN	Left	0.05	5	100.0%	0.0 pp	100.0%	100.0%
CACLA	Both	0.01	12	0.0%	0.0 pp	0.0%	0.0%
CACLA Pre-trained	Both	0.01	20	0.0%	0.0 pp	0.0%	0.0%
LFD	Both	0.01	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Both	0.01	5	42.2%	5.1 pp	36.4%	48.0%
3-NN	Both	0.01	5	78.2%	9.7 pp	67.0%	89.4%
CACLA	Both	0.05	12	0.0%	0.0 pp	0.0%	0.0%
CACLA Pre-trained	Both	0.05	20	0.0%	0.0 pp	0.0%	0.0%
LFD	Both	0.05	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Both	0.05	5	42.8%	5.8 pp	36.1%	49.5%
3-NN	Both	0.05	5	70.8%	12.5 pp	56.4%	85.2%

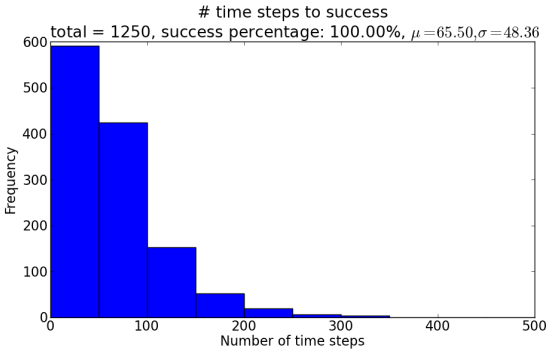
Table 4.3: Results of the algorithms using absolute angles to control the joints. CI LB and CI UB refer to the lower and upper bounds of the 95% confidence interval. μ and the upper and lower bounds express the success percentage on the trials and σ expresses the standard deviation of the success percentage in percentage points (pp).

Algorithm	Arms	σ_{noise}	N	μ	σ	95% CI LB	95% CI UB
CACLA	Left	0.01	12	76.0%	39.3 pp	51.3%	100.0%
CACLA Pre-trained	Left	0.01	20	0.1%	0.3 pp	0.0%	0.2%
LFD	Left	0.01	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Left	0.01	5	21.2%	2.0 pp	18.9%	23.5%
3-NN	Left	0.01	5	43.4%	6.5 pp	35.9%	50.9%
CACLA	Left	0.05	12	77.8%	38.8 pp	53.3%	100.0%
CACLA Pre-trained	Left	0.05	20	15.1%	1.9 pp	14.2%	16.0%
LFD	Left	0.05	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Left	0.05	5	30.0%	5.5 pp	23.7%	36.3%
3-NN	Left	0.05	5	9.6%	2.2 pp	7.0%	12.2%
CACLA	Both	0.01	12	0.0%	0.0 pp	0.0%	0.0%
CACLA Pre-trained	Both	0.01	20	0.0%	0.0 pp	0.0%	0.0%
LFD	Both	0.01	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Both	0.01	5	0.0%	0.0 pp	0.0%	0.0%
3-NN	Both	0.01	5	0.0%	0.0 pp	0.0%	0.0%
CACLA	Both	0.05	12	0.0%	0.0 pp	0.0%	0.0%
CACLA Pre-trained	Both	0.05	20	0.1%	0.2 pp	0.0%	0.2%
LFD	Both	0.05	25	0.0%	0.0 pp	0.0%	0.0%
1-NN	Both	0.05	5	0.0%	0.0 pp	0.0%	0.0%
3-NN	Both	0.05	5	0.8%	0.7 pp	0.0%	1.7%

Table 4.4: Results of the algorithms using relative angles to control the joints. CI LB and CI UB refer to the lower and upper bounds of the 95% confidence interval. μ and the upper and lower bounds express the success percentage on the trials and σ expresses the standard deviation of the success percentage in percentage points (pp).

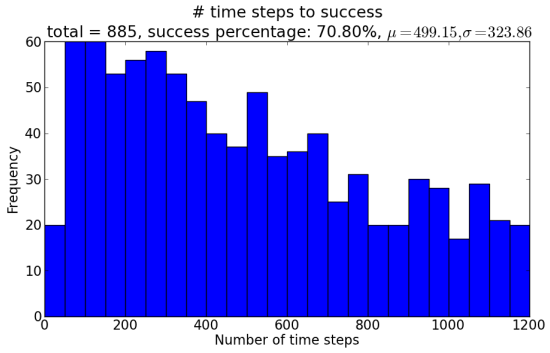


(a) $\sigma_{noise} = 0.05$

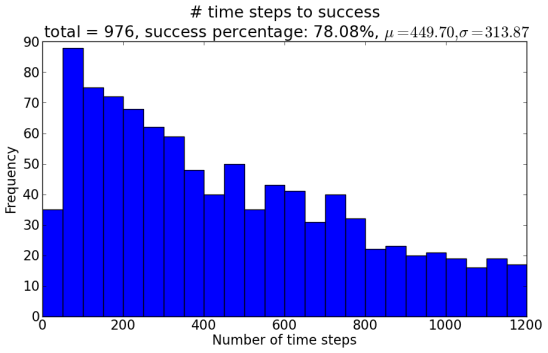


(b) $\sigma_{noise} = 0.01$

Figure 4.1: Results of 3-Nearest Neighbor Regression - Control of one arm with absolute angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better.

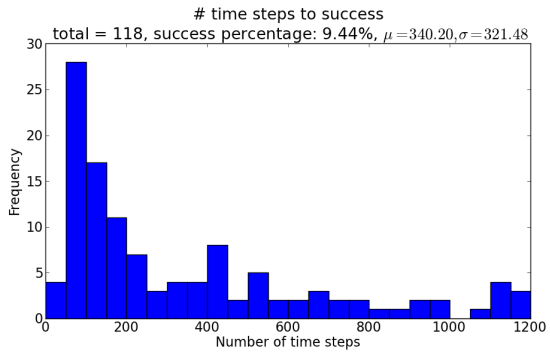


(a) $\sigma_{noise} = 0.05$

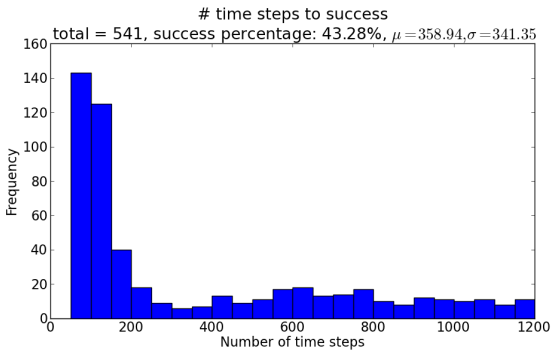


(b) $\sigma_{noise} = 0.01$

Figure 4.2: Results of 3-Nearest Neighbor Regression - Control of both arms with absolute angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better.

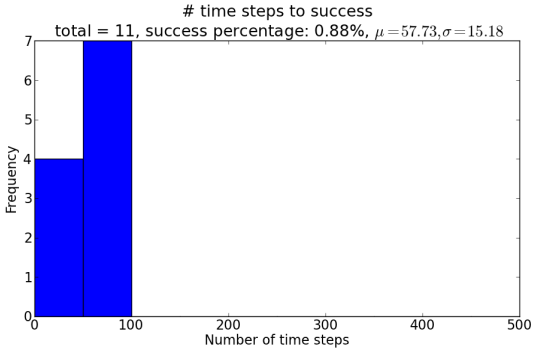


(a) $\sigma_{noise} = 0.05$

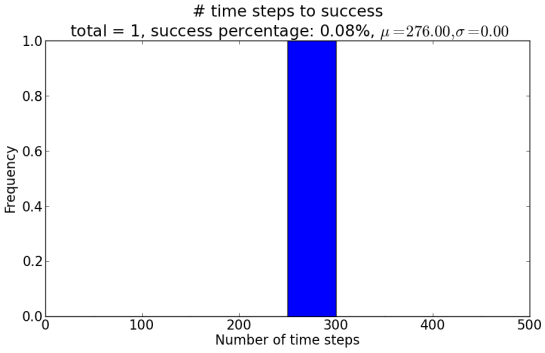


(b) $\sigma_{noise} = 0.01$

Figure 4.3: Results of 3-Nearest Neighbor Regression - Control of one arm with relative angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better.

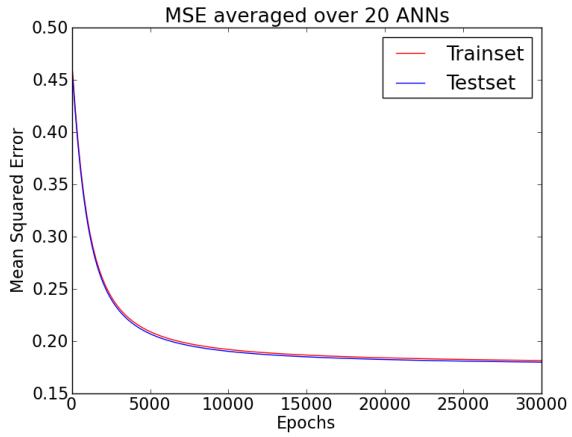


(a) $\sigma_{noise} = 0.05$

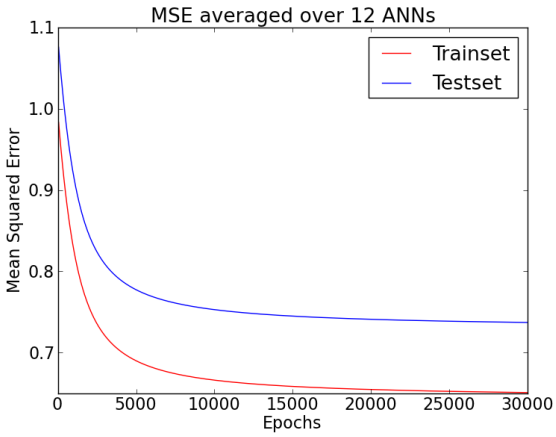


(b) $\sigma_{noise} = 0.01$

Figure 4.4: Results of 3-Nearest Neighbor Regression - Control of both arms with relative angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better. The surprising shape of the right graph is due to the fact that there was only one successful trial.



(a) Training both arms with absolute angles



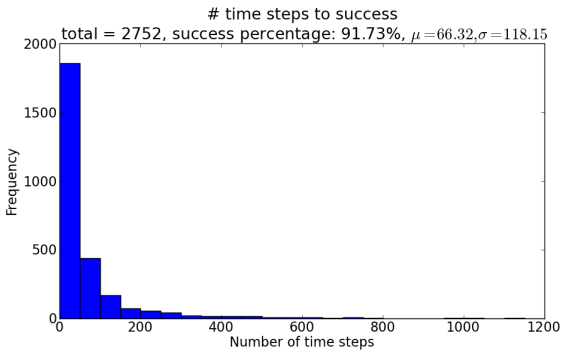
(b) Training one arm with relative angles

Figure 4.5: The MSE on the training set and the test set during training for 30000 epochs, for training on relative and absolute angles.

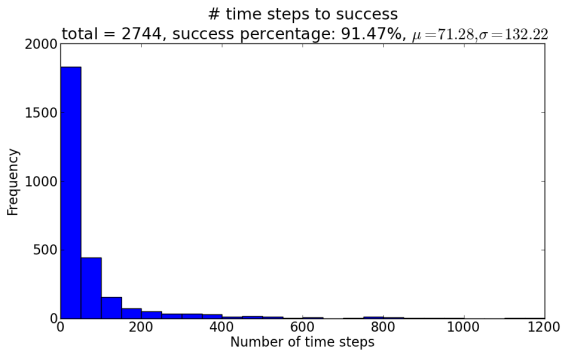
4.2 Learning from Demonstration Using an Artificial Neural Network

Figure 4.5a shows the Mean Squared Error (MSE) on both the training set and the test set during training when training to control both arms using absolute angles. Figure 4.5b shows the same statistics for training on controlling the left arm using relative angles. The graphs for controlling one arm with absolute angles or both arms using relative angles are not shown but show similar shapes and values. The values are averaged over 20 networks trained on the same dataset. However, the graph for the relative angles shows the average of 12 networks. 20 networks were trained on these datasets, but due to power failure, the training needed to be resumed for 8 networks training on relative values. This resulted in a different distribution of the data over the training set and the test set, which would distort the graph. Therefore the average of the MSE on these datasets was measured over the 12 networks that had no such interruption.

Testing any of the the trained networks for 250 trials on random objects failed to result in success every time, both for $\sigma_{noise} = 0.01$ and $\sigma_{noise} = 0.05$. The five ensembles formed, each using four of these ANNs together, also failed to result in success every time.

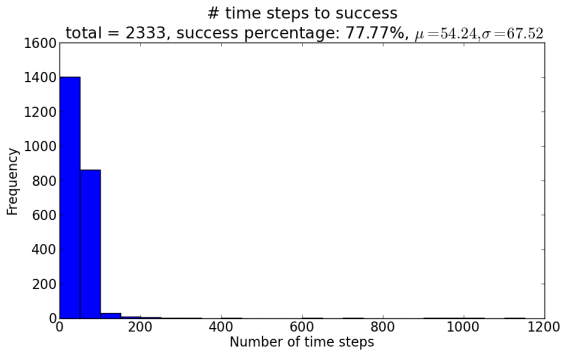


(a) $\sigma_{noise} = 0.05$

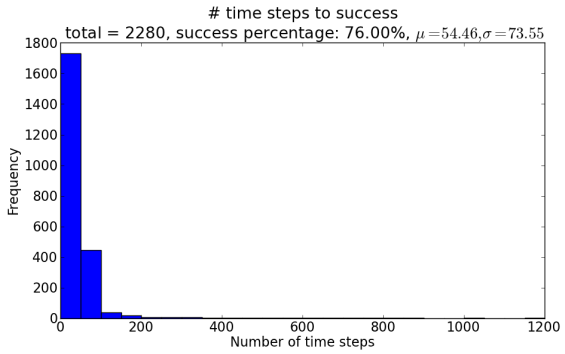


(b) $\sigma_{noise} = 0.01$

Figure 4.6: Results of CACLA+Var - Control of one arm with absolute angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better.



(a) $\sigma_{noise} = 0.05$



(b) $\sigma_{noise} = 0.01$

Figure 4.7: Results of CACLA+Var - Control of one arm with relative angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better.

4.3 CACLA+Var with Random Networks

Training CACLA+Var to control both hands simultaneously yielded no results when trying to pick up an object for 250 times, for none of the 24 trained systems. Therefore, their graphs are not shown. When training to control the left arm with the right arm mirroring the left, the system did manage to grab the object: for absolute angles 2744 times for $\sigma_{noise} = 0.01$ and 2752 times for $\sigma_{noise} = 0.05$. When outputting relative angles, the results were 2280 successes for $\sigma_{noise} = 0.01$ and 2333 successes for $\sigma_{noise} = 0.05$.

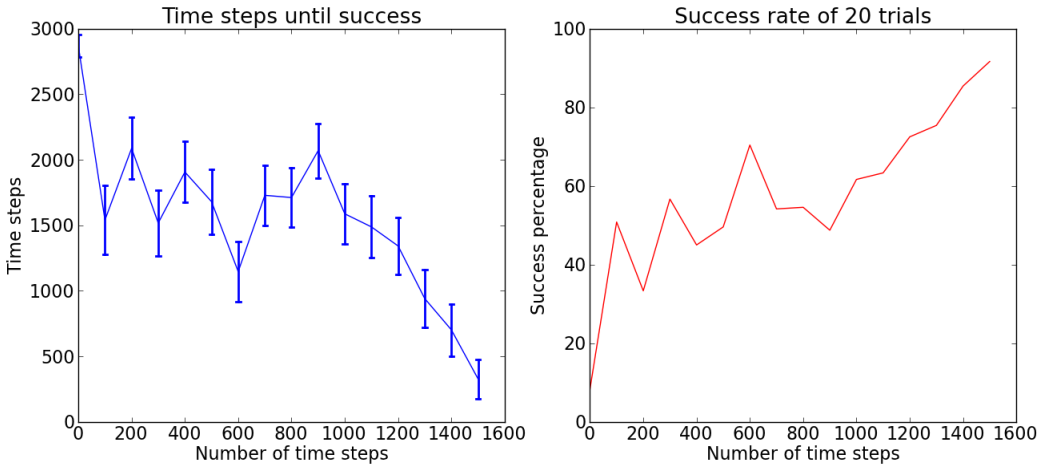


Figure 4.8: Performance during training of randomly initialized CACLA+Var - Control of one arm with absolute angles. The error bars show the 95% confidence interval. The left graph shows the average duration until success plotted against the number of training episodes - lower is better. The right graph shows the success rate when performing 20 trials after every 100 training episodes - higher is better.

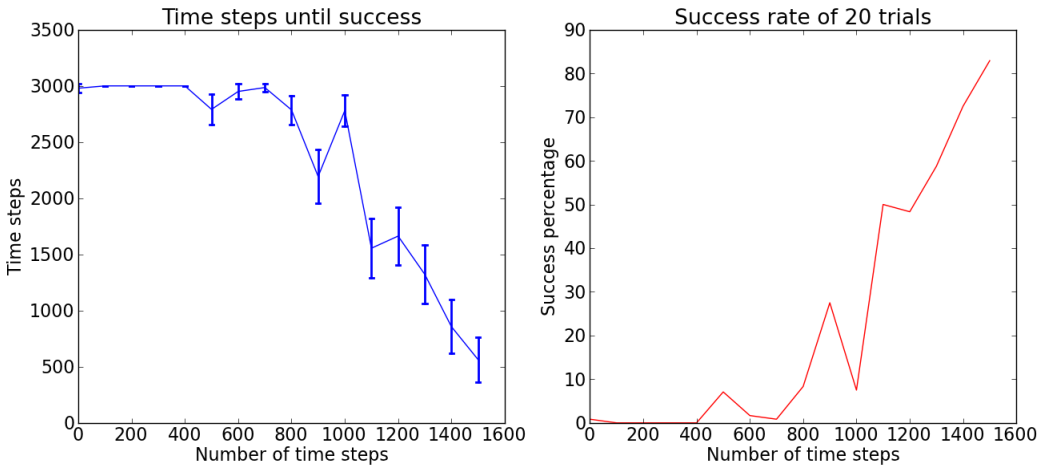
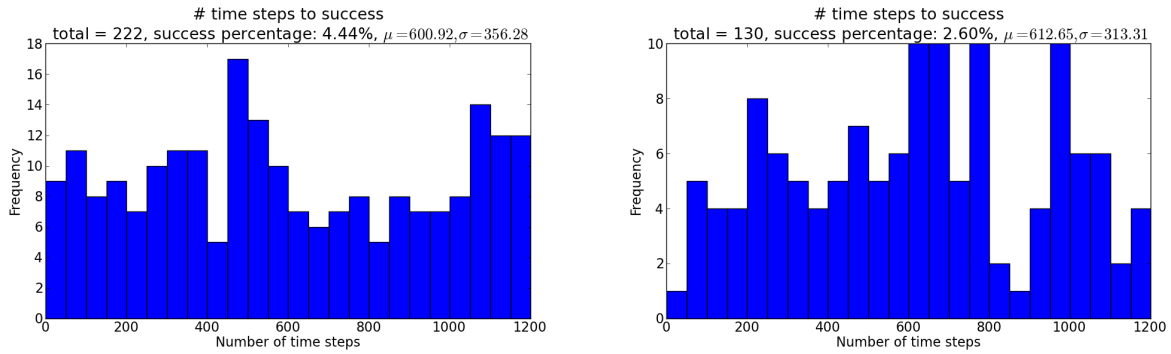


Figure 4.9: Performance during training of randomly initialized CACLA+Var - Control of one arm with relative angles. The error bars show the 95% confidence interval. The left graph shows the average duration until success plotted against the number of training episodes - lower is better. The right graph shows the success rate when performing 20 trials after every 100 training episodes - higher is better.



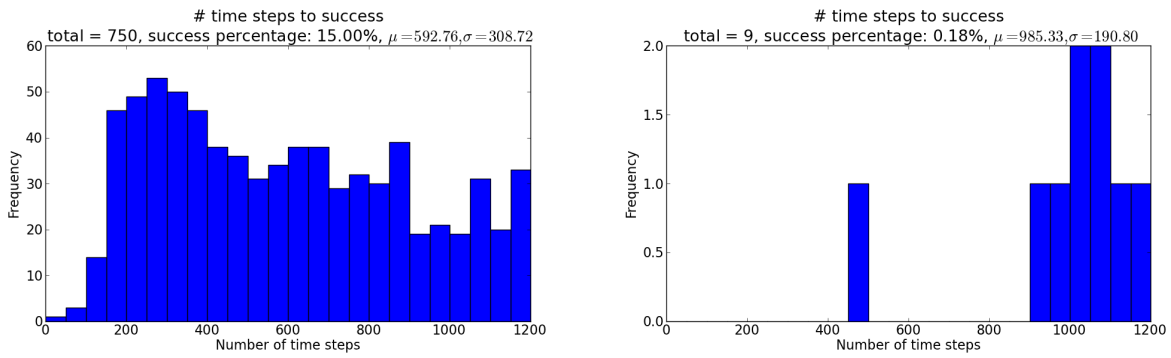
(a) $\sigma_{noise} = 0.05$ (b) $\sigma_{noise} = 0.01$

Figure 4.10: Results of pre-trained CACLA+Var - Control of one arm with absolute angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better.

The distribution of the duration of the successful attempts are shown in figure 4.6 for absolute angles and figure 4.7 for relative angles. During training, each algorithm was evaluated after every 100 episodes for 20 attempts to track the progress. The results of this can be seen for the absolute angles in figure 4.8, and for relative angles in figure 4.9. In these graphs, the average duration for all systems is averaged to generate the plot. The error bars show the 95% confidence interval for these values. Note that failures are represented as duration of 3000 time steps in these graphs, the maximum duration of each attempt used during training.

4.4 CACLA+Var with Pre-trained Networks

CACLA+Var was also evaluated when the actor was not randomly initialized but instead a pre-trained network that had been trained before on the dataset was used. Five different pre-trained actors were used for each set of inputs, where each one consisted of an ensemble of 4 pre-trained ANNs. Each pre-trained actor was used for 4 different runs of the CACLA+Var algorithm during 1500 episodes, giving 20 different CACLA+Var controllers. The value of 1500 was chosen because initial experiments showed that CACLA+Var will usually converge to a solution in under 1500 episodes. Here too, training on controlling both arms resulted in failure in almost all of the 5000 attempts over several runs. Therefore, just the results of the algorithms training to control one arm are shown. The distribution of the number of time steps until success when using absolute angles is shown in figure 4.10 and the distribution when using relative angles are shown in figure 4.11. During training, each algorithm was evaluated after every 100 episodes



(a) $\sigma_{noise} = 0.05$ (b) $\sigma_{noise} = 0.01$

Figure 4.11: Results of pre-trained CACLA+Var - Control of one arm with relative angles. The bars show the number of trials that resulted in a duration corresponding to the bar, with each bar covering a range of 50 time steps. A shorter duration is better.

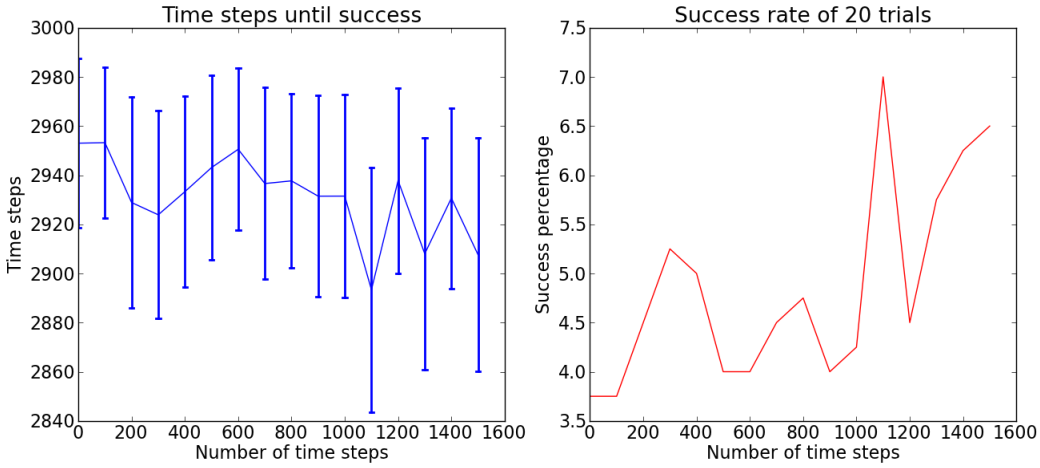


Figure 4.12: Performance during training of pre-trained CACLA+Var - Control of one arm with absolute angles. The error bars show the 95% confidence interval. The left graph shows the average duration until success plotted against the number of training episodes - lower is better. The right graph shows the success rate when performing 20 trials after every 100 training episodes - higher is better.

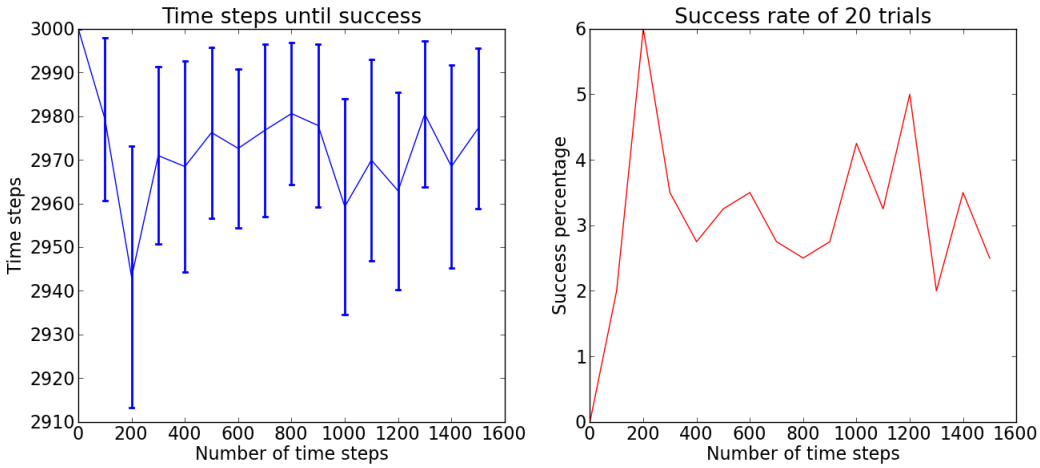


Figure 4.13: Performance during training of pre-trained CACLA+Var - Control of one arm with relative angles. The error bars show the 95% confidence interval. The left graph shows the average duration until success plotted against the number of training episodes - lower is better. The right graph shows the success rate when performing 20 trials after every 100 training episodes - higher is better.

for 20 attempts to track the progress. The results of this can be seen for the absolute angles in figure 4.12, and for relative angles in figure 4.13. In these graphs, the average duration for all systems is averaged to generate the plot. The error bars show the 95% confidence interval for these values. Note that failures are represented as duration of 3000 time steps in these graphs, the maximum duration of each attempt used during training.

4.5 Evaluation on the Real Robot

The two best performing algorithms were tested on a robot, which are clearly CACLA+Var and Nearest Neighbor. Since both 1-NN and 3-NN perform equally well when using absolute angles while mirroring the arms, and equally bad when using relative angles while controlling both arms, the decision which to use on the robot was based on the other two situations for both noise levels, $\sigma_{noise} = 0.01$ and $\sigma_{noise} = 0.05$. In these four situations, 3-NN outperforms 1-NN in three situations. Therefore, 3-NN was evaluated on the real robot.

Both algorithms were evaluated with both absolute and relative angles because in both these setups, the algorithms performed quite well in simulation. The algorithms were evaluated when controlling the left arm while the right arm mirrored the left arm,

	3-NN Mirror Relative			3-NN Mirror Absolute			CACLA Mirror Relative			CACLA Mirror Absolute		
Object	Score	Mean	Stddev	Score	Mean	Stddev	Score	Mean	Stddev	Score	Mean	Stddev
Milk	60%	17.3	4.2	80%	48.8	13.0	100%	3.6	0.4	100%	18.3	18.6
Bottle	10%	35.1	0.0	80%	50.4	12.8	100%	3.5	0.3	100%	17.9	16.8
Toy	50%	37.0	17.2	50%	39.3	11.8	100%	3.9	0.3	90%	28.3	15.0
Fan	20%	17.3	9.7	60%	42.5	6.9	100%	3.6	0.2	90%	24.3	19.9
Pringles	10%	27.5	0.0	20%	47.3	8.3	100%	3.7	0.3	70%	30.4	9.7
Can	10%	26.3	0.0	40%	54.0	16.9	80%	3.6	0.3	80%	23.3	19.4
Coffee Tin	20%	6.8	6.7	70%	35.6	13.7	70%	3.5	0.2	100%	15.8	10.2
Cup	30%	27.8	17.6	20%	44.3	15.8	70%	3.6	0.3	80%	29.3	21.8
Total	26%	24.2	15.1	53%	45.0	14.0	90%	3.6	0.3	89%	22.9	17.7

Table 4.5: The results of 10 attempts to pick up each out of a set of 8 objects on a real robot. The mean and standard deviation relate to the time needed to grab the object in each of the successful attempts, in seconds.

Algorithm	Angles	Arms	N	μ	σ	Robot score
CACLA	Absolute	Left	12	91.8%	26.8 pp	89%
3-NN	Absolute	Left	5	100.0%	0.0 pp	53%
3-NN	Relative	Left	5	43.4%	6.5 pp	26%
CACLA	Relative	Left	12	77.8%	38.8 pp	90%

Table 4.6: Results of the best algorithms in simulation compared to their performance on the real robot. μ expresses the success percentage of the trials and σ expresses the standard deviation of the success percentage in percentage points. The robot score is the success percentage over all the trials for all the objects that were attempted to grab on the real NAO.

because both of the algorithms performed a lot better when using this approach. Each algorithm was allowed 60 seconds per attempt to grab an object. They were tested on the set of 8 objects presented in section 2.9. For each object, 10 attempts were performed to grab the object. The absolute angles revealed a problem on the real robot: in simulation, every set of angles was set instantly while on the real robot, it took much longer to set each set of angles. Therefore, the time required to complete is much longer relative to the relative angles than it was in simulation. The results of grasping with the real robot are shown in table 4.5. This table lists the success rate and the mean and standard deviation of the time in seconds until success. As mentioned above, both in the experiments in simulation and the experiments on the actual robot, no learning occurred during testing to avoid that the trials influence the following trials by updating the system. The four algorithms with their scores both in simulation and on the real robot are listed in table 4.6. This table allows for easy comparison of the effect of the reality gap on both algorithms.

4.6 Discussion

The following three sections will discuss the results of the three algorithms implemented.

4.6.1 Learning from Demonstration Using an Artificial Neural Network

As can be seen from the results in the previous section, the ANN did not perform well at all after training on the dataset, in any of the setups. This can mean that the ANN is not suited to extract the relevant information out of the demonstrations or that the demonstrations did not provide sufficient information to grab the object. The higher performance of the nearest neighbor approach which operates directly on the dataset suggests that even though the dataset might not be perfect, it allows for better performance than the ANN is able to get out of it. The explanation for the disappointing performance probably lies in ambiguity present in the data set. Because the demonstrations were performed by a human, the same action was not taken in the same state every time due. This can cause problems because the ANN has to train on multiple conflicting sets of outputs for the same input, and it has no information about which action is better. This is different from training in CACLA+Var, where the output of the critic will give a verdict about which action is better.

Since the high number of parameters of training an ANN it might be possible to obtain better results by better tuning the parameters. A parameter optimization algorithm

was already used to tune these settings, as was described in section 2.1.3, so it is questionable how much improvements can be gained from this. The graphs of each setup showing the mean squared error (MSE) of the ANN show that this error is quite high. Even after 30,000 training epochs, the error remained comparably high. This suggests that the training set is complex in such a way that it cannot be captured by the ANN. Restructuring the input strongly, extracting more information before feeding it to the ANN, could help but this will increase the complexity of the software.

4.6.2 Nearest Neighbor Regression

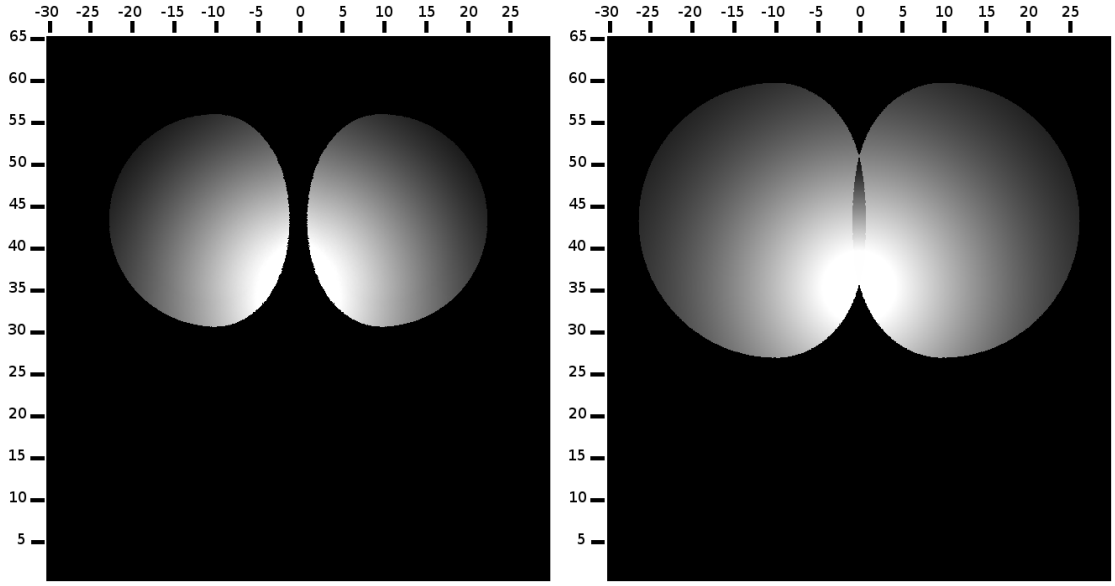
The Nearest Neighbor approach used in this project shows that just interpolating between the data to find the correct action in any state is a feasible approach to obtain reasonable performance. Due to the high level of noise in measurements the results are far from perfect. However, not a lot of time was dedicated to optimize the results using the Nearest Neighbor approach. Performing more parameter tuning or recording a better dataset might strongly improve the performance of this method.

The performance of this algorithm was much worse on the robot than it was in simulation. This is most likely due to the strongly increased noise level when actuating real motors instead of modifying values stored in memory representing the joints. This effect was even stronger for the relative angles. This can be explained by the fact that the error accumulates when using relative angles and the robot gets further and further away from the trajectory from the demonstrations it was trained on. The absolute angles however showed many sudden moves and not a smooth trajectory towards the object to grab.

4.6.3 CACLA+Var

In all situations, the CACLA+Var algorithm was able to learn a valid value function of the problem. A visualization of the value function for an object of 9cm x 10cm located at $(16.5, 0, 35)$ in centimeters is shown in figure 4.14. The figure shows the highest values for the two arms in the white areas, there where the object is. The value decreases gradually when moving away from the object. The areas in the picture that are completely black are areas that are unreachable by the arms of the NAO without moving the rest of its body. It can be seen that at 16.5cm in front of the NAO, the two ellipses have no overlap, meaning that 16.5 cm in front of the NAO, the arms cannot touch each other. Closer to the robot, at 12.5cm, this is not the case. The axes show values in centimeters: the horizontal axis corresponds to the Y axis and the vertical axis to the Z axis in NAO space.

Using the correctly trained critic, the actor function was able to converge to a good



(a) $x = 16.5cm$ (b) $x = 12.5cm$

Figure 4.14: A visualization of the output of the critic for states that result in one plane with two values for x: $x = 16.5cm$, the same plane as the object, and $x = 12.5cm$, closer to the body. The completely black areas are areas that are unreachable by the arms. For the other areas, the brighter the color, the higher the value. The horizontal axis corresponds to the Y axis in NAO space, the vertical axis with the Z axis in NAO space. The values shown are in centimeters.

solution in most cases when it had to output the joint angles for just the left hand, resulting in a successful positioning of the hands after five to ten seconds, a good performance when comparing to the other algorithms. When outputting angles for both hands, the algorithm did not manage to achieve a lot of successes. The reason for this most likely lies in the exponentially increased search space when controlling 10 joints over controlling 5 joints. While it should not pose an insurmountable problem for RL, it will require a lot more training.

The graphs in figures 4.8 and 4.9 show that during training, performance gradually increased, resulting in a decreasing duration until success and an increased success rate. These results could not be obtained for the training to control both hands, which is still due to the exponentially increasing search space when controlling ten joints instead of five.

CACLA+Var with the actor pre-trained on the dataset performed a lot worse com-

pared to running with randomly initialized networks. When considering the results of the learning from demonstration algorithm shown in the previous section this is not surprising, as the network was unable to grab the object at all. Therefore, the bias was towards an incorrect solution, resulting in worse performance on the final tests. Perhaps better results could be obtained when a neural network can be trained to achieve at least a little success, for example trained on a different set of demonstrations. The graphs in figures 4.12 and 4.13 show that when using the pre-trained ANNs, the algorithm had difficulty optimizing the actor, probably because of the large bias towards an incorrect solution.

The performance of CACLA+Var on the robot was surprisingly well, reaching up to 90% performance for both the relative and the absolute angles. The relative angles outperformed the absolute angles slightly but not significantly when looking at the success rate. On the durations of the trials CACLA+Var for absolute and relative angles, a Mann-Whitney test was performed (Mann & Whitney, 1947), with the null-hypothesis that the distributions of the durations of both experiments have the same mean and the alternative hypothesis that $\mu_{rm} > \mu_{am}$, where μ_{am} is the mean for the mirrored arms using absolute angles and μ_{rm} is the mean for the mirrored arms using relative angles. For a significance level of $p < 0.05$ for a one-tailed distribution, this leads to a critical value $U = 2963.4$ with sample sizes 71 and 72. The difference is significant if $u < 2963.5$. The test results in $u = 10.5$, and since $10.5 < 2963.5$, the time required to achieve success is significantly lower when using relative angles than when using absolute angles. The means of both experiments confirm this with a mean of 3.6 seconds for the relative angles as opposed to 22.9 seconds for the absolute angles. Also, when looking at the trajectory towards the goal, the movement of the relative angles was much more smooth and goal-targeted than the absolute angles. When using absolute angles, it looked like the algorithm had converged to an optimal exploration strategy, quickly exploring as much of the action space as possible, while with the relative angles, the arms moved quickly towards the goal, solely reducing the distance towards the object. The explanation for this is that when using absolute angles, the entire action space of the robot is reachable from every state. This means that the next state does not strongly depend on the current state but only on the action selected. Therefore, the trajectory can not really be optimized, just the target. This makes it much harder to get a smooth trajectory towards the target when using absolute angles.

Relative angles on the other hand explicitly make the next state dependent on the current state and reduce the action space for each state. This gives more opportunity to optimize the trajectory, and therefore seems better suited for reinforcement learning.

Interestingly, the actor trained using CACLA+Var seemed to have formed a policy that effectively seems to work for many objects. It first brought the hands together to

enclose the width of the object at a high position and then gradually moved the arms down until the correct position had been reached. This is a good strategy to grab objects of any size without changing the complete trajectory. Also, when looking at the results, the algorithm was able to grab the object in approximately 3 to 4 seconds every time, so the results are really well compared to the other algorithms.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

The results show that CACLA+Var is a good approach to learning complex tasks such as controlling a robot arm to move towards a target. Nearest Neighbor approaches are also viable candidates although the parameters should be tuned. The major drawback is that a large dataset is required to perform good interpolation. CACLA+Var on the other hand can be trained without any prior training and without any dataset, making it easier to apply to different robots using different kinematics. Also, Nearest Neighbor suffers from the noise resulting from operation on the real robot while the results show that CACLA+Var does not suffer from this additional noise.

Seeing that the results of the training of the CACLA+Var algorithm were also directly applicable on the robot without any modification, the use of simulations to speed up training is highly valuable.

5.1.1 The Reality Gap

In the experiments discussed in the previous chapter the learning from demonstration with ANNs was unable to achieve any success. Therefore, it was not tested on the real robot. The K-Nearest Neighbor algorithm and CACLA+Var were tested on the robot. The results clearly show that performance of KNN decreases tremendously when performing on the actual robot. CACLA+Var even slightly increased its performance when running on the real robot, managing to pick up the object around 90% of the trials. Especially the implementation using relative angles performed very good, picking up the object in less than 4 seconds most of the time, while the variant using absolute angles took around five times longer to successfully pick up the object.

A complication when running on the actual robot using actual objects is that the objects are not generated but instead have to be recognized. The object recognition method used in this research, SURF features, was able to recognize most of the objects. It especially had problems recognizing the shiny surface of the coffee pad tin, and to a smaller extent, the plastic bottle. Also, the lack of texture of the computer fan made it hard to recognize using SURF features. The position of the object as calculated from the location of the recognized SURF features was also prone to errors. Because the shape of the object in the camera image as determined by the SURF feature matching algorithm usually differed quite a lot from the actual dimensions and shape of the object, due to distortion because of the camera angle looking down at the object, the system was not always able to properly estimate the position of the object. CACLA+Var exploited the policy it had learned to quickly put the hands together at a high position centered in front of the robot, and then moving the hands down along the width of the object. This policy seemed to work quite well, even when the estimate of the object was incorrect. The other algorithms suffered more from this imperfection in object recognition and pose estimation. A lot can be gained from improving the pose estimation, for example by using a depth sensor in addition to the RGB image of the camera.

Despite all this, CACLA+Var performed very well and did not show any difficulty in dealing with both the increased noise of running on the real robot and the imperfections in the pose estimation data. It can therefore be concluded that reinforcement learning in general and CACLA+Var seems to be a very robust method for robot learning, even when considering the complex task of motor control for a humanoid robot arm.

Table 4.6 also shows the effect of the reality gap on the two algorithms tested on both absolute and relative angles. It is surprising to see that CACLA+Var even performs better on the real robot than it did in simulation, reaching up to 90% performance on the test set of 8 objects.

5.1.2 Research Questions

The main research question, “Can machine learning algorithms be used to control the joints of a humanoid robot in order to grasp an object?” can be answered positively: two of the three algorithms tested are able to achieve over 50% performance on the real robot.

The first subquestion, “Which of the evaluated algorithms, learning from demonstration, nearest neighbor or CACLA+Var, performs best on the task of grasping an object?”, can be answered by looking at the success rate and time needed to reach the

goal state in table 4.5. On the real robot, CACLA+Var performs significantly better using both relative and absolute angles than nearest neighbor.

The second subquestion, “Which form of control, the target angular values for the joints or the angular difference relative to the current state of the joints, is better suited for machine learning?”, can be answered by the same table. CACLA+Var using relative angles resulted in both the most consistent and significantly shortest durations. Also, visually, the trajectories were much smoother than when using absolute angles.

5.2 Future Work

Some ends were left open in this research. The Nearest Neighbor approach could probably be improved by improving the dataset, for example by filtering out outliers and smoothing the movements, or by increased sampling of the movements during the demonstrations. In spite of this, the performance was still a good start. Therefore, a good approach to train the actor of the CACLA+Var algorithm might also be to initially use the output of the Approximate Nearest Neighbor algorithm to select the next action instead of using the output of the (initially random) actor. Gradually, this effect could then be removed to have the actor take over control.

The object recognition system in this project used only SURF features for object recognition and pose estimation. Combining several techniques, such as the MOPED framework (Collet et al., 2011) might help to improve the recognition and pose estimation, resulting in better input to the algorithm and possibly increased performance.

Bibliography

Arkin, R. (1998). *Behavior-based robotics*. The MIT Press.

Baird, I., Klopf, A., & OH., W. L. W.-P. A. (1993). *Reinforcement learning with high-dimensional, continuous actions*. Defense Technical Information Center.

Ballard, D. (1981). Generalizing The Hough Transform To Detect Arbitrary Shapes. *Pattern Recognition*, *13*(2), 111–122.

Barto, A., Sutton, R., & Anderson, C. (1983). Neuronlike Adaptive Elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man & Cybernetics*, *13*(5), 834–846.

Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-Up Robust Features (SURF). *Computer Vision And Image Understanding*, *110*(3), 346–359.

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). SURF: Speeded up robust features. In Leonardis, A and Bischof, H and Pinz, A (Ed.) *Computer Vision - ECCV 2006 , PT 1, Proceedings*, vol. 3951 of *Lecture Notes in Computer Science*, (pp. 404–417). Adv Comp Vis; Graz Univ Technol; Univ Ljubljana, Heidelberger Platz 3, D-14197 Berlin, Germany: Springer-Verlag Berlin. 9th European Conference on Computer Vision (ECCV 2006), Graz, Austria, May 07-13, 2006.

Bocsi, B., Nguyen-Tuong, D., Csato, L., Schölkopf, B., & Peters, J. (2011). Learning inverse kinematics with structured prediction. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, (pp. 698–703). IEEE.

Bridle, J. (1990). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in neural information processing systems 2*, (pp. 211–217). Morgan Kaufmann Publishers Inc.

Chacón, J., Van Elteren, T., Hickendorff, B., Van Hoof, H., Lier, C., Nolte, A., Mutis, D., Neculoiu, P., Oost, C., Van der Wal, E., et al. (2011). Borg – the robocup@ home team of the university of groningen. Team Description Paper.

Cheraghpour, F., Moosavian, S., & Nahvi, A. (2010). Robotic Grasp Planning by Multiple Aspects Grasp Index for Object Manipulation Tasks. In *2010 18th Iranian Conference on Electrical Engineering (ICEE)*, (p. 6 pp.). Piscataway, NJ, USA: IEEE. 2010 18th Iranian Conference on Electrical Engineering (ICEE), 11-13 May 2010, Isfahan, Iran.

Collet, A., Martinez, M., & Srinivasa, S. S. (2011). The MOPED framework: Object recognition and pose estimation for manipulation. *International Journal of Robotics Research*, *30*(10, SI), 1284–1306.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, *13*(1), 21–27.

Daoud, N., Gazeau, J. P., Zeghloul, S., & Arsicault, M. (2011). A fast grasp synthesis method for online manipulation. *Robotics And Autonomous Systems*, *59*(6), 421–427.

Fahlman, S. (1988). Faster-learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 connectionist models summer school*, (pp. 38–51). Morgan Kaufmann.

Fahlman, S. (1990). The cascade-correlation learning architecture. Tech. rep., DTIC Document.

Graf, B., Hans, M., & Schraft, R. (2004). Care-O-bot II–development of a next generation robotic home assistant. *Autonomous robots*, *16*(2), 193–205.

Hagedon, M., Grubbs, L., Morris, J., Lammmers, J., & Fan, H. (2009). *UC Robomow 2009*. Ph.D. thesis, University of Cincinnati.

Hansen, L., & Salamon, P. (1990). Neural network ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *12*(10), 993–1001.

Hansen, N., Müller, S., & Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, *11*(1), 1–18.

Holz, D., Paulus, J., Breuer, T., Giorgana, G., Reckhaus, M., Hegger, F., Müller, C., Jin, Z., Hartanto, R., Ploeger, P., et al. (2009). The b-it-bots robocup@ home 2009 team description paper. *RoboCup 2009@ Home League Team Descriptions, Graz, Austria*.

Hsiao, K., Kaelbling, L. P., & Lozano-Perez, T. (2011). Robust grasping under object pose uncertainty. *Autonomous Robots*, *31*(2-3), 253–268.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. *Advances in artificial life*, (pp. 704–720).

Jones, J., & Palmer, L. (1987). An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, *58*(6), 1233–1258.

Kaelbling, L. (1993). *Learning in embedded systems*. The MIT Press.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, (pp. 340–347). ACM.

Klingbeil, E., Saxena, A., & Ng, A. Y. (2008). Learning to open new doors. In *Proc. of Robotics: Science and Systems*. Zurich, Switzerland.

Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, (pp. 282–293).

Kouskouridas, R., Amanatiadis, A., & Gasteratos, A. (2011). Guiding a robotic gripper by visual feedback for object manipulation tasks. In M. Gokasan, S. Bogosyan, & E. Yesil (Eds.) *Proceedings of the 2011 IEEE International Conference on Mechatronics (ICM)*, (pp. 433–8). IEEE, Piscataway, NJ, USA: IEEE. 2011 IEEE International Conference on Mechatronics (ICM), 13-15 April 2011, Istanbul, Turkey.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *2*(60), 91 – 110.

Lu, H., Zhang, H., Yang, S., & Zheng, Z. (2009). A Novel Camera Parameters Auto-adjusting Method Based on Image Entropy. In *RoboCup 2009: Robot Soccer World Cup XIII*, (pp. 192–203). Berlin, Germany: Springer-Verlag.

Malmir, M., & Shiry, S. (2009). Object Recognition with Statistically Independent Features: A Model Inspired by the Primate Visual Cortex. In *RoboCup 2009: Robot Soccer World Cup XIII*, (pp. 204–14). Berlin, Germany: Springer-Verlag.

Mann, H., & Whitney, D. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, *18*(1), 50–60.

Matheus, K., & Dollar, A. (2010). Benchmarking grasping and manipulation: Properties of the objects of daily living. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, (pp. 5020 –5027).

Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application (VISSAPP'09)*, (pp. 331–340). INSTICC Press.

Nissen, S. (2003). Implementation of a fast artificial neural network library (fann). *Report, Department of Computer Science University of Copenhagen (DIKU)*, *31*.

Oost, C., & Jansen, E. (2011). Object manipulation with a NAO. Bachelor's Thesis.

Peters, J., & Schaal, S. (2006). Reinforcement learning for parameterized motor primitives. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, (pp. 73–80). IEEE.

Peters, J., & Schaal, S. (2008a). Natural actor-critic. *Neurocomputing*, *71*(7), 1180–1190.

Peters, J., & Schaal, S. (2008b). Policy learning for motor skills. In *Neural Information Processing*, (pp. 233–242). Springer.

Pineda, F. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, *59*(19), 2229–2232.

Prokhorov, D., & Wunsch, D. (1997). Adaptive critic designs. *Neural Networks, IEEE Transactions on*, *8*(5), 997–1007.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*, (pp. 586–591).

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Rummery, G., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166*. Univ. of Cambridge, Department of Engineering.

Saxena, A., Wong, L., Quigley, M., & Ng, A. Y. (2007). A vision-based system for grasping novel objects. In *International Symposium of Robotics Research*, vol. 13.

Saxena, A., Wong, L. L. S., & Ng, A. Y. (2008). Learning grasp strategies with partial shape information. In *AAAI*.

Schaal, S. (1997). Learning from demonstration. *Advances in neural information processing systems*, (pp. 1040–1046).

Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., & Poggio, T. (2007). Robust object recognition with cortex-like mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *29*(3), 411–426.

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, *3*(1), 9–44.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, (pp. 1038–1044).

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*, vol. 28. Cambridge Univ Press.

Takamuku, S., Hosoda, K., & Asada, M. (2008). Object Category Acquisition by Dynamic Touch. *Advanced Robotics*, *22*(10), 1143–1154.

Tribelhorn, B., & Dodds, Z. (2007). Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education. In *Robotics and Automation, 2007 IEEE International Conference on*, (pp. 1393–1399). IEEE.

Van Hasselt, H., & Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, (pp. 272 – 279).

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, (pp. I–511 – I–518).

Van der Wal, T. (2011). Automatic parameter optimization. Internal Report.

Watkins, C. (1989). *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge.

Wisspeintner, T., Van der Zant, T., Iocchi, L., & Schiffer, S. (2009). RoboCup@Home Scientific Competition and Benchmarking for Domestic Service Robots. *Interaction Studies*, *10*(3, SI), 392–426.

Wisspeintner, T., Van der Zant, T., Iocchi, L., & Schiffer, S. (2010). Robocup@home: Results in benchmarking domestic service robots. In J. Baltes, M. Lagoudakis, T. Naruse, & S. Ghidary (Eds.) *RoboCup 2009: Robot Soccer World Cup XIII*, vol. 5949 of *Lecture Notes in Computer Science*, (pp. 390–401). Springer Berlin / Heidelberg.

Yoshikawa, T. (2010). Multifingered robot hands: Control for grasping and manipulation. *Annual Reviews in Control*, *34*(2), 199–208.

Van der Zant, T., Schomaker, L., & Haak, K. (2008). Handwritten-word spotting using biologically inspired features. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *30*(11), 1945–1957.

Van der Zant, T., Wiering, M., & Van Eijck, J. (2005). On-line robot learning using the interval estimation algorithm. In *Proceedings of the 7th European Workshop on Reinforcement Learning*, vol. 7, (pp. 11–12). Napoli, Italy.