KTH Computer Science
and Communication

Robot Task Learning from Human Demonstration

STAFFAN EKVALL

Doctoral Thesis
Stockholm, Sweden 2007

TRITA-CSC-A 2007:01
ISSN-1653-5723
ISRN-KTH/CSC/A–07/01–SE
ISBN 978-91-7178-570-1

KTH School of Computer Science and Communication
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi fredagen den 23 februari 2007 klockan 10.00 i sal E2, Lindstedtsvägen 3 entreplan, Kungl Tekniska högskolan, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Today, most robots used in the industry are preprogrammed and require a well-defined and controlled environment. Reprogramming such robots is often a costly process requiring an expert. By enabling robots to learn tasks from human demonstration, robot installation and task reprogramming are simplified. In a longer time perspective, the vision is that robots will move out of factories into our homes and offices. Robots should be able to learn how to set a table or how to fill the dishwasher. Clearly, robot learning mechanisms are required to enable robots to adapt and operate in a dynamic environment, in contrast to the well defined factory assembly line.

This thesis presents contributions in the field of robot task learning. A distinction is made between direct and indirect learning. Using direct learning, the robot learns tasks while being directly controlled by a human, for example in a teleoperative setting. Indirect learning, however, allows the robot to learn tasks by observing a human performing them. A challenging and realistic assumption that is decisive for the indirect learning approach is that the task relevant objects are not necessarily at the same location at execution time as when the learning took place. Thus, it is not sufficient to learn movement trajectories and absolute coordinates. Different methods are required for a robot that is to learn tasks in a dynamic home or office environment. This thesis presents contributions to several of these enabling technologies. Object detection and recognition are used together with pose estimation in a Programming by Demonstration scenario. The vision system is integrated with a localization module which enables the robot to learn mobile tasks. The robot is able to recognize human grasp types, map human grasps to its own hand and also evaluate suitable grasps before grasping an object. The robot can learn tasks from a single demonstration, but it also has the ability to adapt and refine its knowledge as more demonstrations are given. Here, the ability to generalize over multiple demonstrations is important and we investigate a method for automatically identifying the underlying constraints of the tasks.

The majority of the methods have been implemented on a real, mobile robot, featuring a camera, an arm for manipulation and a parallel-jaw gripper. The experiments were conducted in an everyday environment with real, textured objects of various shape, size and color.

Acknowledgements

There are many people who have inspired and supported me on this thesis. First I would like to thank my supervisor Danica Kragic, for your enthusiasm, our fruitful discussions and for your extraordinary guidance, support and encouragement which pushed me to perform my very best. Thank you Frank Hoffmann, for inspiring me to pursue research in the first place. My thanks also goes to Jan-Olof Eklundh and Stefan Carlsson for providing a stimulating research environment.

The many friendly people at CAS/CVAP have also contributed to this thesis by creating a nice atmosphere filled with interesting discussions. In particular, thank you Daniel Aarno for our research discussions, for sharing your deep programming knowledge and for your great patience with my Unix frustration. Thank you Patric Jensfelt, for your never ending patience, helpful attitude and support on technical matters. You keep CAS running! Many thanks to all the other people at CAS/CVAP. Thank you Babak, for the challenging coffee breaks. Hugo, always fun to talk with. Johan S, for the fun of sharing a room with you. Christian, you can truly discuss anything. Frank L, was ist los? Paul, you are an optimal friend. Andreas, for our pizza days. Johan T and Oscar, for our productive discussions. And to all others at CAS/CVAP, thank you **all**, you all contributed to this thesis in some way.

Finally, I would like to express my gratitude to my family for your interest and encouragement. Special thanks to my beloved wife Marika. Thank you, for your endless love and support.

This work has in part been funded by the Swedish Research Council. The funding is gratefully acknowledged.

Contents

Contents **vi**

1 Introduction **1**
1.1 Direct and Indirect Learning 2
1.2 Supervised and Unsupervised Learning 3
1.3 Outline and Contributions 4
1.4 List of Publications 5

2 Machine-Assisted Task Execution Using Direct Learning **7**
2.0.1 Human Machine Collaborative Systems 8
2.1 System Overview 8
2.2 Theoretical Background 10
2.2.1 Hidden Markov Models 10
2.2.2 Probability Estimators for Hidden Markov Models 11
2.2.3 Support Vector Machines 11
2.3 Related Work 13
2.4 Trajectory Analysis 14
2.4.1 Retrieving Measurements 14
2.4.2 Estimating Lines in the Demonstrated Trajectories 14
2.4.3 Estimating Observation Probabilities Using Support Vector Machines 15
2.4.4 State Sequence Analysis Using Hidden Markov Models 16
2.5 Experimental Evaluation 16
2.5.1 Experiment 1: Trajectory Following 18
2.5.2 Experiment 2: Changed Workspace 19
2.5.3 Experiment 3: Unexpected Obstacle 21
2.6 Discussion 21

3 Robot Vision for Indirect Learning **23**
3.1 System Overview 24
3.2 Related Work 25
3.3 Color Cooccurrence Histograms 26
3.3.1 Image Normalization 26
3.3.2 Image Quantization 27

3.3.3 Histogram Matching 27
3.3.4 Object Detection and Segmentation 28
3.4 Receptive Field Cooccurrence Histograms 29
3.4.1 Image Descriptors 29
3.4.2 Image Quantization 30
3.4.3 An Alternative Segmentation Approach 30
3.4.4 Complexity 32
3.5 Object Detection Evaluation 32
3.5.1 CODID - CVAP Object Detection Image Database 33
3.5.2 Training 33
3.5.3 Detection Results 34
3.5.4 Segmentation Results 35
3.5.5 Free Parameters 36
3.5.6 Scale Robustness 38
3.5.7 Conclusion 38
3.6 Pose Estimation 39
3.6.1 Model Based Pose Estimation 40
3.6.2 Experimental Evaluation 42
3.6.3 Object Recognition and Rotation Estimation 42
3.6.4 Full 6-DoF Pose Estimation 45
3.7 Discussion 46

4 Grasp Mapping, Recognition and Execution 49
4.0.1 GraspIt! 50
4.1 Mapping Human Grasps to Robot Grasps 50
4.1.1 Measuring the Hand Posture 51
4.1.2 Using an Artificial Neural Network for Grasp Mapping 53
4.1.3 Evaluation 54
4.1.4 Object Grasping 57
4.1.5 Conclusion 59
4.2 Autonomous Grasping Based on Human Advice 59
4.3 Grasp Recognition 62
4.3.1 Applications 63
4.3.2 Related Work on Grasp Recognition 63
4.3.3 Grasp Recognition: Two Methods 64
4.3.4 Grasp Classification Based on Fingertip Positions 64
4.3.5 Grasp Classification Based on Arm Movement Trajectories 66
4.3.6 Experimental Evaluation 68
4.3.7 Conclusion 72
4.4 Autonomous Grasping Inspired by Human Demonstration 73
4.4.1 Related Work on Grasping 74
4.4.2 Grasp Mapping 75
4.4.3 Grasp Controllers 76
4.4.4 Grasp Planning 77

4.4.5 Experimental Evaluation 78
4.4.6 Conclusion 82
4.5 Discussion 83

5 Task Level Learning from Demonstration 85
5.1 Motivation and Related Work 86
5.2 System Description 88
5.2.1 Experimental Platform 89
5.3 Task Level Planning 89
5.3.1 Pose Estimation 92
5.3.2 Detecting Object Collisions 92
5.3.3 Finding Free Space 95
5.3.4 Taking Profit from Human Advice 96
5.4 Automatic Generalization from Multiple Examples 96
5.4.1 Example Task 97
5.4.2 State Generation 98
5.4.3 Task Generalization 99
5.5 Experimental Evaluation 100
5.5.1 Planning Example 100
5.5.2 Imitation Learning 101
5.5.3 Learning from Human Advice 103
5.5.4 Generalizing from Multiple Examples 104
5.6 Discussion 107

6 A Service Robot Application 109
6.1 Motivation and Related Work 109
6.1.1 Active Vision 111
6.2 Building a Map of the Environment 112
6.3 Active Object Recognition 112
6.3.1 Active Object Learning from Demonstration 113
6.3.2 Hypotheses Generation 114
6.3.3 Hypotheses Evaluation Strategy 114
6.4 Integrating SLAM and Object Recognition 116
6.5 Experimental Evaluation 116
6.5.1 Evaluating the Search Effectiveness 117
6.5.2 Searching for Objects in Several Rooms 119
6.5.3 Fetching Objects 120
6.6 Discussion 122

7 Summary and Future Work 123
7.1 Summary 123
7.2 Future Work and Perspective 125

Bibliography 127

Chapter 1

Introduction

Today, most robots used in the industry are preprogrammed and require a well-defined and controlled environment. Reprogramming such robots is often a costly process requiring an expert. Enabling the robot to learn tasks by demonstrating them would simplify the robot installation and task reprogramming. In a longer time perspective, the vision is that robots will move out of factories into our homes and offices. Robots should be able to learn how to set a table, or how to fill the dishwasher. Clearly, robot learning mechanisms are required to enable robots to adapt and operate in a dynamic environment, in contrast to the well defined factory assembly line. That is why robot learning is one of the key research areas in robotics. However, constructing a robot that is able to learn what is shown is a challenging problem. Although prototype platforms for robot learning by demonstration have been around for more than 10 years, the many difficulties have restrained the robots to only operate in lab environments. Some of the key challenges are *perception and task/object recognition*, *task generalization*, *planning* and *object manipulation*. This thesis presents various contributions in each of these fields and also gives several examples of robotic task learning solutions.

An example task which robots should be able to learn is *setting the table*. It involves moving plates and cutlery to the correct positions on the table. This task has to be learned on site since a preprogrammed robot cannot know the size and shape of the table, among other things. Despite the simple appearance of the task, it is actually very complicated. The robot has to learn to recognize plates, knives, pots and napkins, to name a few items. Then, it has to learn how to grasp them in a robust manner, and transport them to the correct location on the table. Some items may block each other so that the robot cannot grasp them. It has to create a plan on how to achieve the task goals despite these obstacles. Thus, the robot has to understand the task goals from a demonstration.

As shown in the above example, robot learning is utilized on many different levels, from simple parameter tuning to high-level task learning. Fig. 1.1 shows some examples of different levels of learning.

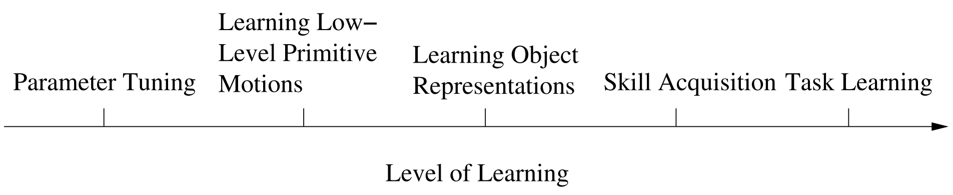

Parameter Tuning
Learning Low–Level Primitive Motions
Learning Object Representations
Skill Acquisition
Task Learning
Level of Learning

Figure 1.1: Some examples of different levels of robot learning. On the left we find simple parameter tuning and learning of low-level primitive motions, while on the right high-level learning systems such as skill acquisition(e.g object manipulation) and task learning are situated.

1.1 Direct and Indirect Learning

We consider two ways for a robot to learn from demonstration, direct learning and indirect learning.

- **Direct Learning**
  A human performs the task by directly controlling the robot through a joystick or similar device. The robot records sensory information during the demonstration and is then able to reproduce the task. The robot can generalize over multiple demonstrations and gain ability to perform the task even better than the human. This approach has the advantage that no mapping from human to robot kinematics is needed. Also, the robot can expect about the same sensor readings during execution. The disadvantage with direct learning is that controlling a high degree-of-freedom robot is quite hard, and some tasks are not be possible to demonstrate using the robot.

- **Indirect Learning**
  In this approach, the robot learns by observing a human performing the task. This method is much more difficult to realize, as it requires the robot to have remote sensing(vision), and reasoning about what it is observing. The trajectory of the demonstration cannot be mapped directly to the robot because of the different kinematics. The low-level sensory information must be transformed to high-level situation-action descriptors (Friedrich *et al.*, 1996), and then mapped back to low-level motor controls dependent of the world state at run-time. The advantages of this approach are both that the operator can demonstrate the task in a natural way, and that it results in a much more flexible system. Learning a concept rather than low-level trajectories allows the robot to adapt its knowledge to new situations, never encountered before.

Fig. 1.2 shows what sensors and methods are required for a complete learning system in a dynamic environment. The contributions of this thesis lie more in the development of *enabling technologies* for robot task learning from demonstration, than actual task learning techniques, although we present some in Chapter 5.

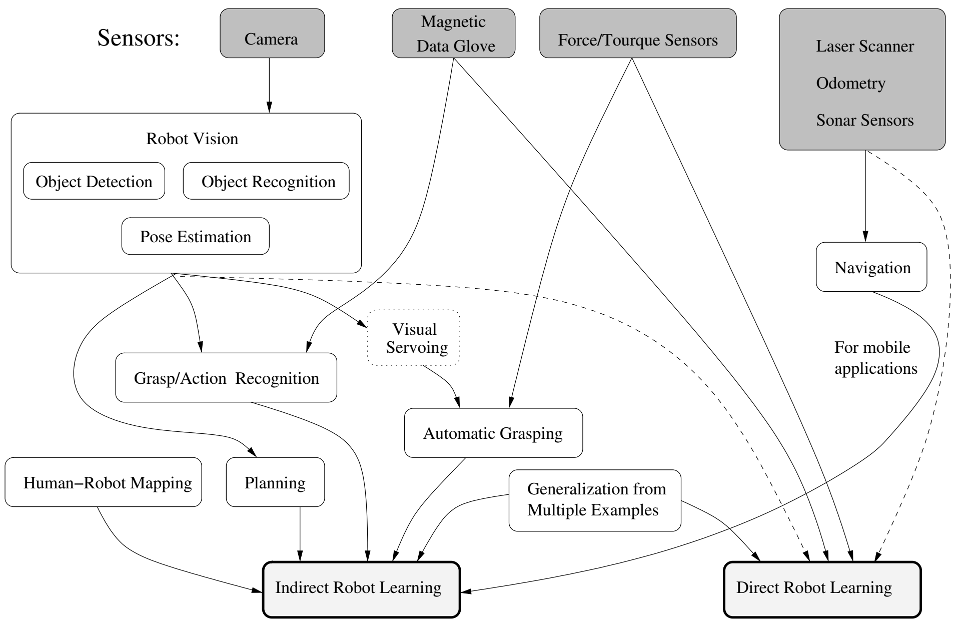


Figure 1.2: From sensors to complete learning systems. As seen, the direct method mostly operates on raw sensor data, while the indirect method require many high-level learning modules. The dotted lines represent possible connections that are not used in this thesis.

1.2 Supervised and Unsupervised Learning

In the field of machine learning, it is common to distinguish between *supervised* and *unsupervised* learning. In supervised learning, the learning agent is provided with the correct answers to the problems faced. Often the answers are in the form of target output vectors $\mathbf{y}_i$, which is the desired output for each input vector $\mathbf{x}_i$. These targets can be learned by observing a human performing the task. On the other hand, unsupervised learning models a set of inputs when labeled examples are not available. In this thesis, mostly supervised learning methods are utilized. One example of an unsupervised approach is the clustering technique which is frequently used throughout the thesis. Here, the challenge is to find structures in n-dimensional data sets without any *a priori* information.

A popular machine learning method which falls in between the two above categories is *Reinforcement Learning*, (Sutton and Barto, 1998). Instead of providing a target for each input vector, the robot is guided by rewards and penalties. This has the advantage the robot can find an optimal solution to a problem using trial and error, just given the desired goal state. However, robot tasks have often huge state spaces and since most tasks cannot be simulated, the robot has to perform many time-consuming trials when exploring the state space. It must also be able to detect all changes to the environment that is caused by each trial. Due to these problems, we have chosen not to use reinforcement learning in this work.

1.3 Outline and Contributions

This thesis presents background and contributions several of the methods shown in Fig. 1.2.

- **Chapter 2: Machine-Assisted Task Execution Using Direct Learning**
  In this chapter, a human-machine collaborative system is considered. Such systems are useful when the task requires high precision or power, but cannot be automated due to the need for human decision making. It has been demonstrated in a number of robotic areas how the use of *virtual fixtures* improves task performance both in terms of execution time and overall precision, (Kuang *et al.*, 2004). However, the fixtures are typically inflexible, resulting in a degraded performance in cases of unexpected obstacles or incorrect fixture models. In Chapter 2, we present *adaptive virtual fixtures* that enable us to cope with the above problems.

- **Chapter 3: Robot Vision for Indirect Learning**
  To enable indirect learning, the robot must be able to learn by observing a demonstration instead of learning as it performs the task. In Chapter 3, we present techniques for autonomous object detection and pose estimation, which are some of the key modules to enable indirect learning. The methods are designed with the learning scenario in mind; the robot is to operate in cluttered home and office environments.

- **Chapter 4: Grasp Mapping, Recognition and Execution**
  Some other necessary modules for indirect learning are grasp recognition and mapping. The chapter starts with grasp mapping in a direct-control setting. Then, more intelligence is added as grasp recognition is introduced. The robot is to learn not only *what* is done, but also *how* it is done. Most objects can be grasped in several ways, depending on the task at hand. Grasp recognition allows the robot to recognize the human grasps during a demonstration. Then, a fixed grasp mapping is necessary to translate the grasps to a robot equivalent type. In the end of the chapter, we present a technique to enable autonomous grasping of objects once the grasp has been recognized and the pose of the object has been estimated.

- **Chapter 5: Task Level Learning from Demonstration**
  In this chapter, we demonstrate how the robot can be taught a pick-and-place task from demonstration. The key challenge here is that the initial task setting may change after the demonstration, which requires the robot to understand the task and plan a series of actions to achieve the task goals. It is not sufficient to learn low-level movement trajectories. We also show how the robot can generalize the task model from multiple demonstrations.

- **Chapter 6: A Service Robot Application**
  In this chapter, we integrate some of our methods with a navigation system, which

allows the robot to perform mobile tasks. The vision system from Chapter 3 is extended to robustly recognize objects without any false positives. The robot is then able to perform sophisticated tasks, such as moving to a room and searching for a specific object.

- **Chapter 7: Summary and Future Work**
  The final chapter summarizes the most important parts of the thesis, provides some further discussion and also highlights issues for future research.

1.4 List of Publications

Most of the work presented in this thesis can also be found in the following publications:

- Learning and Evaluation of the Approach Vector for Automatic Grasp Generation and Planning (S. Ekvall and D. Kragic) To appear in IEEE/RSJ International Conference on Robotics and Automation, 2007
- Object Detection and Mapping for Service Robot Tasks (S. Ekvall, D. Kragic and P. Jensfelt) To appear in *Robotica*, Cambridge Journals, 2007
- On-line Task Recognition and Real-Time Adaptive Assistance for Computer Aided Machine Control (S. Ekvall, D. Aarno and D. Kragic) *Transactions on Robotics*, October 2006, pp 1029-1033, vol 22, issue 5
- Integrating Active Mobile Robot Object Recognition and SLAM in Natural Environments (S. Ekvall, P. Jensfelt and D. Kragic) In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp 5798-5804
- Learning Task Models from Multiple Human Demonstrations (S. Ekvall and D. Kragic) In IEEE International Symposium on Robot and Human Interactive Communication, 2006, pp 358-363
- Task Learning Using Graphical Programming and Human Demonstrations (S. Ekvall, D. Aarno and D. Kragic) In IEEE International Symposium on Robot and Human Interactive Communication, 2006, pp 398-403,
- Augmenting SLAM with Object Detection in a Service Robot Framework (P. Jensfelt, S. Ekvall, D. Kragic and D. Aarno) In IEEE International Symposium on Robot and Human Interactive Communication, 2006, pp 741-746
- Object Recognition and Pose Estimation using Color Cooccurrence Histograms and Geometric Modeling (S. Ekvall, D. Kragic and F. Hoffmann) *Image and Vision Computing*, October 2005, pp 943-955, vol 23, issue 11
- Selection of Virtual Fixtures Based on Recognition of Motion Intention for Teleoperation Tasks (D. Aarno, S. Ekvall and D. Kragic) In Proceedings of the third Swedish Workshop on Autonomous Robotics, 2005

- Receptive Field Cooccurrence Histograms for Object Detection (S. Ekvall and D. Kragic) In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005, pp 84-89
- Grasp Recognition for Programming by Demonstration (S. Ekvall and D. Kragic) In IEEE/RSJ International Conference on Robotics and Automation, 2005, pp 748-753
- Adaptive Virtual Fixtures for Machine-Assisted Teleoperation Tasks (D. Aarno, S. Ekvall and D. Kragic) In IEEE/RSJ International Conference on Robotics and Automation, 2005, pp 897-903
- Integrating Object and Grasp Recognition for Dynamic Scene Interpretation, (S. Ekvall and D. Kragic) In IEEE/RSJ International Conference on Advanced Robotics, 2005, pp 331-336
- Interactive Grasp Learning Based on Human Demonstration (S. Ekvall and D. Kragic) In IEEE/RSJ International Conference on Robotics and Automation, 2004, pp 3519-3524, vol 4
- Object Recognition and Pose Estimation for Robotic Manipulation using Color Cooccurrence Histograms (S. Ekvall, F. Hoffmann and D. Kragic) In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003, pp 1284-1289, vol 2

The following paper is under review.

- Robot Learning from Demonstration: A Task-Level Planning Approach (S. Ekvall and D. Kragic) Submitted to IEEE *Transactions on Robotics*

Chapter 2

Machine-Assisted Task Execution Using Direct Learning

In today's manufacturing industry, large portions of the operation has been automated. However, many processes are too difficult to automate and must rely on humans' decision making and superior performance in areas such as identifying defective parts, dealing with process variations, pushing cable bundles aside (Peshkin *et al.*, 2001), or medical applications (Taylor and Stoianovici, 2003). When such skills are required, humans still have to perform straining tasks. We believe that Human-Machine Collaborative Systems (HMCS) can be used to help prevent ergonomic injuries and operator wear, by allowing cooperation between a human and a (mobile) manipulation platform. In such a system, the user's intention is recognized and the system is aiding the user in performing the task.

Segmentation and recognition of operator generated motions are commonly facilitated to provide appropriate assistance during task execution in teleoperative and human-machine collaborative settings. The assistance is usually provided in a virtual fixture framework where the level of compliance can be altered on-line thus improving the performance in terms of execution time and overall precision. However, the fixtures are typically inflexible, resulting in a degraded performance in cases of unexpected obstacles or incorrect fixture models. In this chapter, we present a method for on-line task tracking and propose the use of *adaptive virtual fixtures* that can cope with the above problems. Here, rather than executing a predefined plan, the operator has the ability to avoid unforeseen obstacles and deviate from the model. To allow this, the probability of following a certain trajectory (subtask) is estimated and used to automatically adjust the compliance, thus providing the on-line decision of how to fixture the movement.

Related to Fig. 1.2, the system presented in this chapter is an example of direct task learning, where the robot learns directly from sensory data. The goal of this chapter is to equip a stationary robot with learning capabilities for direct learning. The human controls the robot using either a joystick, a force-torque controller or some other device. The robot records the end effector position during the human demonstration. When training is complete, the robot has learned the nature of the task and is therefore aware of the user's intention. Hence, it is possible to aid the user in upcoming task executions. We demonstrate the learning system with a series of experiments using a real robot.

2.0.1 Human Machine Collaborative Systems

In the area of HMCSs and teleoperation, task segmentation and recognition are two important research problems. In this chapter, it is shown how a flexible design framework can be obtained by building a low-level Programming by Demonstration system where the robot can be trained in a fast and easy way. In our system, a high-level task is segmented into subtasks where each of the subtasks has a *virtual fixture* obtained from 3D training data. Virtual fixtures are commonly defined as a task-dependent aid for teleoperative purposes, (Payandeh and Stanisic, 2002) and used to constrain the user's or the manipulator's motion in undesired directions while allowing or aiding motion along the desired directions. Here, a virtual fixture is a physical constraint that forces a robot to move along desired paths. A state sequence analyzer learns what subtasks are more probable to follow each other which is then used by an on-line state estimator that estimates the probability of the user being in a particular state. A specific virtual fixture, corresponding to the most probable state can then be applied.

2.1 System Overview

Given a training trajectory, we wish to apply a virtual fixture to aid the user in following the trajectory. Furthermore, to cope with the above mentioned problems with virtual fixtures, we introduce the concept of *adaptive* virtual fixtures. Here, the trajectory is divided into several line segments, and each line segment is "stretchable", meaning that the user can continue to follow a certain line segment for as long as necessary. However, this solution comes with a number of challenges. We have to automatically divide the trajectory into lines, and at run-time, identify which line segment the user is currently following. An overview of the system is shown in Fig. 2.1.

The components of the system are shortly introduced below:

Measurement Retrieval - During both training and execution, sensor measurements are recorded and used to control the robot.

Line Estimation - The recorded time-position tuples form a trajectory. We model this trajectory as a sequence of linear movements. Higher-order models are possible, but in this work we chose a linear model because of its simplicity. As presented in Section 2.4.2, lines are automatically found in the demonstrated trajectories using K-means clustering.

Line Probability Estimation - Each sample provides, together with the previous sample, a short line. Given that the task model consists of a limited number of lines, it is possible to estimate the probability that a particular sample origins from a specific line. This is done using Support Vector Machines (SVMs), presented in Section 2.2.3.

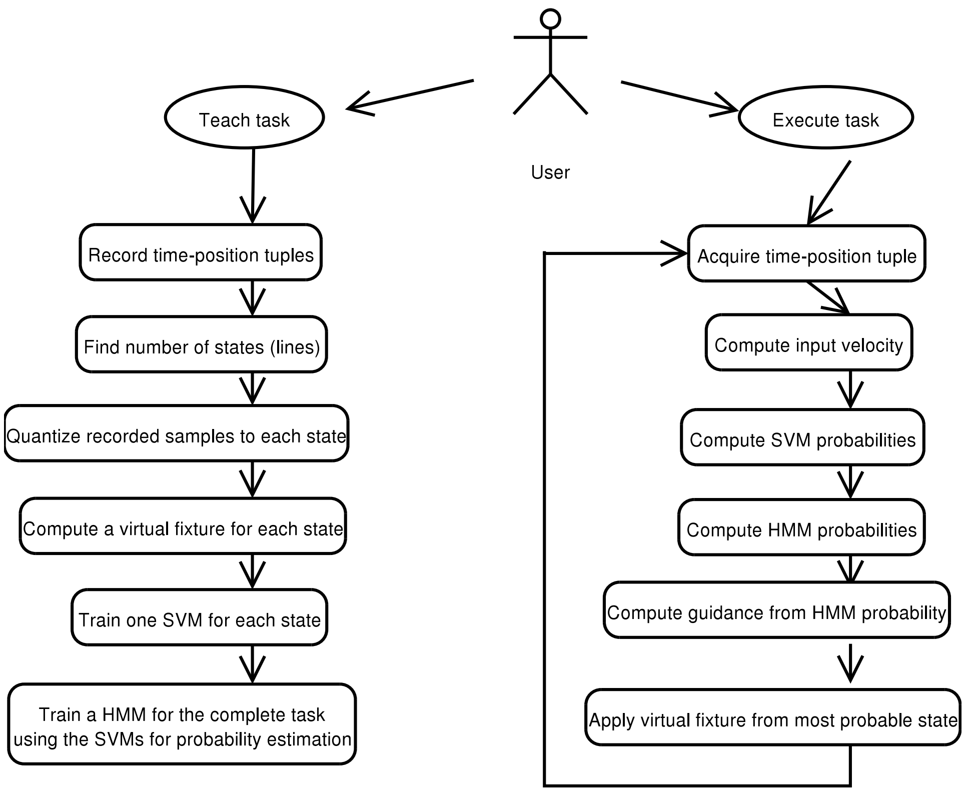

User
Teach task
Execute task
Record time-position tuples
Find number of states (lines)
Quantize recorded samples to each state
Compute a virtual fixture for each state
Train one SVM for each state
Train a HMM for the complete task using the SVMs for probability estimation
Acquire time-position tuple
Compute input velocity
Compute SVM probabilities
Compute HMM probabilities
Compute guidance from HMM probability
Apply virtual fixture from most probable state

Figure 2.1: Overview of the system used for task training and task execution.

State Probability Estimation - Although it is now clear which line is the most probable, the line probability estimation is only based on the information from a single sample. Using Hidden Markov Models (HMMs), a better estimation is achieved using all samples obtained so far. HMMs are described in Section 2.2.1.

Virtual Fixture Guidance - The virtual fixture corresponding to the most probable line segment is applied to aid the user in following the line.

In (Peshkin *et al.*, 2001), the concept of *Cobots* as special purpose human-machine collaborative systems is presented. Although frequently used, the Cobots are designed for a single task and when the assembly task changes, they have to be reprogrammed. We use a combination of K-means clustering, HMMs and SVMs for state generation, state sequence analysis and associated probability estimation. In our system, task segmentation is performed off-line and used by an on-line state estimator that applies a virtual fixture with a fixturing factor determined by the probability of being in a certain state. The use of the HMM/SVM approach is motivated by the good generalization over similar tasks. Our system consists of an off-line task learning and an on-line task execution step.

The system is fully autonomous and is able to i) decompose a demonstrated task into states, ii) compute a virtual fixture for each state and iii) aid the user with task execution by applying the correct virtual fixture at all times.

2.2 Theoretical Background

In a collaborative system, it is important to detect the current state or user's intention to correctly guide the user. Virtual fixtures can be used to constrain the motion of the manipulator through definition of virtual walls and forbidden regions or through definition of a desired directions and trajectories of motions, (Li and Taylor, 2004). Another example of virtual fixtures is to directly constrain the user motion in undesired directions while allowing motion along desired directions using a haptic interface, (Payandeh and Stanisic, 2002). The approach adopted in this work defines a *desired* direction

$$\mathbf{d} \in \mathbf{R}^3,\ \|\mathbf{d}\| = 1$$

or the *span* of the task, (Li and Taylor, 2004; Kragic *et al.*, 2005). The user's input, which may be force, position or velocity measurements, is transformed to a desired velocity $\mathbf{v_{user}}$. The desired velocity is divided into normal and orthogonal components and scaled by a fixturing factor k as shown by (2.1). The fixturing factor determines the compliance of the system. A high value ($\simeq 1$) of k defines a *hard* fixture, i.e. only motion in the direction of the fixture is allowed (low compliance). A value of $k = 0.5$ is in our notation equivalent to no fixture at all, supporting isotropic motion (high compliance). The output velocity $\mathbf{v}$ of the robot is then obtained by scaling $\hat{\mathbf{v}}$ to match the input speed as shown in (2.2).

$$\hat{\mathbf{v}} = \mathbf{proj_d}(\mathbf{v_{user}}) \cdot k + \mathbf{perp_d}(\mathbf{v_{user}}) \cdot (1-k) \tag{2.1}$$

$$\mathbf{v} = \frac{\hat{\mathbf{v}}}{\|\hat{\mathbf{v}}\|} \cdot \|\mathbf{v_{user}}\| \tag{2.2}$$

2.2.1 Hidden Markov Models

The main idea behind Hidden Markov Models (HMMs) is to integrate a simple and efficient temporal model and the available statistical modeling tools for stationary signals into a mathematical framework. HMMs have been primarily used in speech recognition (Rabiner, 1989) but their use have recently been reported in many other fields. The advantage of HMMs is the introduction of *hidden states* which enables more detailed an accurate modeling of stochastic processes. The user of a HMM does not necessarily need to know what states represent, the method automatically assign probability distributions that fit the training data.

The HMM, denoted by $\lambda = (A, B, \pi)$, is defined by three elements over a collection of N states and M discrete observation symbols:

- A, which is the state transition probability matrix. $A = \{a_{ij}\}$, where a_{ij} is the probability of taking the transition from state i to state j.
- B, which is the observation probability matrix. $B = \{b_i(o_k)\}$, where $b_i(o_k)$ is the probability, $P(o_k|i)$, of observing the kth possible observation symbol out of the total M discrete observation symbols in state i.
- π, which is the initial state probability vector. $\pi = \{\pi_i\}$, where π_i is the probability of starting in state i.

Since a_{ij}, $b_i(o_k)$ and π_i all are probability density functions, they obey the following properties:

$$\begin{aligned}
&a_{ij} > 0,\ b_i(o_k) > 0,\ \pi_i > 0\ i,j = 1,\ldots,N,\ k = 1,\ldots,M \\
&\sum_i^N(\pi_i) = 1 \quad \text{and} \quad \sum_j^N(a_{ij}) = 1,\ i = 1,\ldots,N \\
&\sum_k^M(b_i(o_k)) = 1,\ i = 1,\ldots,N
\end{aligned}$$

To construct a suitable HMM for modeling, we have to select the number of states N, and the number of discrete possible observations M. In addition, the probability density matrices A, B, and π have to be determined by training. The most commonly used method is the Baum-Welch method, which is in iterate process that finds the local maximum given some starting values of A, B, and π.

2.2.2 Probability Estimators for Hidden Markov Models

A problem inherit to HMMs is the choice of the probability distribution for estimating the observation probability matrix B. With continuous input, a parametric distribution is often assumed when $M \gg N$ (Elgammal *et al.*, 2003). Using a parametric distribution, similarities may decrease the performance of the HMM since the real distribution is hidden and the assumption of a parametric distribution is a strong hypothesis on the model (Castellani *et al.*, 2004). Using probability estimators avoids this problem since they compute the observation symbol probability instead of using a look-up matrix or parametric model, (Bourlard and Morgan, 1990). Another advantage is that they allow to use continuous input instead of discrete observation symbols for the HMM. Successful use of probability estimators using multi layer perceptrons (MLP) and Support Vector Machines (SVM) are reported in (Bourlard and Morgan, 1990; Renals *et al.*, 1994). In this work, SVMs are used to estimate the observation probabilities $P(\mathbf{x}|\text{state i})$.

2.2.3 Support Vector Machines

Support Vector Machines (SVM) have been used extensively for pattern classification in a number of research areas (Roobaert, 2001; Rychetsky *et al.*, 1999; Hyunsoo and Haesun,

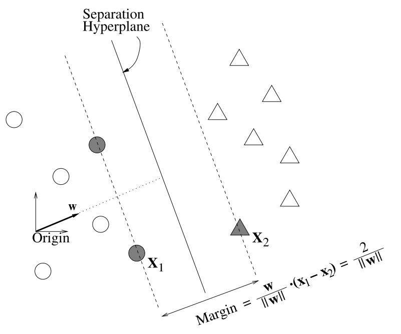

Separation Hyperplane
w
Origin
$\mathbf{x}_1$
$\mathbf{x}_2$
Margin $= \frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = \frac{2}{\|\mathbf{w}\|}$

Figure 2.2: A binary classification example: circles are separated from triangles by a separation hyperplane. The training samples corresponding to the support vectors are marked by filled symbols.

2004). SVMs have several appealing properties such as fast training, accurate classification and good generalization (Chen *et al.*, 2003; Burges, 1998). In short, SVMs are binary classifiers that separate two classes by an optimal separation hyperplane. The separation hyperplane is found by minimizing the expected classification error which is equal to maximizing the distance to the margin as demonstrated in Fig. 2.2.

SVMs work with linear separation surfaces in a Hilbert space (Chen *et al.*, 2003). However, the input patterns are often not linearly separable, or even defined in such a dot-product space. To overcome this limitation, a "kernel trick" is used to transform the input pattern to a Hilbert space (Aizerman *et al.*, 1964). A map ϕ:

$$\phi : \chi \rightarrow \mathcal{H}, x \rightarrow \mathbf{x}$$

is defined for the patterns x from the domain χ. The Hilbert space $\mathcal{H}$ is commonly called the feature space. There are three benefits of transforming the data into $\mathcal{H}$: this makes it possible to define a similarity measure from the dot product in $\mathcal{H}$. In addition, it provides a setting to deal with the patterns geometrically and moreover makes it possible to study learning algorithms using linear algebra and analytic geometry, Finally, it provides the freedom to choose the mapping ϕ which, in its turn, makes it possible to design a large variety of learning algorithms. SVMs try to estimate a function $f : \chi \rightarrow \{\pm 1\}$ that classifies the input $x \in \chi$ to one of the two classes ± 1 based on input-output training data. Vapnik-Chervonenkis (VC) theory shows that it is imperative to restrict the class of functions that f is chosen from, in order to avoid over-fitting.

Let us now consider a class of hyperplanes

$$\mathbf{w} \cdot \mathbf{x} + b = 0, \ \mathbf{w} \in \mathbb{R}^N, \ b \in \mathbb{R}$$

with the corresponding decision function

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b).$$

Among all such hyperplanes there exists a unique one that gives the maximum margin of separation between the two classes, that is (Chen *et al.*, 2003):

$$\max_{\mathbf{w},b}\left(\min(\|\mathbf{x}-\mathbf{x}_i\| : \mathbf{x}\in\mathbb{R}^N,\ \mathbf{w}\cdot\mathbf{x}+b=0,\ i=1,2,...,m)\right)$$

The optimal hyperplane can then be computed by solving the following optimization problem:

$$\begin{aligned}&\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 \text{ over } \mathbf{w},b\\ &\text{subject to} : y_i((\mathbf{w}\cdot\mathbf{x_i})+b)\geq 1,\ i=1,2,...,m\end{aligned}\tag{2.3}$$

One way to solve (2.3) is through the Lagrangian dual:

$$\max_{\alpha\geq 0}\left(\min_{\mathbf{w},b}(L(\mathbf{w},b,\alpha))\right)$$

From the above, it can be shown (Chen *et al.*, 2003) that the hyperplane decision function can be written as

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} y_i\cdot\alpha_i(\mathbf{x}\cdot\mathbf{x})+b\right)$$

which implies that the solution vector **w** has an expansion in terms of a subset of the training samples. The subset is formed by the training samples with a non-zero Lagrange multipliers, α_i. The samples with a non-zero Lagrange multiplier are known as the *support vectors*. The support vectors can easily be computed by solving a quadratic programming problem (Chen *et al.*, 2003).

2.3 Related Work

Approaches similar to ours have been considered in HMCS settings. In (Li and Okamura, 2003), a HMCS system is presented where virtual fixtures facilitate tracking of a curve in two dimensions and a HMM framework estimates whether the user is doing nothing, following or not following the curve. Based on these, the virtual fixture is automatically switched on or off, enabling the user to avoid local obstacles. In (Nolin *et al.*, 2003), different ways of setting the compliance level are described, depending on how well the user is following the fixture. Three different compliance behaviors were evaluated: *toggle*, *fade* and *hold*. The results show that the *fade* behavior, which linearly decreases the compliance with the distance from the fixture, achieves best results when using automatic task detection. In our work, the compliance is adjusted through a fixturing factor presented in the next section. Instead of using the distance to the fixture, the probability that the user is in a certain state is used as a basis for setting the compliance which is one of the contributions of our work.

Commonly, the fixtures are generated from a predefined model of the task which works well as long as the trajectory to be followed in the real world is exactly as described by the model. In robotic applications, the system must be able to deal with model errors and there is a requirement to perform the same type of tasks in terms of *sequencing* but the length (type) of each subtask may vary. Therefore, an adaptive approach in which the trajectory is decomposed into straight lines is evaluated in this work. The system constantly estimates which state the user is currently in and aids the execution. Therefore, it is necessary to decompose the task into several subtasks, recognize the subtasks on-line and handle deviations from the learned task in a flexible manner. For this purpose, HMMs have been used to model and detect state changes corresponding to different predefined subtasks, (Li and Okamura, 2003; Castellani *et al.*, 2004). However, in most cases only one- or two-dimensional inputs have been considered. In our system, the subtasks are automatically detected given the assumption of of straight line motion in 3D and a hybrid HMM/SVM automata is constructed for on-line state probability estimation.

2.4 Trajectory Analysis

This section describes the implementation of the virtual fixture learning system. The virtual fixtures are generated automatically from a number of demonstrated tasks. The overall task is decomposed into several subtasks, each with its own virtual fixture. According to Fig. 2.1, the first step is to filter the input data. Then, a line fitting step is used to estimate how many lines (states) are required to represent the demonstrated trajectory. An observation probability function learns the probabilities of observing specific 3D-vectors when tracking a specific line. Finally, a state sequence analyzer learns what lines are more probable to follow each other. In summary, the demonstrated trajectories results in a number of support vectors, a HMM and a set of virtual fixtures. The support vectors and the HMM are then used to decide when to apply a certain fixture.

2.4.1 Retrieving Measurements

The first task is to obtain measurements from a sensor. The input data consist of a set of 3D-coordinates that may be obtained from a number of sensors, describing a position and time tuple denoted as $\{\mathbf{q}, t\}$. From the input samples, movement directions are extracted. The noisy input samples are filtered using a dead-zone of radius δ around $\mathbf{q}$, i.e. a minimum distance δ since the last stored sample is required so that small variations in position are not captured.

2.4.2 Estimating Lines in the Demonstrated Trajectories

Once the task has been demonstrated, the input data is quantized in order to segment different lines. The input data consists of normalized 3D-vectors representing directions and K-means clustering (MacQueen, 1967) is used to find the lines. For convenience, the method is presented below. The position of a cluster center is equal to the direction of the corresponding line. Given a trajectory, the number of lines required to represent it has to be estimated automatically. For this purpose a search method is used that evaluates the result for different number of clusters and then chooses the quantization with the best results.

Prior to clustering, two thirds of the data points are stored for validation. These are used to measure how well the current clusters represent unseen data. We estimate an optimal number of clusters that maximizes the validation score for the unseen data. The algorithm starts with a single cluster and gradually increases the number of clusters by one as long as the validation score increases. However, more clusters typically give a lower error, so a penalty is given proportional to the number of clusters to facilitate a simple solution.

2.4.2.1 K-means Clustering

K-means clustering is an algorithm for partitioning N L-dimensional data points into K disjoint subsets, while minimizing the squared distance over all data points and their closest cluster center. The algorithm consists of a simple iteration procedure as follows. Initially, the cluster centers are distributed randomly on the L-dimensional space. In the first step, each point is assigned to the cluster whose centroid is closest to that point. In the next step, each centroid is moved to the mean position of that data points assigned to it. These two steps are repeated until the cluster center positions have stabilized. This is a simple, yet efficient method of obtaining good quantization of data.

2.4.3 Estimating Observation Probabilities Using Support Vector Machines

For each state detected by the clustering algorithm, a SVM is trained to distinguish it from all the others (one-vs-all). In order to provide a probability estimation for the HMM, the distance to the margin from the sample to be evaluated is computed as (Castellani *et al.*, 2004):

$$f_j(\mathbf{x}) = \sum_i \alpha_i \cdot y_i \cdot \mathbf{x} \cdot \mathbf{x_i} + b \tag{2.4}$$

where $\mathbf{x}$ is the sample to be evaluated, $\mathbf{x}_i$ is the i-th training sample, $y_i \in \{\pm 1\}$ is the class of $\mathbf{x}_i$ and j denotes the j-th SVM. The distance measure $f_j(\mathbf{x})$ is then transformed to a conditional probability using a sigmoid function, $g(\mathbf{x})$, (Castellani *et al.*, 2004). The probability for a state i given a sample $\mathbf{x}$ can then be computed as:

$$\begin{aligned} &P(\text{state } i|\mathbf{x}) = g_i(\mathbf{x}) \cdot \prod_{j \neq i} (1 - g_j(\mathbf{x})) \\ &\text{where } g_i(\mathbf{x}) = 1/(1 + e^{-\sigma \cdot f_i(\mathbf{x})}) \end{aligned} \tag{2.5}$$

Given the above and applying Bayes' rule, the HMM observation probability $P(\mathbf{x}|\text{state } i)$ may be computed.

$$P(\mathbf{x}|\text{state } i) = \frac{P(\text{state } i|\mathbf{x})P(\mathbf{x})}{P(\text{state } i)} \tag{2.6}$$

We assume equal unconditional probabilities for all states and observations, and thus $P(\mathbf{x})/P(\text{state } i)$ is constant. The SVMs now serve as probability estimators for both the HMM training and state estimation. Since the standard SVMs do not cope well with outliers, a modified version of SVMs is used (Cortes and Vapnik, 1995).

2.4.4 State Sequence Analysis Using Hidden Markov Models

Even if a task is assumed to consist of a sequence of line motions, in an on-line execution step, the lines may have different lengths compared to the training data. Hence, it is not possible to exactly follow the training trajectory. When a certain line is followed, it is assumed that the corresponding line state is active. Thus, there are equally many states as there are line directions. Given that a certain state is active, some states are more likely to follow each other depending on the task and, in our system, a fully connected Hidden Markov Model is used to model the task. The number of states is equal to the number of line types found in the training data. The A-matrix is initially set to have probability 0.7 to remain in the same state and a uniformly distributed probability to switch state. The π vector is set to uniformly distributed probabilities, meaning that all states are equally probable at the start time. For training, the Baum-Welch algorithm is used until stable values are achieved.

With each line, there is an associated virtual fixture defined by the direction of the line. In order to apply the correct fixture, the current state has to be estimated. The system continuously updates the state probability vector $\mathbf{p} = p_i$, where $p_i = P(\mathbf{x_k}, \mathbf{x_{k-1}} \ldots, \mathbf{x_1} | \text{state } i)$ is calculated according to

$$\begin{aligned} \hat{p}_i &= \begin{cases} \pi_i \cdot P(\mathbf{x}|\text{state } i) & \text{if } \mathbf{p}^{\text{last}} = \mathbf{0} \\ P(\mathbf{x}|\text{state } i) \cdot \sum_j^{Nstates} A_{ij} \cdot p_j^{last} & \text{otherwise} \end{cases} \\ p_i &= \hat{p}_i / \sum_j^{Nstates} \hat{p}_j \end{aligned} \tag{2.7}$$

The state s with the highest probability p_s is chosen and the virtual fixture corresponding to this state is applied with the fixturing factor $k = \max(0.5,\ p_s \cdot \xi),\ \xi \in [0,1]$, where $p_s = \max_i\{p_i\}$ and ξ is the maximum value for the fixturing factor. As shown in (2.1), the fixturing factor describes how the virtual fixture will constrain the manipulator's motion. In the case of a haptic input device, the fixture can also be used to provide the necessary feedback to the user and not only constraining the motion of the teleoperated device. Thus, when unsure which state the user is currently in, the user has full control over the system. On the other hand, when all observations indicate a certain state, the fixturing factor k is set to ξ. This automatic adjustment of the fixturing factor allows the user to leave the fixture and move freely without having a special "not-following-fixture"-state.

2.5 Experimental Evaluation

In this section, three experiments are presented. The first experiment is a simple trajectory tracking task in a workspace with obstacles, shown in Fig. 2.3. The second is similar to the first one, but the workspace was changed *after* training, in order to test the algorithm's automatic adjustment to similar workspaces. In the last experiment, an obstacle was placed

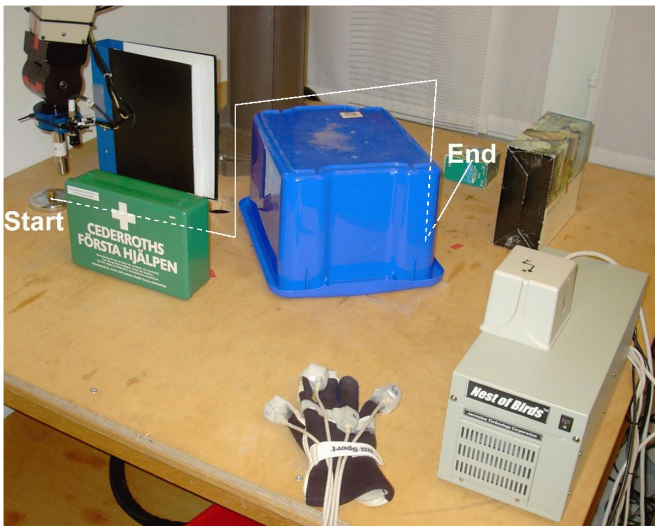


Figure 2.3: The experimental workspace with obstacles: the white line shows the expected path of the end-effector.

along the path of the trajectory, forcing the operator to leave the fixture. This experiment tested the adjustment of the fixturing factor as well as the algorithm's ability to cope with unexpected obstacles.

In the experiments, a teleoperated setting was considered. The PUMA 560 robot was controlled via a magnetic tracker called Nest of Birds (NOB) (Ascension Tech., 2006) mounted on a data-glove carried by the user. The NOB consists of a transmitter and pose measuring sensors. The glove with the sensors can be seen in the lower part of Fig. 2.3 - there is one sensor mounted on a thumb, index and a little finger and the fourth sensor is placed in the middle of the hand. In the experiments, only the hand sensor is used since it provides the full position and orientation estimate of the user's hand motion. Subtask recognition is performed with a frequency of 30 Hz due to the limited sampling rate of the NOB sensor. The movements of the operator measured by the NOB sensor were used to extract a desired input velocity to the robot. After applying the virtual fixture according to (2.2), the desired velocity of the end effector is sent to the robot control system. Controlling the end-effector manually in this way is hard, but the experiments will show that the use of virtual fixtures makes the task easier. The system also works well with other input modalities. For instance, a force sensor mounted on the end effector has also been used to control the robot.

In all experiments, a dead-zone of $\delta = 2$ cm was used. This value of δ corresponds to the approximate noise level of our input device. One of the major difficulties of the system is that the input device provides no haptic feedback. Therefore, the virtual fixture framework is used to filter out sensor noise and correct unintentional operator motions.

This is done by scaling down the input velocity that is perpendicular to the desired direction of the virtual fixture as long as the commanded motions is along the general direction of the learned fixture.

In all experiments, a maximum fixturing factor was $\xi = 0.8$. A radial basis function with $\sigma = 2$ was used as the kernel for the SVMs and the value of σ in the sigmoid transfer function (2.5), was empirically chosen to 0.5.

2.5.1 Experiment 1: Trajectory Following

The first experiment was a simple trajectory following task in a narrow workspace. The user had to avoid obstacles and move along certain lines to avoid collision. At start, the operator demonstrated the task five times, the system learned from training data and four states were automatically identified. An example training path is shown in Fig. 2.4. The user then performed the task again using the glove, the states were automatically recognized and the robot was controlled aided by the virtual fixtures generated from the training data. The path taken by the robot is shown in Fig. 2.5. For clarity, the state probabilities and fixturing factor estimated by the SVM and HMM during task execution are presented in Fig. 2.8. This example clearly demonstrates the ability of the system to successfully segment and repeat the learned task, allowing a flexible state change.

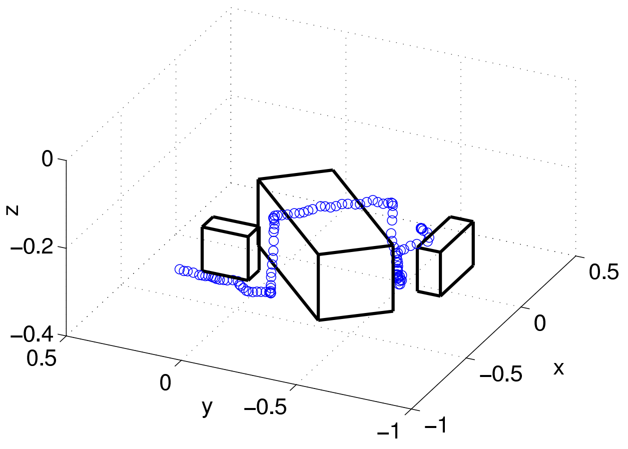


Figure 2.4: A training example demonstrated by the user. This example was used for training the robot in all experiments.

Initially, the end-effector is moving along the y-axis, corresponding to the direction of state 3. Because of deviations from the state direction, the SVM probability will fluctuate since its estimation is based on the distance from the decision boundary. However the HMM probability remains steady due to the estimation history. This shows the advantage of using a HMM on top of SVM for state identification. At sample 24, the user switches direction and starts raising the end-effector. The fixturing factor decreases with

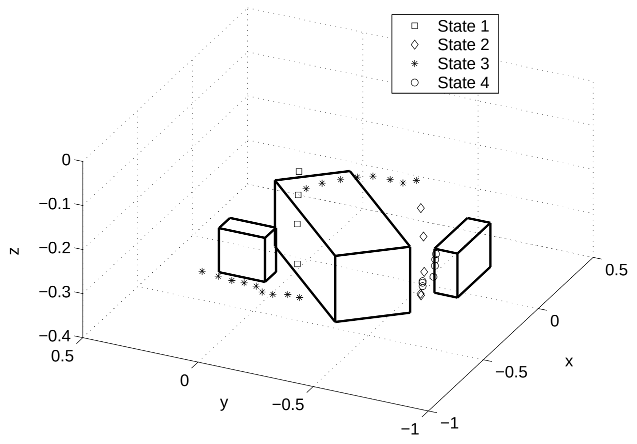


Figure 2.5: End effector position when following the trajectory using virtual fixtures. The different symbols corresponds to the different states recognized by the HMM.

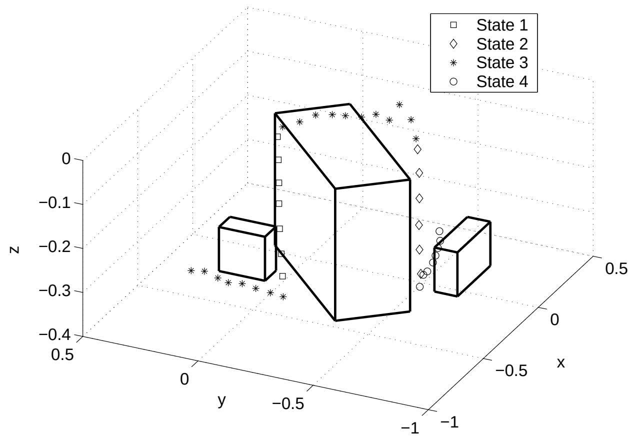


Figure 2.6: Same as Fig. 2.5, but in a modified workspace compared to training.

the probability for state 3, simplifying the direction change. Then, the probability for state 1, corresponding to movement along the z-axis, increases. In total, the user performed 4 state transitions in the experiment.

2.5.2 Experiment 2: Changed Workspace

This experiment demonstrates the ability of the system to deal with a changed workspace. The same training trajectories as in the first experiment were used, but the workspace was

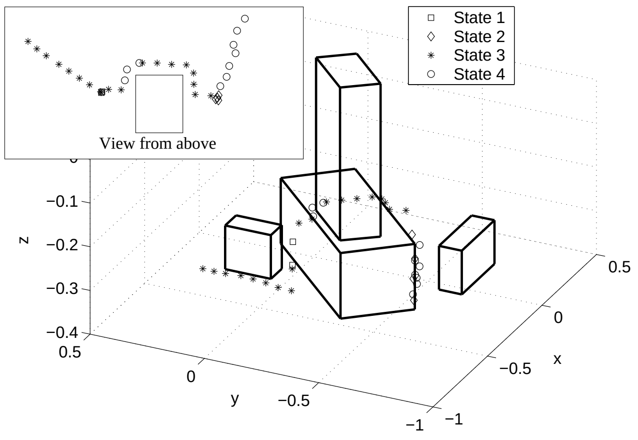


Figure 2.7: Same as Fig. 2.5, but with an unexpected obstacle not present during training.

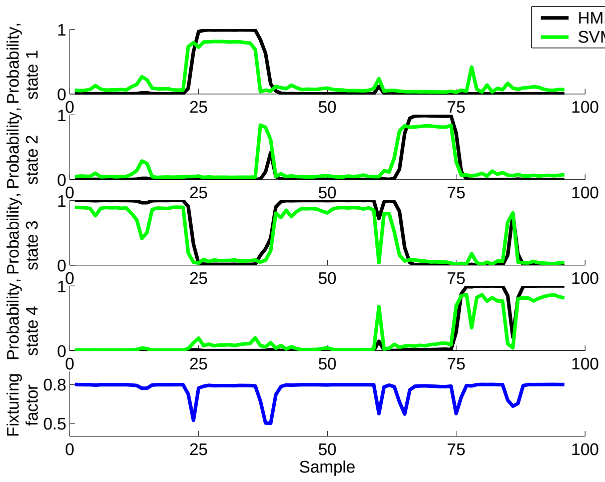


Figure 2.8: Estimated probabilities for the different states in experiment 1. Estimates are shown for both the SVM and HMM, the fixturing factor is also shown.

changed after training. As it can be seen in Fig. 2.6, the size of the obstacle the user has to avoid has been changed. As the task is just a variation of the trained task, the system is still able to identify the operator's intention and correct unintentional operator motions. The trajectory generated from the on-line execution shows that the changed environment does not introduce any problem for the control algorithm since an appropriate fixturing factor is provided at each state. This clearly justifies the proposed approach compared to the work previously reported in (Peshkin *et al.*, 2001).

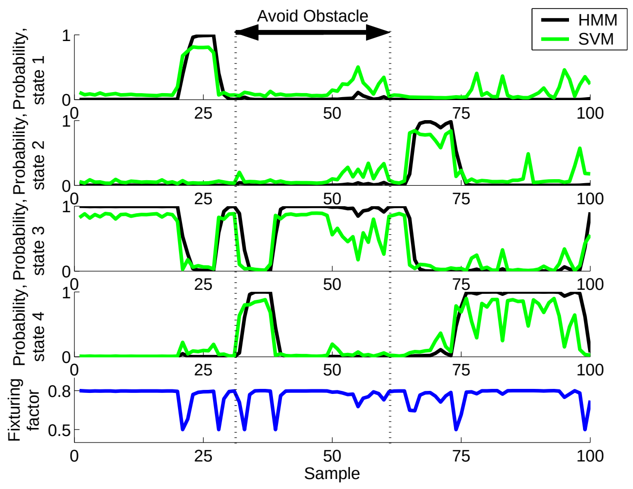


Figure 2.9: Estimated probabilities for the different states in the obstacle avoidance experiment 3. Estimates are shown for both the SVM and HMM estimator.

2.5.3 Experiment 3: Unexpected Obstacle

The final experiment was conducted in the same workspace as the first one. However, this time a new obstacle was placed right in the path of the learned trajectory, forcing the operator to leave the fixture. In this case, the virtual fixture is not aiding the operator, but may instead do the opposite as the operator wants to leave the fixture in order to avoid the obstacle. Hence, in this situation it is desired that the effect of the virtual fixture decreases as the operator avoids the obstacle. Once again, the same training examples as in the previous experiments were used.

Fig. 2.7 illustrates the path taken in order to avoid the obstacle. The system always identifies the class which corresponds best with input data. Fig. 2.9 shows the probabilities and fixturing factor for this experiment. Initially, the task is identical to experiment 1, the user follows the fixture until the obstacle has to be avoided. The fixturing factor decreases as the user diverts from the direction of state 3, and thus the user is able to avoid the obstacle. It can be seen that the overall task has changed and that new states were introduced in terms of sequencing. The proposed system not only provides the possibility to perform the task, but can also be used to design a new task model by demonstration if this particular task has to be performed several times.

2.6 Discussion

We have presented a system based on the use of *adaptive virtual fixtures*. It is widely known that one of the important issues for robot control systems is the ability to divide the overall task into subtasks and provide the desired control in each of them. We have shown that it is possible to use a HMM/SVM hybrid state sequence analyzer on multi-dimensional data to obtain an on-line state estimate that can be used to apply a virtual fixture. Furthermore, the process is automated to allow construction of fixtures and task segmentation from demonstrations, making the system flexible and adaptive by separating the task into subtasks. Hence, model errors and unexpected obstacles are easily dealt with.

In this chapter we only provide qualitative experiments, and it might be interesting to add some quantitative experiments and measure how much stability is gained using the fixtures. The user could be told to steer the end effector along a predefined path, for example a drawing on a piece of paper. The result could then be compared to the true path. However, it has previously been shown that virtual fixtures increase the overall performance (Payandeh and Stanisic, 2002), and there is no reason to believe that this approach is an exception. The goal of this chapter was to make the traditional virtual fixtures more adaptive.

In the current design, the algorithm automatically estimates the number of subtasks required to divide the training data. If instead it is possible for the user to manually select the number of states, the algorithm may be expected to perform even better. Such approach may be, for example, used in medical applications since surgical tasks are expected to be well-defined and known in advance (Kragic *et al.*, 2005).

In terms of the HMMs, the design of the transition matrix A depends on the properties of the task. Although the transition matrix is trained, only a local solution is found so the overall performance varies for different A matrices. We have used a fully connected transition matrix, which allows each state to jump to any other state. This is the most general design type and is applicable to virtually any task. However, if the nature of the task is so that certain transitions are impossible, they should be encoded as zeroes in the matrix. We also experimented with the more common left-to-right matrix design, meaning that once the state has changed, it cannot change back to the previous. We found that the design did not work in this case, as if the user did a movement by mistake, triggering the state change, it was impossible to return to the correct state.

A motivated question is for which tasks the decomposition into lines is applicable. A natural extension to this work is to add second order components like the arc of a circle. Still, a circle can be approximated by a sequence of lines, so adding complex shape primitives may only reduce the tracking error.

Focus in this work was on machine guidance in a human machine collaborative system, but the method presented is not restricted to that setting. As will be presented in Chapter 4, HMMs are usually used for recognition. This is done by calculating the probability for the entire observation sequence. Several different demonstrated tasks can easily be recognized using this method, by creating HMM models for each of them and comparing the posterior probabilities.

The method described in this chapter is an example of direct learning. The robot learns the task while it is being remotely controlled by the human. No mapping from human to robot body is necessary, and sensing is very precise. The following chapters instead focuses on indirect learning. In the next chapter, we investigate methods for robot vision to enable remote sensing and learning from observations.

Chapter 3

Robot Vision for Indirect Learning

As we move from direct learning to indirect learning, a fundamental requirement is put upon the robot. It has to be able to utilize remote sensing to understand the world outside its own body. This way, the robot can learn tasks just by observing a human performing them. There are several ways of performing remote sensing, e.g., using sonars and laser scanners, but vision is certainly the method which provides the most information. Vision is also the most important sense used by humans to understand the surrounding. Furthermore, vision plays a vital role in human learning. Clearly, a robot that is to learn from human demonstrations must also be equipped with some form of vision capabilities. Here, the most relevant ability is detecting and recognizing objects.

Object recognition is a large research area in both robotics and in computer vision, and numerous methods have been proposed. A few major attributes define the differences between them:

- **Appearance/Feature-Based**
  The representation of an object can either be based on the *appearance* of the object, calculated over the entire training image, or on specific features of the object, calculated over several small image patches at key locations in the training image. In general, feature-based methods have a lower false positive rate and higher scalability, but can only recognize *instances* of objects, not object categories. The objects must be textured to ensure that enough key points are found. Also, they work mainly on rigid object, as on deformable objects key points may change appearance and position.

- **Translation-, Rotation- and Scale Invariance**
  Translation invariance is a requirement for detecting objects that are not pre-segmented. The algorithm must produce about the same result regardless if the object is shifted in the image plane. Rotation invariance means that the algorithm gives the same result when the object is rotated in the image plane. Since the robot in most cases know what pose the object it is looking for has, it is desired that it is able to detect the object even if it is rotated in an angle never encountered before. However, for

some applications it may be necessary to separate between an object lying down or standing up. Scale invariance is also an important property. In uncontrolled environments it is quite unlikely that the object will be found in the same scale as the robot was trained on.

- **Occlusion Robustness**
  Often there are several objects on top and in front of each other in a cluttered scene. The ability to detect and recognize objects that are only partly visible is often desired.

- **Scalability**
  In general, the more objects the robot needs to recognize, the worse the recognition rate will be. A method with high scalability will have a low reduction in recognition rate as the number of objects increases.

- **Training Complexity**
  Before a method can be used for recognizing objects, it has to be trained. In general, the more images available for training, the better the results. However, for a robot that is to learn recognition from human demonstrations, it is desired that the robot is able to learn from only a few training images since it is tedious to show the robot the same object over and over again. Also, training time should be fast.

- **Computational Efficiency**
  Most applications require a quick response from the vision system. A learning scenario is no exception, as the robot has to be able to interpret a live demonstration. During the last few years the computational processing power of an average computer has increased, and systems with real-time performance have emerged.

An object recognition system is typically designed to classify an object to one of several predefined classes assuming that the segmentation of the object from the background has already been performed. The task for an object detection algorithm is much harder. Its purpose is to search for a specific object in an image of a complex scene. Most of the object recognition algorithms may be used for object detection by using a search window and scanning the image for the object.

3.1 System Overview

In most of the recognition methods reported in the literature, a large number of training images are needed to recognize objects viewed from arbitrary angles. The training is often performed off-line and, for some algorithms, it can be very time consuming. For robotic applications, it is important that new objects can be learned easily. i.e. putting a new object in the database and retraining should be fast and computationally cheap. Our goal in the work reported in this chapter is to develop an on-line learning scheme that can be effective after just one training example but still has the ability to improve its performance with more examples. Also, learning new objects should be possible without heavy recalculations on already learned objects. We believe the method is general enough to be easily used for a variety of applications requiring robust object detection.

Our vision system is a combination of several modules working together, as depicted in Fig. 3.1. The system is able to detect, recognize and estimate the pose of an object.

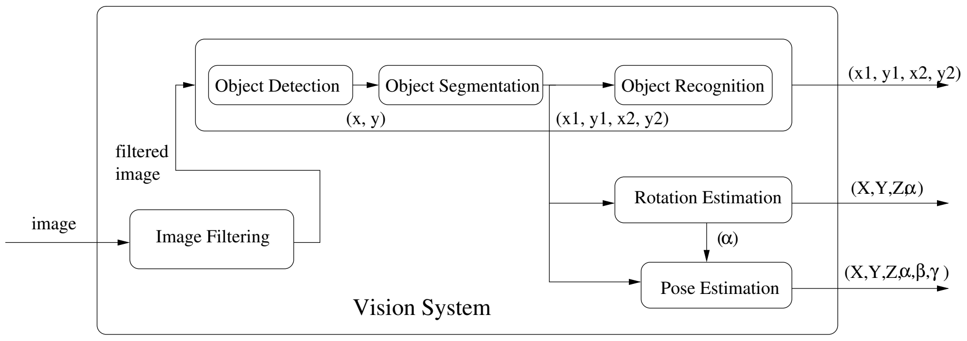


Figure 3.1: An overview of our vision system. Most vision systems in the literature rely on image filtering as a first step, to detect edges and corners at various scales. Then, the object is detected and segmented, and the image coordinates are fed to a recognition algorithm (some algorithms perform recognition without segmentation). Depending on the application, the image coordinates may be enough, but if interaction with the object is desired, a pose estimation algorithm is usually necessary. A model based pose estimation is able to provide all six degrees of freedom of the object, including the orientation and the true (X,Y,Z) world position relative to the camera.

We initially started to work on object detection using Color Cooccurrence Histograms (CCH), presented in Section 3.3. However, we found that the pure color-based method was much too sensitive to lighting conditions. In Section 3.4 we present an extended object detection system, which utilizes Receptive Field Cooccurrence Histograms (RFCH). The two methods are then evaluated in Section 3.5. In Section 3.6 we show how an appearance-based method can be used to estimate the pose of an object.

In terms of the properties listed in the beginning of this chapter, the presented method is an appearance-based method which is translation- and rotation invariant, and robust to scale changes. It also copes well with object occlusions. The scalability has not been explicitly evaluated, but the method performs well for 10 object instances, although the results are expected to degrade as more objects are added. The method has a low training complexity and a high computational efficiency.

3.2 Related Work

The field of computer vision is enormous. Here, we focus on methods which can be used to detect object instances in cluttered environments such as homes or offices. In terms of object recognition, the appearance-based representations are commonly used, (Murase and Nayar, 1995; Selinger and Nelson, 2001; Caputo, 2004). However, the appearance-based methods suffer from various problems. For example, a representation based on the color of an object is sensitive to varying lighting conditions, while a representation based on the shape of an object is sensitive to occlusion. These drawbacks motivated us to develop a new detection system to cope with these problems. As we demonstrate in Section 3.5, the RFCH-method robustly copes with both of the mentioned problems. This property makes the algorithm ideal for use on robotic platforms which are to operate in natural scenes.

Back in 1991, Swain and Ballard (1991) demonstrated how RGB color histograms can be used for object recognition. Schiele and Crowley (2000) generalized this idea to histograms of receptive fields, and computed histograms of either first-order Gaussian derivative operators or the gradient magnitude and the Laplacian operator at three scales. Linde and Lindeberg (2004) evaluated more complex descriptor combinations, forming histograms of up to 14 dimensions. Excellent performance on both the COIL-100 and the ETH-80 database were shown. Mel (1997) also developed a histogram based object recognition system that uses multiple low-level attributes such as color, local shape and texture. Although these methods are robust to changes in rotation, position and deformation, they cannot cope with recognition in a cluttered scene. The problem is that the background visible around the object confuses the methods. Chang and Krumm (1999) show how color cooccurrence histograms can be used for object detection, performing better than regular color histograms.

The methods mentioned so far are appearance-based methods, meaning that they calculate the object representation on all available image data. In contrast, local feature-based methods only capture the most representative parts of an object. Lowe (1999) presents the SIFT features, which is a promising approach for detecting objects in natural scenes. However, the method relies on the presence of feature points and, for objects with simple or no texture, this method fails. The method also requires a high resolution camera, or that the object occupies a rather large part of the image.

3.3 Color Cooccurrence Histograms

A Color Histogram is a statistical representation of the occurrence of colors an image. A Color Cooccurrence Histogram (CCH) is able to capture more of the geometric properties of an object. Instead of just counting each pixel's color value, the histogram is built from *pairs* of pixels. The pixel pairs can be constrained based on, for example, their relative distance. This way, only pixel pairs separated by less than a maximum distance, d_{max} are considered. Thus, the histogram represents not only how common a color is in the image but also how common it is that certain combinations of color occur close to each other.

3.3.1 Image Normalization

The appearance of colors in an image is highly affected by the illumination. To make the representation more robust to illumination changes colors are normalized according to

$$r_{norm} = \frac{r}{r+g+b},\ g_{norm} = \frac{g}{r+g+b}$$

Another alternative is to use the HSV color space and only use the hue and saturation, which give similar results.

3.3.2 Image Quantization

Using one histogram bin for each color may result in a very sparse histogram. Instead, the image is first quantized using K-means clustering (MacQueen, 1967). Each normalized pixel is quantized to one of N 2-dimensional cluster centers in rg-space. As distance measure, we use the Euclidean distance in the color space. That is, each cluster has the shape of a circle. input dimensions to be of the same scale, otherwise some descriptors would be favored. Thus, we scale all descriptors to the interval [0,255]. The clusters are randomly initialized, and a cluster without members is relocated just next to the cluster with the highest total distance over all its members. After a few iterations, this leads to a shared representation of that data between the two clusters. Each object ends up with its own cluster scheme in addition to the RFCH calculated on the quantized training image.

When searching for an object in a scene, the image is quantized with the same cluster-centers as the cluster scheme of the object being searched for. Quantizing the search image also has a positive effect on object detection performance. Pixels lying too far from any cluster in the descriptor space are classified as the background and not incorporated in the histogram. This is because each cluster center has a radius that depends on the average distance to that cluster center. More specifically, if a pixel has a Euclidean distance d to a cluster center, it is not counted if $d > \alpha \cdot d_{avg}$, where d_{avg} is the average distance of all pixels belonging to that cluster center (found during training), and α is a free parameter. We have used $\alpha = 1.5$ i.e., most of the training data is captured. $\alpha = 1.0$ corresponds to capturing about half the training data.

The quantizing of the image can be seen as a first step that simplifies the detection task. To maximize detection rate, each object should have its own cluster scheme. This, however, makes it necessary to quantize the image once for each object being searched for. If several different objects are to be detected and a very fast algorithm is required, it is better to use shared cluster centers over all objects known. In that case, the image only has to be quantized once.

It has to be noted that multiple histograms of the object across a number of training images may share the same set of cluster centers.

3.3.3 Histogram Matching

The similarity between two normalized CCHs is computed as the histogram intersection:

$$\mu(h_1, h_2) = \sum_{n=1}^{N^2} \min(h_1[n], h_2[n]) \tag{3.1}$$

where $h_i[n]$ denotes the frequency of pixels pairs in bin n for image i, quantized into N cluster centers. Note here that the histogram size is N^2 since we use cooccurrence histograms. The higher the value of $\mu(h_1,h_2)$, the better the match between the histograms. Prior to matching, the histograms are normalized with the total number of pixel pairs.

Another popular histogram similarity measure is the χ^2:

$$\mu(h_1,h_2) = \sum_{n=1}^{N^2} \frac{(h_1[n]-h_2[n])^2}{h_1[n]+h_2[n]} \tag{3.2}$$

In this case, the lower value of $\mu(h_1,h_2)$, the better the match between the histograms. The χ^2 similarity measure usually performs better than the histogram intersection method on object recognition image databases. However, we have found that χ^2 performs much worse than histogram intersection when used for object detection. We believe that this is because the background that is visible in the search window and not present during training, is not penalizing the match correspondence as much as with the χ^2. Histogram intersection focuses on bins that represent the searched object best, while χ^2 treats all bins equally. As mentioned, χ^2 still performs slightly better on object recognition databases. In these databases there is often only a black background, or even worse, the background provides information about the object (e.g., airplanes shown on a blue sky background).

3.3.4 Object Detection and Segmentation

Object detection is a more challenging task than object recognition. Usually, the object only occupies a small area of the image and object recognition algorithms cannot be run directly considering the entire image. Instead, the image is scanned using a small search window. The window is shifted such that consecutive windows overlap to 50 % and the RFCH of the window is compared with the object's RFCH according to (3.1). Each object may be represented by several histograms if its appearance changes significantly with the view angle of the object. However, in this work we only used one histogram per object.

The matching vote $\mu(h_{object}, h_{window})$ indicates the likelihood that the window contains the object. Once the entire image has been searched through, a vote matrix provides a hypothesis of the object's location. Fig. 3.2 shows a typical scene from our experiments together with the corresponding vote matrix for the yellow soda can. The vote matrix reveals a strong response in the vicinity of the object's true position close to the center of the image.

The vote matrix may then be used to segment the object from the background, as described below, or just provide an hypothesis of the object's location. The most probable location is corresponding to the vote cell with the maximum value.

3.3.4.1 Object Segmentation

The local maxima in the vote matrix serve as starting points to initiate the identification of candidate windows. Each window is iteratively expanded by adjacent rows or columns, as long as the new cells give sufficient support for the object. The expansion process stops

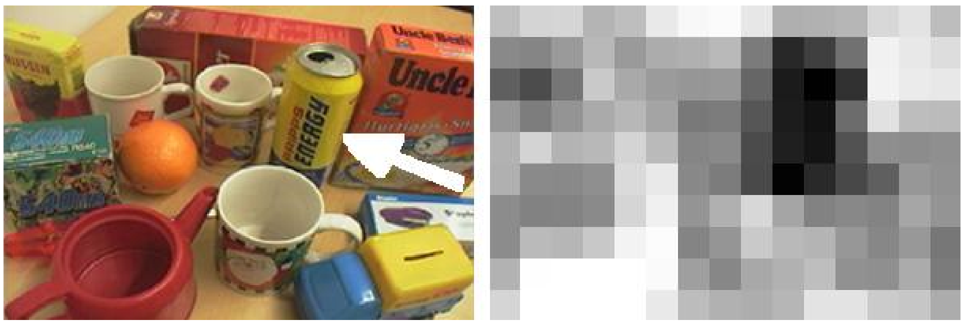


Figure 3.2: Example of searching for the yellow soda can, placed closed to the center of the image. Dark areas indicate high likelihood of the object being present. There is a strong response where the soda can is placed, but also a small response at the location of the raisin box, standing to the left in the image, because of the similar yellow color. Note that this image was generated using the more advanced Receptive Field method, described later in this chapter.

when the ratio between the average vote in the border cells and the local maxima vote becomes falls short of the threshold Φ. In principle, the optimal threshold value Φ depends on the objects color distribution and texture. If the threshold is too high, parts of the object may be omitted. If the threshold is too low, the window contains too much background that reduces the signal to noise ratio in the subsequent image processing steps. An experimental evaluation of different threshold values showed that our algorithm achieves similar performance for a range of $\Phi \in [0.3, 0.6]$, as shown in Fig. 3.17.

3.4 Receptive Field Cooccurrence Histograms

To improve the detection and recognition rate, we have extended CCHs to take advantage of the information in the local gradient field of an image. We denote this improved representation a Receptive Field Cooccurrence Histogram (RFCH). Instead of representing just colors, the cooccurrence of several image descriptor responses within an image are counted. Examples of such image descriptors are color intensity, gradient magnitude and Laplace response, described in detail below.

3.4.1 Image Descriptors

We will evaluate the performance of histogram-based object detection using different types of image descriptors. The descriptors we use are all rotationally and translationally invariant. If rotational invariance is not required for a particular application, increased recognition rate could be achieved by using Gabor filters. In short, we will consider the following basic types of image descriptors, as well as various combinations of these:

- **Normalized Colors**
  The color descriptors are the intensity values in the red and green color channels, in normalized RG-color space, according to $r_{norm} = \frac{r}{r+g+b}$ and $g_{norm} = \frac{g}{r+g+b}$.

- **Gradient Magnitude**
  The gradient magnitude is a differential invariant, and is described by the combination of partial derivatives (L_x, L_y): $|\nabla L| = \sqrt{L_x^2 + L_y^2}$. The partial derivatives are calculated from the scale-space representation $L = g * f$ obtained by smoothing the original image f with a Gaussian kernel g, with standard deviation σ.

- **Laplacian**
  The Laplacian is an on-center/off-surround descriptor. Using this descriptor is biologically motivated, as it is well known that center/surround ganglion cells exist in the human brain. The Laplacian is calculated from the partial derivatives (L_{xx}, L_{yy}) according to $\nabla^2 L = L_{xx} + L_{yy}$. From now on, $\nabla^2 L$ denotes calculating the Laplacian on the intensity channel, while $\nabla^2 L_{rg}$ denotes calculating it on the normalized color channels separately.

3.4.2 Image Quantization

Regular multidimensional receptive field histograms (Schiele and Crowley, 2000) have one dimension for each image descriptor. This makes the histograms huge. For example, using 15 bins in a 6-dimensional histogram means 15^6 ($\sim 10^7$) bin entries. As a result the histograms are very sparse, and most of the bins have zero or only one count. Building a cooccurrence histogram makes things even worse, in that case we need about 10^{14} bin entries. By first clustering the input data, a dimension reduction is achieved. Hence, by choosing the number of clusters, the histogram size may be controlled. In this work, we have used 80 clusters resulting in that our cooccurrence histograms are dense and most bins have high counts.

Dimension reduction is performed much like described in Section 3.3.2. Each pixel is quantized to one of N cluster centers. The cluster centers have a dimensionality equal to the number of image descriptors used. For example, if both color, gradient magnitude and the Laplacian are used, the dimensionality is six (three descriptors on two colors). As distance measure, we use the Euclidean distance in the descriptor space. That is, each cluster has the shape of a hypersphere. This requires all input dimensions to be of the same scale, otherwise some descriptors would be favored. Thus, we scale all descriptors to the interval [0,255].

Fig. 3.3 shows an example of a quantized search image, when searching for a red, green and white Santa-mug.

3.4.3 An Alternative Segmentation Approach

While the segmentation method described in Section 3.3.4.1 is very fast, it has the drawback that the object searching is only performed on a single scale. Most of the time only

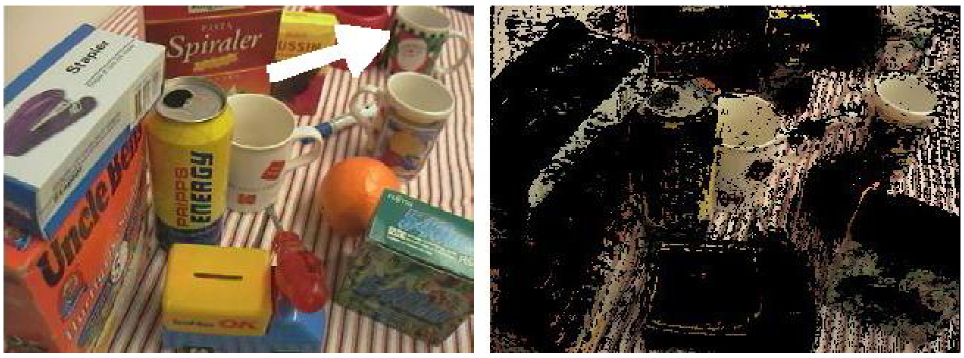


Figure 3.3: Example when searching for the Santa-mug, visible in the top right corner. Left: The original image. Right: Pixels that survive the cluster assignment. The pixels that lie too far away from their nearest cluster are ignored (set to black in this example). The red striped table cloth still remains, as the Santa-mug contains red-white edges.

a small part of the object in the test image will be matched to the entire object from the training image. Another approach is to compute the histogram over a multitude of squares in the image, each with a different size and position. Then, the square with the highest vote corresponds to the object location, and thus segmentation is given by the square borders. In (Viola and Jones, 2001), the *integral image* is presented. With this representation, a large number of squares can be evaluated in a very short time.

3.4.3.1 Integral Image

The integral image is a collection of histograms calculated from the top left corner to specific points in the image. This allows for quick calculations of histograms over arbitrary squares in the image. Building the integral image do not take much time, as it is a dynamic programming method in that each histogram uses the previous histogram and just add the new pixels. Then, the histogram of the square ABCD in Fig. 3.4 can quickly be calculated as A+D-B-C.

An integral image of a cooccurrence histogram is slightly different since each histogram contains some pixels outside its border. However, with this in mind correct calculations can still be made on the integral image.

3.4.3.2 Histogram Boosting

Another technique presented by Viola and Jones (2001) is to add boosting (Freund and Schapire, 1995) to build a more robust representation. Although Viola and Jones (2001) did not use color based histograms, the technique can easily be used with RFCHs. In short, the object is represented not only by a single histogram, but with many histograms covering different parts of the object. Thus, specific features may be dominant in histograms over

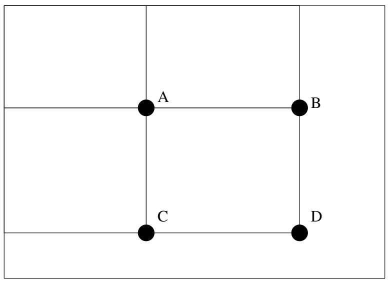


Figure 3.4: By using the integral image, the histogram for the square ABCD can be calculated very quickly as A+D-B-C.

small areas, while for a histogram over the entire object those features are barely visible due to the huge amount of data. Boosting is used to automatically find out which areas of the object that best represents it. In addition, each area is assigned a voting strength that indicates its importance. However, the approach also comes with a number of drawbacks, which we present in the experimental evaluation.

3.4.4 Complexity

The running time can be divided into three parts. First, the test image is quantized. The quantization time has complexity $O(N)$. Second, the histograms are calculated. The calculation time has complexity $O(d_{max}^2)$. Last, the histogram similarities are calculated. Although histogram matching is a fast process, its running time has complexity $O(N^2)$.

The algorithm is very fast which makes it applicable even on mobile robots. Depending on the number of descriptors used and the image size, the algorithm implemented in C++ runs at about 3-10 Hz on a 3 GHz regular PC.

3.5 Object Detection Evaluation

We evaluate six different descriptor combinations in this section. The descriptor combinations are chosen to show the effect of the individual descriptors as well as the combined performance. The descriptor combinations are:

- $[R, G]$ - Capturing only the absolute values of the normalized red and green channel. Corresponding to a color cooccurrence histogram. With $d_{max} = 0$ this means a regular color histogram (except that the colors are quantized).

- $[R, G,\ \nabla^2 L_{rg}\ \sigma = 2]$ - The above combination extended with the Laplacian operator at scale $\sigma = 2$. As the operator works on both color channels independently, this combination has dimension 4.

- $[R, G, \ |\nabla L_{rg}|, \ \nabla^2 L_{rg} \ \sigma = 2]$ - The above combination extended with the gradient magnitude information on each color channel, scale $\sigma = 2$.

- $[|\nabla L| \ \sigma = 1,2,4, |\nabla L_{rg}| \ \sigma = 2]$ - Only the gradient magnitude, on the intensity channel and on each color channel individually. On the intensity channel, three scales are used, $\sigma = 1,2,4$. For each of the color channels, scale $\sigma = 2$ is used. 5 dimensions.

- $[\nabla^2 L \ \sigma = 1,2,4, \nabla^2 L_{rg} \ \sigma = 2]$ - The same combination as above, but for the Laplacian operator instead.

- $[R, G, \ |\nabla L_{rg}|, \ \nabla^2 L_{rg} \ \sigma = 2,4]$ - The combination of colors, gradient magnitude and the Laplacian, on two different scales, $\sigma = 2,4$. 10 dimensions.

All descriptor combinations were evaluated using CODID - CVAP Object Detection Image Database, (Ekvall, 2005).

3.5.1 CODID - CVAP Object Detection Image Database

CODID is an image database designed specifically for testing object detection algorithms in a natural environment. The database contains 40 test images of size 320x200 pixels, and each image contains 14 objects. The test images include problems such as object occlusion, varying illumination and textured background. Out of the 14 objects, 10 are to be detected by an object detection algorithm. The database provides 10 training images for this purpose, i.e. only one training image per object. The database also provides bounding boxes for each of the ten objects and each scene and an object is considered to be detected if the algorithm can provide pixel coordinates within the object's bounding box for that scene. In general, detection algorithms may provide several hypotheses of an object's location. In this work, only the strongest hypothesis is taken into account.

The test images are very difficult from a computer vision point of view, with cluttered scenes and objects lying rotated behind and on top of each other. Thus, many objects are partially occluded in the scene. In total, the objects are arranged in 20 different ways and each scene is captured under two lighting conditions. The first lighting condition is the same as during training, a fluorescent ceiling lamp, while the second is a closer placed light bulb illuminating from a different angle.

We have evaluated the six different descriptor configurations on CODID. Besides the detection rate, we present graphs of how various parameters, such as α and d_{max} affect the results. We also present segmentation results which show how well RFCHs perform at segmenting an object from the background.

3.5.2 Training

For training, one image of each object is provided. Naturally, providing more images would improve the recognition rate but our main interest is to evaluate the proposed method using just one training image. The training images are shown in Fig. 3.5. As it can be seen, some objects are very similar to each other, making the recognition task non-trivial. The

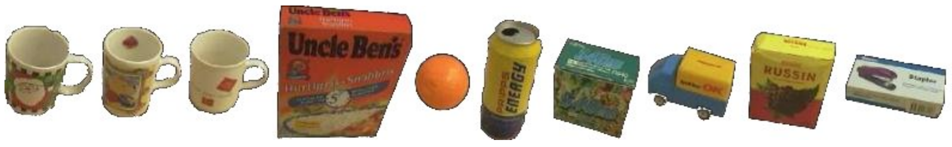


Figure 3.5: The ten images used for training.

histogram is built only from non-black pixels. In these experiments, the training images have been manually segmented. For training in robotic applications, we assume that the robot can observe the table before and after the object is placed in front of the camera and perform the segmentation based on image differencing.

3.5.3 Detection Results

The experimental evaluation has been performed using six combinations of feature descriptors. As it can be seen in Table 3.1, the combination of all feature descriptors gives the best results. The color descriptor is very sensitive to changing lighting conditions, despite the fact that the images were color normalized prior to recognition. On the other hand, the other descriptors are very robust with respect to this problem and the combination of descriptors performs very well. We also compare the method with regular color histograms which show much worse results.

Adding descriptors on several scales does not seem to improve the performance in the first case, but when lighting conditions change, some improvement can be seen. With changed illumination, colors are less reliable and the method is able to benefit from the extra information given by the Laplace and gradient magnitude descriptors on several scales. All descriptor combinations have been tested with $N = 80$ cluster-centers, except the 10-dimensional one which required 130 cluster-centers to perform optimally. Also, $d_{max} = 10$ was used in all tests, except for the color histogram method, which of course use $d_{max} = 0$.

All detection rates reported in Table 3.1 are achieved using the histogram intersection method (3.1). For comparison, we also tested the 6D descriptor combination with the χ^2 method (3.2). With this method, only 60 % of the objects were detected, compared to 95 % using histogram intersection.

3.5.3.1 Misclassification Analysis

Among the objects to be detected, there are three quite similar mugs as shown in Fig 3.5. Most detection errors originate from these three mugs being confused with each other. If one mug is partially occluded, its appearance may be less similar to its training image compared to the appearance of the other mugs. Thus, in these cases, the algorithm suggest the location of another mug as the most probable location.

Table 3.1: The detection rate of different feature descriptor combinations in two cases: i) same lighting conditions as when training, and ii) changed lighting conditions.

Lighting Condition:	Same	Changed
Descriptor Combination:		
2D: Color histogram	71.5	38.0
2D: $[R,G]$ (CCH)	77.5	38.0
4D: $[R,G,\ \nabla^2 L_{rg}\ \sigma=2]$	88.5	61.5
5D: $[\lvert\nabla L\rvert\ \sigma=1,2,4,\lvert\nabla L_{rg}\rvert\ \sigma=2]$	57	51
5D: $[\nabla^2 L\ \sigma=1,2,4,\nabla^2 L_{rg}\ \sigma=2]$	77.5	62.0
6D: $[R,G,\ \lvert\nabla L_{rg}\rvert,\ \nabla^2 L_{rg}\ \sigma=2]$	95.0	80.0
10D: $[R,G,\ \lvert\nabla L_{rg}\rvert,\ \nabla^2 L_{rg}\ \sigma=2,4]$	93.5	86.0

3.5.4 Segmentation Results

We evaluated two methods for object segmentation. We first evaluated the voting based approach presented in Section 3.3.4.1. We also tried representing the object with nine different histograms, each covering one ninth of the object. This approach was evaluated with the use of an integral image as described in Section 3.4.3.1. The search was constrained to squares with had the same geometric proportions as the object in the training image. Fig. 3.6 illustrates the different segmentation results obtained with these two methods. The more advanced method produces slightly better results, but has a number of drawbacks. First, it is not as fast. It takes a couple of seconds to come up with a result, compared the fractions of a second with the simple method. Second, while the use of multiple histograms gives a better representation, it constrains the method to only find the objects that are oriented in the same way as they were during training, because of the geometrical relation of the nine sub-windows. Also, it is less robust to occlusion.

To measure the difference of the methods we defined a segmentation success rate λ as

$$\lambda = 2 \cdot \frac{size(W_t \cup W_h)}{size(W_h) + size(W_t)} \tag{3.3}$$

where W_t denotes the true window around the object and W_h denotes the hypothesis. Thus, $\lambda = 1$ means perfect segmentation while $\lambda = 0$ means that the object is completely missed. Achieving a high λ is very difficult, and usually the best segmentations, which are visually very good, achieve a λ of about 0.85. The average λ over the 10 objects and the first 20 images in CODID was 0.56 for the voting based method and 0.6 for the integral image based method. Hence, slightly better results are obtained. Additional detection improvements are expected with boosting. However, the increased computation time is not acceptable for our applications.

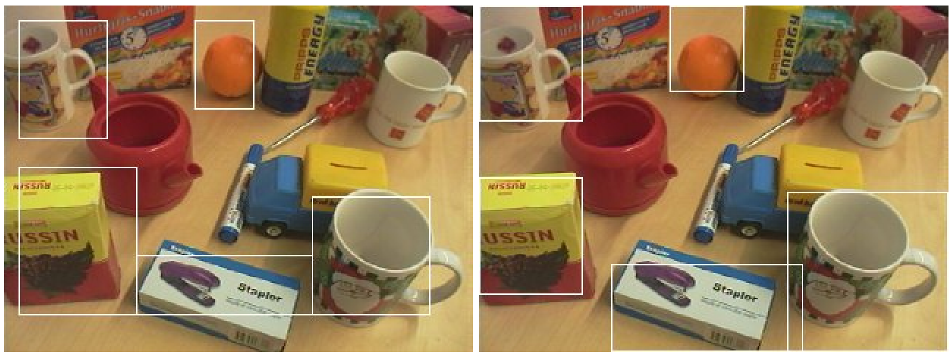


Figure 3.6: Left: The segmentation obtained with the voting based method. Right: The segmentation obtained with the integral image method. This method is able to capture the less discriminant parts of for example the mug and the stapler, due to the representation with multiple histograms.

3.5.5 Free Parameters

The algorithm requires setting a number of parameters which were experimentally determined. However, it is shown that the detection result are not significantly affected by the values of parameters. The parameters are:

- **Number of cluster-centers,** N
  We found that using too few cluster-centers reduces the detection rate. From Fig. 3.7 it can be seen that feature descriptor combinations with high dimensionality require more cluster centers to reach their optimal performance. As seen, 80 clusters is sufficient for most descriptor combinations.

- **Maximum pixel distance,** d_{max}
  The effect of cooccurrence information is evaluated by varying d_{max}. Using $d_{max} = 0$ means no cooccurrence information. As seen in Fig. 3.8, the performance is increased radically by just adding the cooccurrence information of pixel neighbors, $d_{max} = 1$. For $d_{max} > 10$ the detection rate starts to decrease. This can be explained by the fact that the distance between the pixels is not stored in the RFCH. Using a too large maximum pixel distance will add more noise than information, as the likelihood of observing the same cooccurrence in another image decreases with pixel distance. As seen in Fig. 3.8, the effect of the cooccurrence information is even more significant when lighting conditions change.

- **Size of cluster-centers,** α
  We have investigated the effect of limiting the size of the cluster centers. Pixels that lie outside all of the cluster centers are classified as background and not taken into account. As seen in Fig. 3.8, the algorithm performs optimally when $\alpha = 1.5$, i.e.

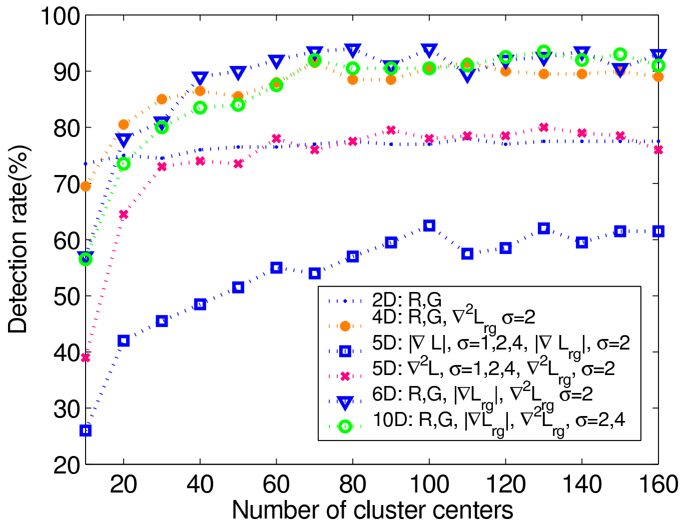
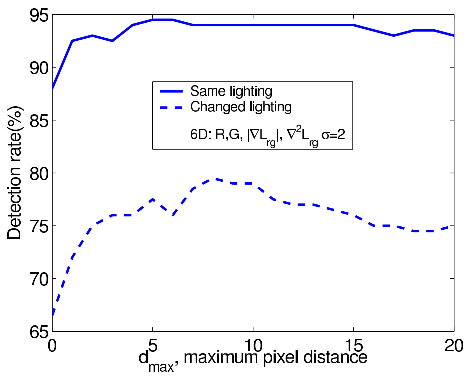


Figure 3.7: The importance of the number of cluster-centers for different image descriptor combinations.



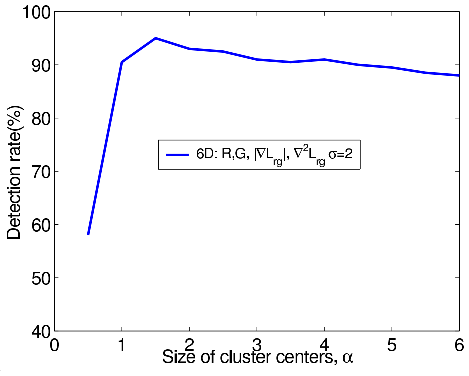


Figure 3.8: Left: The detection rate mapped to the maximum pixel distance used for cooccurrence, d_{max}, in two cases: Same lighting as when training, and different lighting from training. Right: The detection rate mapped to the size of the cluster centers, α.

the size is 1.5 times the average distance to the cluster center used during training. Smaller α removes too many of the pixels and, as α grows, the effect described in Section 3.4.2 starts to decrease.

- **Search window size**
  We have found that some object recognition algorithms require a search window size of a specific size to function properly for object detection. This is a serious drawback, as the proper search window size is commonly not known in advance. Searching the image several times with different sized search windows is a solution, although it is quite time consuming. As Fig. 3.9 shows, the choice of search window size is not crucial for the performance of our algorithm. The algorithm performs equally well for window sizes of 20 to 60 pixels. In our experiments, we have used a search window of size 40.

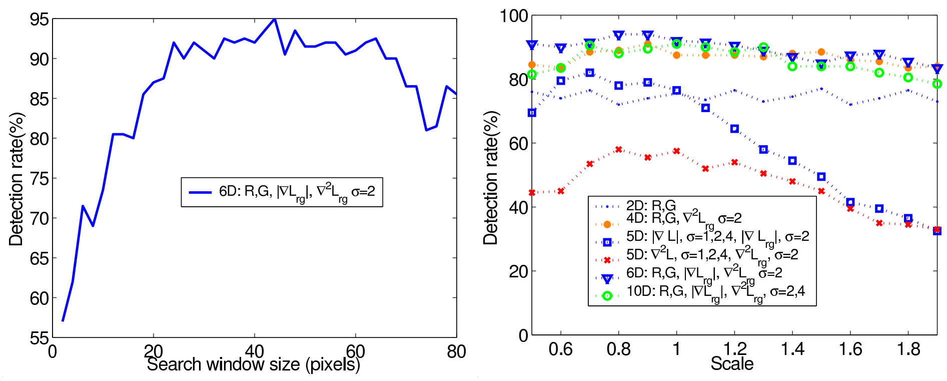


Figure 3.9: Left: The detection rate mapped to the size of the search window. Right: The effect on detection rate when the training examples are scaled, for six different image descriptor combinations.

3.5.6 Scale Robustness

We have also investigated how the different descriptor combinations performs when the scale of the training object is changed. The training images were rescaled to between half size and double size, and the effect on detection performance was investigated. As seen in Fig. 3.9, the color descriptors are very robust to scaling, while the other descriptors types decrease in performance as the scale increase. However, when the descriptor types are combined, the performance is partially robust to scale changes. To improve scale robustness, the image can be scanned at different scales.

3.5.7 Conclusion

In the last few sections, a new method for object detection has been presented. Receptive Field Cooccurrence Histograms are used to represent the object appearance, and such a histogram captures how common *pairs* of certain filter responses and colors are within an image. The experimental evaluation shows that the method is able to successfully detect objects in cluttered scenes, and that the method is robust to scale changes and illumination variations. The performance of the method depends on a number of parameters but we have shown that the choice of these are not crucial. On the contrary, the algorithm performs very well with a wide variety of parameter values.

For an algorithm to be used for object detection, it has to be able to recognize the object although it is placed on a textured cloth and only partially visible. The CODID image database was specifically designed for testing these types of natural challenges, and we have reported good detection results on this database. The algorithm is fast and fairly easy to implement. Training of new objects is a simple procedure and only a few images are sufficient for a good representation of the object.

We have presented two segmentation methods. One is based on a single histogram, while the other features multiple histograms for object representation. Initial results show that multiple histograms give somewhat better segmentation, however at the cost of increased complexity and decreased speed.

In the experiments presented in this section, the search object was assumed to be visible in the image. To decide if the most probable location contains the object, one has to compare the histogram vote with a threshold value. However, the threshold value depends both on the object as well as the scene, which makes it difficult to use this method for object recognition in complex scenes. Still, the method is good for providing hypotheses of where the searched object could be.

In the next section, we will estimate the pose of the object using the appearance-based CCH method to initialize a frame of the object, which is then fitted to the image.

3.6 Pose Estimation

To grasp an object, knowing the location of the object is not enough. The robot must know the pose of the object in order to adjust its gripper accordingly. In this section, we present a method of obtaining the pose using the appearance-based method in combination with a model based method. Because this work was done prior to the discovery of RFCHs, the results in this section were obtained using CCHs. Even better results are expected using RFCHs.

Once the object has been segmented from the image, its rotation around the vertical axis may be estimated. The appearance-based rotation estimation is performed using many training images of the object taken from different angles. Each training image is labeled with the correct rotation of the object in that image. When estimating the pose for a new image, the database is searched for the image with the most similar appearance. The similarity in appearance of two poses i and j is calculated according to equation 3.1. Fig. 3.10 shows the dependency between the match value $\mu(i,j)$ and angular separation in object pose $|\alpha(i)-\alpha(j)|$. To improve the robustness, the hypotheses are first weighted by a Gaussian. If the i-th training image with a known angle α_i matches the segmented image of unknown pose to a degree μ_i, the likelihood $P(\beta)$ of the object angle rotation β is calculated as:

$$P(\beta) = \frac{\sum_{i=0}^{N} \mu_i g(\beta, \alpha_i)}{\sum_{i=0}^{N} g(\beta, \alpha_i)} \tag{3.4}$$

The Gaussian kernel function

$$g(\beta, \alpha) = \frac{1}{\sigma(2\pi)^{1/2}} e^{\frac{-(\beta-\alpha)^2}{2\sigma^2}} \tag{3.5}$$

captures the degree to which the vote μ_i of a training image contributes to $P(\beta)$ based on the distance $\beta - \alpha_i$. The maximum of $P(\beta)$ emerges in vicinity of training images with high match values μ. For the example shown in Fig. 3.10, the match values μ_i of training images are clearly correlated with the object's angle of rotation α_i. The distribution has a global maximum at $-39 deg$, and a second local maximum at ± 180 *deg*. The two minima occur at ± 100 *deg*. The algorithm estimates the rotation angle of -39 *deg* at the global maximum which is a fairly accurate estimate of the true rotation angle of -37 *deg*.

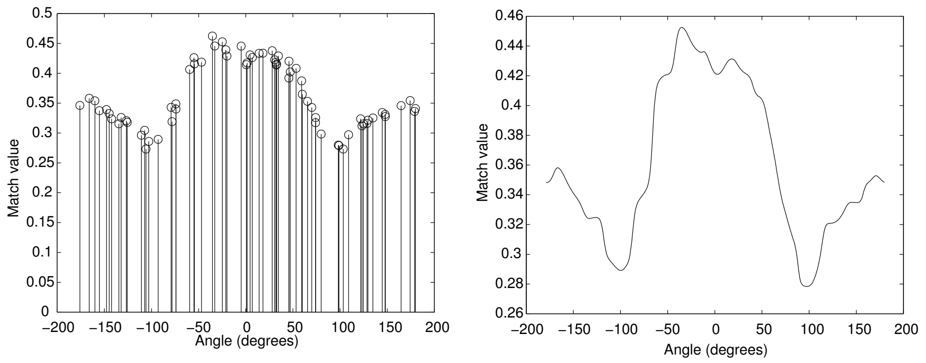


Figure 3.10: Left) The match values μ of training images before, and Right) after convolution with a Gaussian kernel.

3.6.1 Model Based Pose Estimation

If a model of the object is available, a more accurate pose estimation can be achieved, and a full 6-D object pose is obtained, Vincze *et al.* (1999). We have integrated the work in (Kragic and Christensen, 2002) with the appearance-based rotation estimation. Our approach combines the accuracy of geometry based methods with the robustness of appearance-based methods in a synergistic fashion where the key idea of the integrated algorithm is to obtain the initial pose estimate using the appearance-based method. This estimate allows it to project features of the object model onto the image. These projected features provide sufficient prior information to initialize the local search and matching of corresponding features in the image. The integrated approach reduces the global correspondence problem to a local tracking problem.

A typical model based tracking system usually involves the following steps: detection, matching, pose estimation, update and prediction, see Fig. 3.11. The input to the algorithm

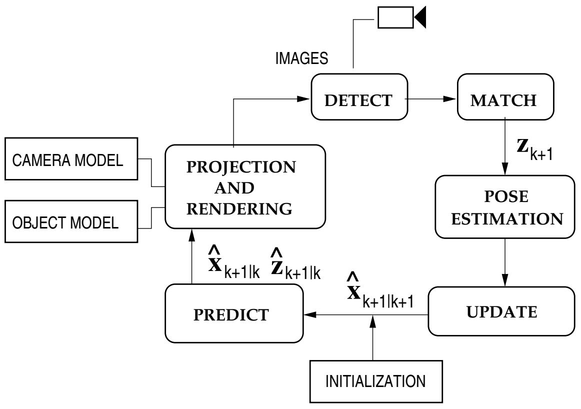


Figure 3.11: Block diagram of our model based tracking system.

is a wire–frame model of the object. The main loop starts with a *prediction* step where the state of the object is predicted by means of the current pose (velocity, acceleration) estimate and a motion model. The visible parts of the object are then projected into the image (*projection and rendering* step). After the features are *detected*, they are *matched* to the projected ones and used to estimate the new pose of the object. Finally, the calculated pose is input to the *update* step.

3.6.1.1 Prediction and Update

The system state vector consists of three parameters describing translation of the target, another three for orientation and an additional six for the velocities:

$$\mathbf{x} = \left[X, Y, Z, \phi, \psi, \gamma, \dot{X}, \dot{Y}, \dot{Z}, \dot{\phi}, \dot{\psi}, \dot{\gamma}\right] \tag{3.6}$$

where ϕ, ψ and γ represent roll, pitch and yaw angles (Craig, 1989). The following piece-wise constant white acceleration model is considered (Bar-Shalom and Li, 1993):

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{v}_k, \qquad \mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{w}_k \tag{3.7}$$

where $\mathbf{v}_k$ is a zero–mean white acceleration sequence, $\mathbf{w}_k$ is the measurement noise and

$$\mathbf{F} = \begin{bmatrix} \mathbf{I}_{6\times6} & \Delta T\mathbf{I}_{6\times6} \\ \mathbf{0} & \mathbf{I}_{6\times6} \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} \frac{\Delta T^2}{2}\mathbf{I}_{6\times6} \\ \Delta T\mathbf{I}_{6\times6} \end{bmatrix}, \quad \mathbf{H} = \left[\mathbf{I}_{6\times6} \mid \mathbf{0}\right] \tag{3.8}$$

For the prediction and update, the $\alpha - \beta$ filter is used:

$$\begin{aligned} \hat{\mathbf{x}}_{k+1|k} &= \mathbf{F}_k\hat{\mathbf{x}}_k, \quad \hat{\mathbf{z}}_{k+1|k} = \mathbf{H}\hat{\mathbf{x}}_{k+1|k} \\ \hat{\mathbf{x}}_{k+1|k+1} &= \hat{\mathbf{x}}_{k+1|k} + \mathbf{W}[\mathbf{z}_{k+1} - \hat{\mathbf{z}}_{k+1|k}] \end{aligned} \tag{3.9}$$

Here, the pose of the target is used as measurement rather than image features, as commonly used in the literature, (Dickmanns and Graefe, 1988), (Gengenbach *et al.*, 1996). An approach similar to the one presented here was considered in (Wunsch and Hirzinger, 1997). This approach simplifies the structure of the filter which facilitates a computationally more efficient implementation. In particular, the dimension of the matrix **H** does not depend on the number of matched features in each frame but it remains constant during the tracking sequence.

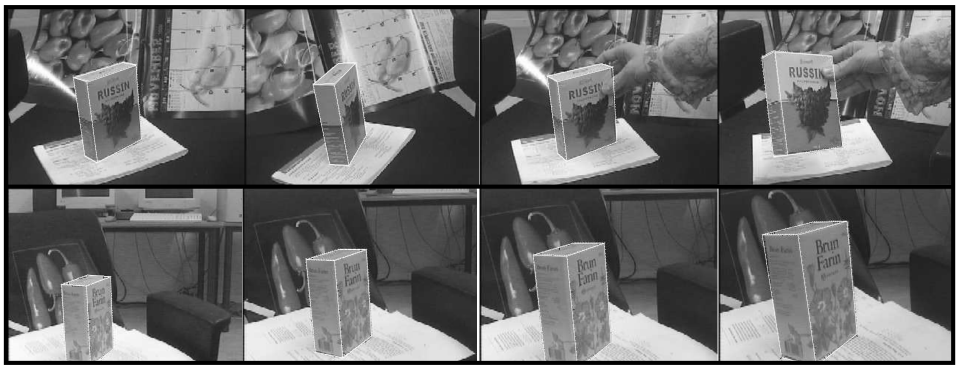


Figure 3.12: first row) An example of tracking a package of raisins: a fairly textured object against a textured background. The estimated pose of the object is overlaid in white. During this experiment a 6mm lens was used and the object was at a distance of approximately 50cm from the camera, and second row) A moving camera and a static object show the ability of the system to cope with significant depth changes and perspective effects.

3.6.2 Experimental Evaluation

The proposed system was experimentally evaluated for: i) segmentation and rotation estimation, and ii) full 6-DoF pose estimation and tracking.

3.6.3 Object Recognition and Rotation Estimation

For training, the correct pose of the object is estimated by manually matching corresponding corner points between the image and a wire–frame model of the object. We have implemented a combination of methods proposed in (DeMenthon and Davis, 1995) and (Araujo *et al.*, 1996). For each training image the complete CCH is computed off–line and stored together with the known rotation of the object. To minimize the noise in the training images, the background is manually removed from the images prior to training. During the experimental evaluation we observed that after about 50 training images no significant improvement in the accuracy is gained. At run time, the CCHs of the candidate windows are matched to the stored information to retrieve the rotation α of the object around the vertical axis. The background is not removed from the test images and CCHs are based on all pixel pairs separated by less than 10 pixels, which roughly amounts to 600k pixel pairs per segmented test image. Fig. 3.13 illustrates how the CCH of a training image changes as the object is rotated by 0, 45 and 90 degrees.

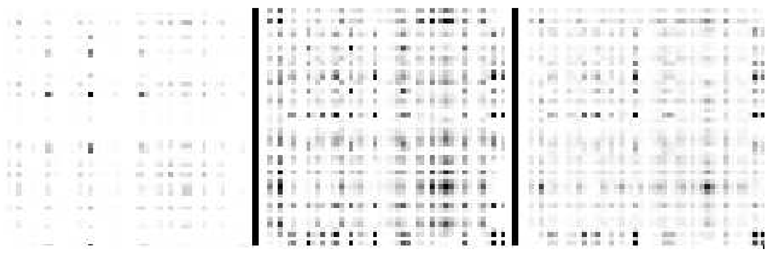


Figure 3.13: The CCH of a training image changes significantly with the angle of the object. The size of the CCH is 50x50 bins. Dark areas indicate high counts in the corresponding CCH bin. Left: Object rotated with 0 degrees. Center: Object rotated with 45 degrees. Right: Object rotated with 90 degrees.

Our object recognition and rotation estimation algorithms are tested in combination where the strongest hypothesis from the former serves as input to the latter. 70 images of size 100x100 pixels, with a removed background, were used for training. 30 images of size 320x200, showing the rice package in natural scenes were used for testing. In these images, the rice package typically occupied about 5 % of the image.

The CCHs corresponding to $\pm X\ deg$ are commonly very similar since they are basically “mirrored”. This results in an ambiguous match value distribution. To deal with this problem, pixel pairs corresponding to positive and negative angles are stored in separate bins, see Fig. 3.14. In addition, the most dominating part of the CCH are pixel pairs with a distance 0. Therefore, these pairs are excluded from histograms during rotation estimation. As a confidence value, C for rotation estimation we use the ratio between the magnitude of the two largest matching values

$$C = \frac{\mu_{max} - \mu_{avg}}{\mu_{2ndmax} - \mu_{avg}} \tag{3.10}$$

We experimentally determined the optimal values for the number of color clusters N and the width of the Gaussian kernel σ by means of cross-validation. For the rice package, the optimal values are $N = 50$ and $\sigma = 5$. The results for the winner-take-all approach with $\sigma = 0$ are inferior compared to applying convolution.

As an example, the two narrow surfaces of the rice object are easily confused as they appear almost identical, except for a small patch of letters on one of the sides. The right part of Fig. 3.15 shows segmented images of the rice package taken from opposite directions. As a result of this confusion, the match value graph is bimodal, as seen in Fig. 3.15. In our experiments, the algorithm successfully estimates the angle in all cases in spite of this problem. However, a small fraction of noise is enough to make the algorithm select

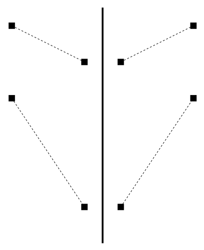


Figure 3.14: By separating pixel pairs with different orientation, and storing them in separate bins, mirrored images will not have the same CCH. Pixel pairs on the left side have the same orientation, opposite to the orientation of the pixel pairs on the right side.

the other alternative. For the purpose of grasping such a symmetric object, however, it is irrelevant whether it is rotated -90 *deg* or $+90$ *deg*.

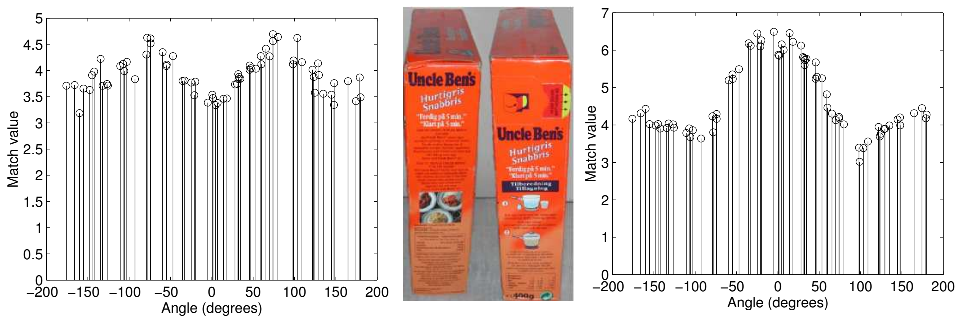


Figure 3.15: Center: The appearance of rice package rotated -90 *deg* is very similar to the appearance when it is rotated $+90$ *deg*. This results in a bimodal match value graph (left). An example of a match value graph in an unambiguous case is shown to the right.

The average angular error is 6 *deg* which is quite remarkable considering that the angles computed by means of manual feature matching already carry an uncertainty of about 5 *deg*.

In our application, the main purpose of the appearance-based method is to robustly provide a pose estimate that is accurate enough for initialization of corresponding features in the tracking based scheme. The feature-based tracking method tolerates angular errors in the initial pose of up to $25-30$ *deg*. As shown in the angular error histogram in Fig. 3.16, all of the 30 test cases meet this requirement.

We also tested the robustness of the pose estimation with respect to changes in scale, camera angle and noise level. The camera tilt angle is varied between 0 and about 30 *deg* between test- and training images, which this time contained a raisins package instead of the rice package. The average angular error increases to 17 *deg*. Thus, it can be concluded

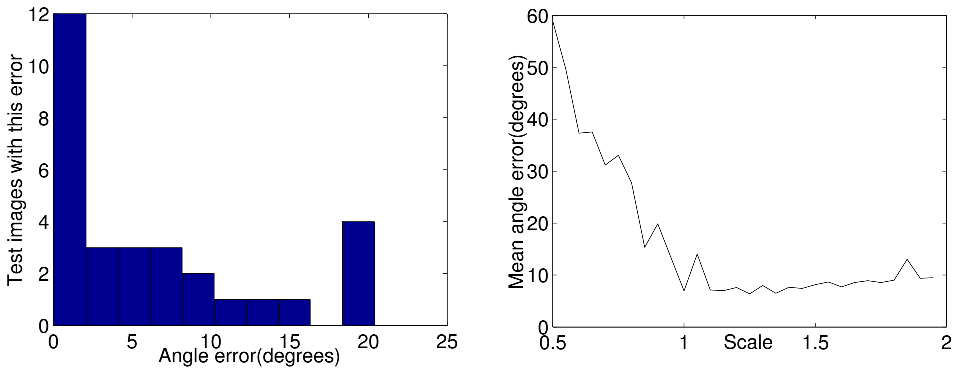


Figure 3.16: Left) Distribution of angular error, and Right) Mean angular error as a function of variations in scale.

that the algorithm is robust with respect to reasonable changes in the camera perspective effects.

We further evaluated the robustness with respect to changes in scale for a range $[0.5 - 2.0]$. As shown in Fig. 3.16, the angular error remains below 20 *deg* over a range $[0.8 - 2]$. In our application, the table area that can be reached by the manipulator is fairly limited, such that the distance to the object to be grasped does not vary significantly. For applications in which the distance between object and camera is more uncertain, it may become necessary to perform additional training at wider range of scales and orientations.

Random pixels are added to the test images in order to test the robustness towards noise. In Fig. 3.17, the impact of image noise on the mean angular error is shown. Noise levels above 40% cause a considerable decrease in performance. This is easily explained by the fact that the information stored in a CCH is already corrupted if one of the two pixels is effected by noise or occlusion. At a noise level of 40% per pixel, effectively only 36% of the pixel pairs remain intact. This observation underlines the need for proper object segmentation prior to the pose estimation step. We note here that an angular error of $25 - 30$ *deg* is still sufficiently accurate for proper initialization of the model based pose estimator. In terms of timing, the execution time for the pose estimation step on a Sunblade 100 (500 MHz) was 0.3 seconds.

3.6.4 Full 6-DoF Pose Estimation

In the integrate scheme, object recognition and rotation estimation serve as the initial values for the model based pose estimation and tracking algorithm. The distance of the object from the camera, Z is estimated according to the ratio between the height of the segmented window and the height of the object (which is known from the model) together with the camera parameters. Similarly, X and Y are estimated from the window position in the

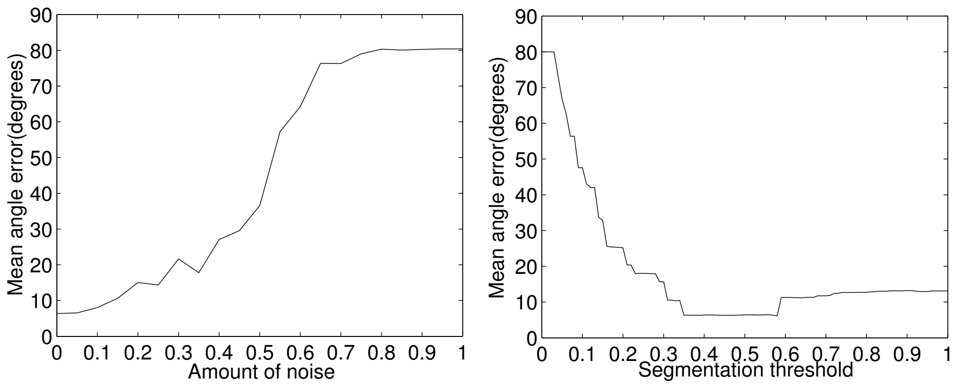


Figure 3.17: Left) Angular error as a function of image noise , and Right) segmentation threshold θ.

image. The rotation of the object around the vertical axis is obtained from the rotation recognition step, while the remaining two angles are initialized to zero.

Fig. 3.18 shows a few examples of processing steps in the integrated scheme. With the incomplete pose calculated in the recognition (first figure from the left) and orientation estimation step, the initial full pose is estimated (second figure from the left). After that, a local fitting method matches lines in the image with edges of the projected object model. The image obtained after convergence of the tracking scheme is shown on the right. Table 3.2 contains the pose values before and after the fitting stage. It is important to note, that even under the incorrect initialization of the two other rotation angles as zero, our approach is able to cope with significant deviations from this assumption. This is strongly visible in the last row in Fig. 3.18 where, according to the results reported in Table 3.2, the angle around camera's Z-axis is larger than 20 deg.

3.7 Discussion

Object recognition and pose estimation are basic prerequisites for robust robotic manipulation and object grasping. In this chapter, a novel approach for object recognition and pose estimation based on receptive field cooccurrence histograms and geometric model based techniques have been presented. The particular problems addressed were: i) robust recognition of objects in natural scenes, ii) estimation of partial pose using an appearance-based approach, and iii) complete 6-DoF model based pose estimation and tracking.

It has been demonstrated that CCHs are computationally efficient for representing the appearance of an object in the context of object recognition and partial pose estimation. Because of their invariance to scaling and translations, the algorithm performs robustly in natural settings. We have shown that RFCHs perform better for object detection, especially under dynamic lighting conditions. It is expected that RFCHs would perform better for rotation estimation as well.

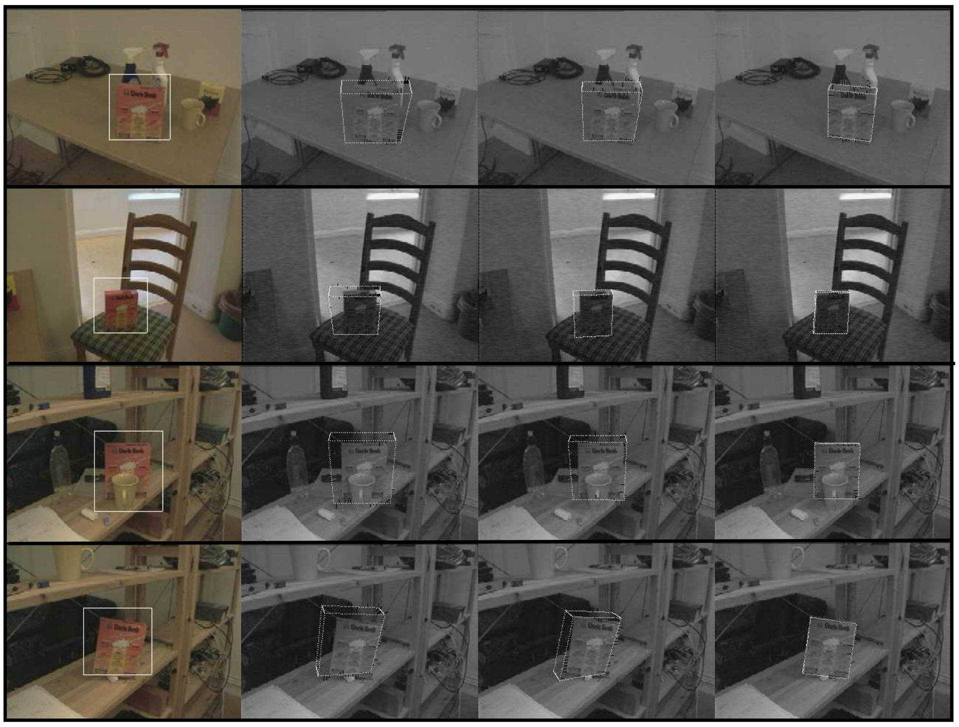


Figure 3.18: From object recognition to pose estimation Test1 - Test4, (from left): i) the output of the recognition, ii) initial pose estimation, iii) after few fitting iterations, iv) the estimated pose of the object.

Test NO	X_{bef}	X_{aft}	Y_{bef}	Y_{aft}	Z_{bef}	Z_{aft}	ϕ_{bef}	ϕ_{aft}	ψ_{bef}	ψ_{aft}	γ_{bef}	γ_{aft}
	mm	mm	mm	mm	mm	mm	deg	deg	deg	deg	deg	deg
Test 1	-41	-19	125	147	590	802	45	22	0	9	0	5
Test 2	-93	-89	206	265	748	976	40	16	0	-4	0	1
Test 3	-72	-66	138	159	587	774	30	16	0	-2	0	0
Test 4	-112	-110	120	143	584	756	30	13	0	2.5	0	-22

Table 3.2: Values show object's pose before and after the fitting stage. Test1-4 represent experiments shown in Fig. 3.18.

On a basis of 70 training images, our scheme consistently estimates the object poses of all 30 test images with a maximum angular error of less than 20 *deg* and an average angular error of 6 *deg*. The method is sufficiently robust towards variations in camera angle and scale and is partially able to cope with image noise and occlusion.

It has to be noted here that the experiments were conducted under the assumption that the object is visible in the image. The RFCH algorithm merely returns the most probable location for an object. The problem of deciding if the object is actually there or not is harder, and extensive experiments have shown that using local features for this is better than appearance-based methods. In Chapter 6 we present a method of combining an appearance-based method with a feature-based method for both finding and recognizing objects in complex scenes.

Chapter 4

Grasp Mapping, Recognition and Execution

For indirect task learning, recognizing objects is important but not sufficient. If the task involves object manipulation, the robot also has to learn how the object should be grasped. The proper grasp type depends both on the target object, and the current task. As an example, let us assume that the task is to pick up a cup to fill it with coffee. Generating suitable grasps based solely on the object shape provides three possible grasps: a circular sphere grasp, a power wrap grasp, and a two-finger-thumb precision grasp as seen in Fig. 4.1. However, the task knowledge introduces additional constraints because, in order to fill the cup, the opening should not be covered. In this case, only two grasps remain feasible.

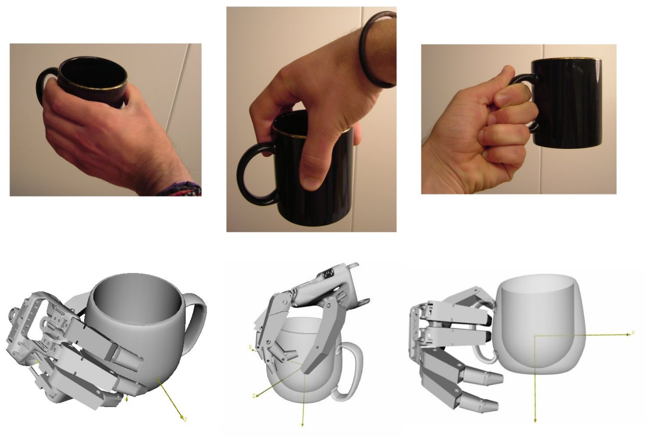


Figure 4.1: Different possible grasps for picking up a cup.

Another issue is the *mapping* from human grasp to robot grasp. In the above example, the robot hand is similar to the human, and should ideally have the same joint values. However, due to kinematic constraints this is not always possible. This will become more evident as we examine other robot hand types in Section 4.1. Here, we investigate semi-autonomous grasping and human-robot grasp mapping. One particular challenge is that our equipment only allows us to measure fingertip positions, not finger joint values. Thus, those positions have to be translated into robot joint values.

In Section 4.2 we move to autonomous grasping. The grasp mapping presented in Section 4.1 is hard to realize in an autonomous grasp setting because it requires the robot to identify *where* on the object the grasp is taken, i.e., where the contact points are. This is difficult since the hand will often occlude the object, or vice versa. Instead, our strategy here is to, in Section 4.3, recognize the human grasp *type*, and then map this type into a corresponding robot type by a fixed mapping scheme. The grasp recognition is based on the hand pose during the entire grasp sequence, and trained from human examples. Once the robot knows the pose of an object, and what grasp type to use, the problem is how to grip the object using the grasp type. This is investigated in Section 4.4. Here, the robot evaluates several approach vectors to the object before it actually performs the grasp.

4.0.1 GraspIt!

In this work, we have used a grasping simulator, called GraspIt! (Miller and Allen, 2000), to analyze and visualize the poses of a variety of different robot hands which are not available to us in practice. Another advantage of using the simulator is the possibility to perform a large number of experiments with slightly changed conditions and system parameters. This is not feasible in the real world as the world state has to be reset between each experiment.

Graspit! can import a wide variety of different hands and robots, model environments with objects and all of these can be manipulated within a virtual 3D workspace. A custom collision detection and contact determination system prevents bodies from passing through each other and can find and mark contact locations. A dynamics engine can compute contact and friction forces over time.

4.1 Mapping Human Grasps to Robot Grasps

The human hand can perform a wide variety of grasps. Because of the difficulties in constructing a robotic gripper equivalent to the human hand, most robotic grippers have much less flexibility. Consequently, the human and robot hand cannot, in general, perform the same types of grasps. As the kinematics and configuration spaces of a human hand and an artificial robot hand are generally different, the fingertip positions of the robot hand cannot correspond exactly to the fingertip positions of the human hand (especially when fingertip grasps are considered). Consequently, a mapping between human hand and robot hand is necessary in order to control a robot hand with a human hand. This mapping changes for different robotic manipulators. For example, the mapping for a three finger robot hand like the Barrett hand (Townsend, 2000), which has only four degrees of freedom, is different to the mapping for a five finger robot hand like the Robonaut hand (Lovchik and Diftler, 1999), which has 14 DoFs.

One way of performing grasp mapping is to calculate the inverse kinematics of the robot hand and then try to minimize the sum over fingertip distances. However, this requires an analytical solution for each robot hand type. In this work we have instead used an Artificial Neural Network (ANN) for grasp mapping. From a few training examples the ANN can learn the mapping space from human to robot hand. This method also fits well in the Programming by Demonstration paradigm.

In general, there are two ways of representing mapping between these two spaces, using:

- **Joint space**
  Mapping using the joint space representation facilitates the *similarity* between hands' poses. This is suitable for enveloping or power grasps, which are explained later on in this chapter.

- **Cartesian space**
  Mapping using the Cartesian space is more suitable for representing the fingertip positions. This is a natural approach when, for example, precision grasps are considered.

Related to the work presented here, a third group may be added to this list, namely the combination of the above. Here, the positions of the human hand's fingertips are mapped to some joint values of the robot hand. If the robot and the human have similar hand configurations, the result is likely to be the same as for purely Cartesian mapping. However, if the robot hand's kinematics is very different from the human one, as for example in the Barrett hand case, Cartesian mapping is not suitable.

4.1.1 Measuring the Hand Posture

To be able to recognize a grasp, information about the hand posture, such as fingertip locations or joint angles, is needed. The three approaches used in the literature are:

- **Vision**
  Visual feedback is commonly used as an external sensor to provide fingertip locations. In this case, the sensor does not have to be mounted on the person performing the grasp. A difficult problem is to obtain robust measurements in case of occlusions, shadows, specular reflections, etc. A common approach is to use predefined hand models and fit the estimated vertices to the predefined grasp postures in 3D, (Ogawara *et al.*, 2003; Bretzner, 1999).

- **Magnetic trackers**
  Another alternative is the use of magnetic trackers, such as the Nest of Birds (Ascension Tech., 2006). It is a magnetic tracker that consists of an electronics unit, a transmitter and four pose measuring sensors. From signals measured by the sensors,

Nest of Birds calculates the position and orientation of each sensor providing six degrees of freedom.

- **Optical Gloves**
  A popular measuring device is also the Cyber Glove (Virtual Technologies Inc., 1995), with 22 degrees of freedom. By using optical wires in the fingers, it is able to measure joint angles. Clearly, joint angles are preferred for tasks such as grasp recognition. Unfortunately, such detailed information can be obtained only with this type of sensor and it is difficult to scale the evaluation methods with respect to different sensory inputs such as vision. In that manner, a grasp recognition system based on fingertip locations is much more flexible.

4.1.1.1 Measurement System

To obtain measurements, we have used the Nest of Birds. It is a magnetic tracker that consists of an electronics unit, a transmitter and four pose measuring sensors. From signals measured by the sensors, Nest of Birds calculates the position and orientation of each sensor providing six degrees of freedom. The sensors are mounted on a glove as illustrated in Fig. 4.2. The center sensor, mounted on the back side of the glove, serves as a reference sensor. It measures the position and orientation of the hand. The remaining three sensors are mounted on the thumb, index finger and little finger, and provide fingertip position measurements.

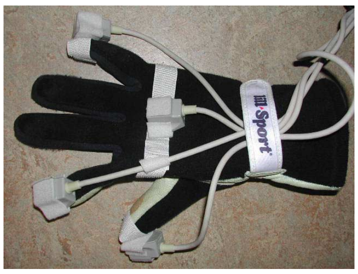


Figure 4.2: The glove used for human input.

4.1.1.2 Calculating the Fingertip Positions

Each of the sensors provides a 3D-position **p**, calculated according to (4.1) resulting in a total of nine values. The position of the sensor is represented by $\mathbf{x} = (x, y, z)$, and the reference sensor is represented by $\mathbf{x_r} = (x_r, y_r, z_r)$. The rotation matrix M is calculated from the Euler angles ϕ, ψ and γ according to (4.2). As seen in (4.1), the features derived from the sensors are both translationally and rotationally invariant. This means that the same values are extracted regardless of the rotation angle of the hand.

$$\mathbf{p} = M(\mathbf{x} - \mathbf{x_r}) \tag{4.1}$$

$$M = \begin{pmatrix} c_\phi c_\psi & s_\phi c_\psi & -s_\psi \\ -s_\phi c_\gamma + c_\phi s_\psi s_\gamma & c_\phi c_\gamma + s_\phi s_\psi s_\gamma & c_\psi s_\gamma \\ s_\phi s_\gamma + c_\phi s_\psi c_\gamma & -c_\phi s_\gamma + s_\phi s_\psi c_\gamma & c_\psi c_\gamma \end{pmatrix}$$

$$(c_\phi = cos(\phi),\ s_\psi = sin(\psi)\ ...) \tag{4.2}$$

4.1.2 Using an Artificial Neural Network for Grasp Mapping

A two-layered ANN illustrated in Fig. 4.3 was used to learn the mapping from the human hand to the robot hand. The input values are the position coordinates of each of the fingers. There are three fingers with 3D-coordinates which gives us nine input values. Note here that the coordinates are provided relative to the reference sensor on the back of the hand. The output layer has as many neurons as the number of DoFs of the robot hand, and each neuron correspond to a joint angle.

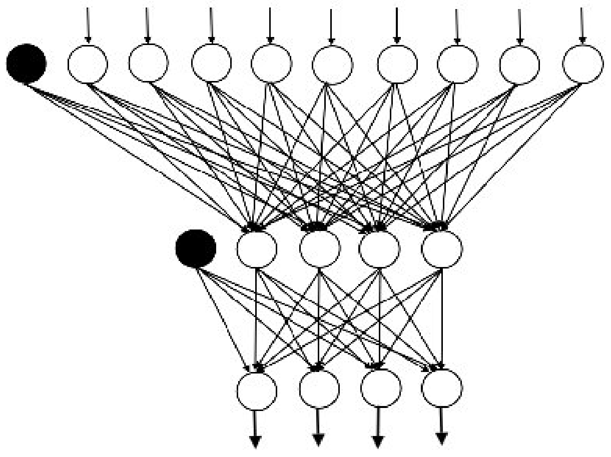


Figure 4.3: The neural network used for mapping the human hand to the Barrett hand. The input consists of fingertip positions scaled to [-1, 1] and the output is the Barrett joint values, also scaled to [-1, 1].

First, the user wearing the glove demonstrates a pose to the system which is captured when a key is pressed. Then, Graspit! is used so that the user drags the links and joints of the hand to achieve the corresponding pose. These two poses then constitute a training sample for the ANN. When enough training samples have been gathered (usually only a few is sufficient), the ANN is trained. This step takes only a few seconds on a regular Pentium 450MHz PC. For training, we used the back-propagation algorithm. A sigmoid function is used at each neuron to add non-linearity to the system, and the input values are scaled to [-1, 1].

4.1.3 Evaluation

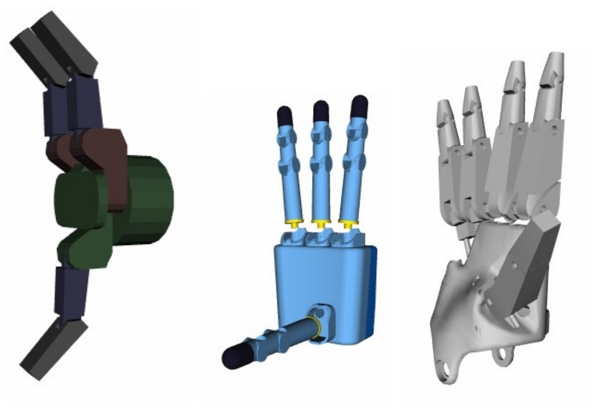


Figure 4.4: Three different robotic manipulators studied in this thesis: (a) Barrett hand (b) DLR hand (c) Robonaut hand.

We have performed grasp mapping evaluation on three different manipulators, shown in Fig. 4.4. We first evaluated how good mapping we could obtain, using just seven training postures: Open hand, each individual finger closed and each half-closed. Many additional postures were captured for testing, these included postures with many fingers closed simultaneously. We used the RMS error for the obtained finger joints compared to the expected finger joints as a mapping quality measure. The RMS error is more descriptive than the mean error because if the mapping error is large for a single finger but zero for the others, the quality is worse than if there is a low error on all fingers. We used this measure to choose the number of hidden neurons, and the best values were found to be 4, 11, 12 for the Barrett, DLR and Robonaut hand, respectively. The results show that seven postures is enough for most cases, but as we will show later, when it comes to object grasping more training examples improves the mapping accuracy.

4.1.3.1 Barrett Hand

The Barrett hand (Townsend, 2000) shown in Fig. 4.4(a) has three fingers and four degrees of freedom (DoFs), one DoF for each finger, the fourth degree is for spread angle. One finger is static and only has the possibility to close towards the center, while the movement of the two remaining fingers is defined by spread angle and the closing angle for each finger.

The kinematic constraints of this hand means that there are many differences between the Barrett and the human hand. This makes controlling the fourth DoF, the spread angle, very difficult. Therefore, this joint was fixed and only three degrees of freedom were controlled (the closing angle of each finger).

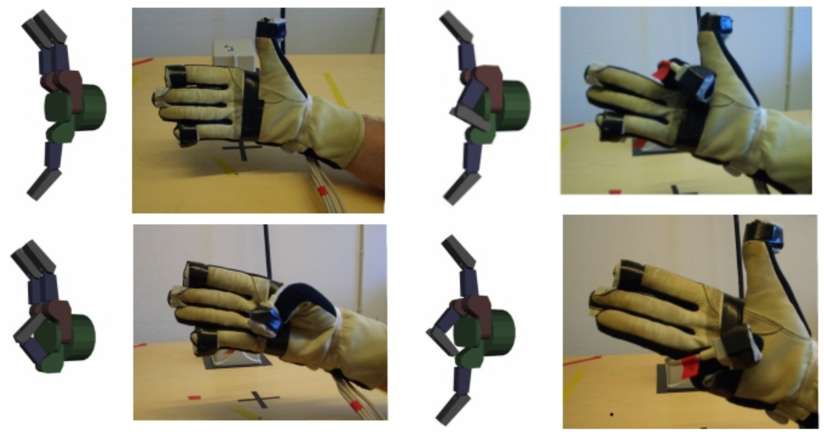


Figure 4.5: Example training postures used for the Barrett hand: Open hand, thumb closed, index finger closed and little finger closed.

Despite the big differences between human and Barrett hand, an intuitive joint to joint mapping is possible, if the spread angle is fixed. Note that we have the same number of sensors and DoFs to control. Some exemplary training images are shown in Fig. 4.5. For this hand we obtained an average RMS DoF error of 1.47 degrees.

4.1.3.2 DLR Hand

The DLR hand (Borst *et al.*, 2003) shown in Fig. 4.4(b) has four fingers and twelve degrees of freedom: two DoFs for each finger closure and another degree for the spread angle of each finger. This hand has the possibility of closing each finger in two different directions limited by the spread angle. The kinematic constraints make DLR hand different from the Barrett hand and more similar to the human hand. This makes it easier and more intuitive to control. However, the spread DoF of each finger is difficult to control using position to joint mapping, and it may cause problems with finger collisions. Therefore, these joints were restricted, and the two remaining DoFs for each finger were directly controlled (the closing angle of each finger). For the DLR hand different training configurations have been tested. Most similar to human motion is controlling the DLR thumb with the human thumb and separate the three remaining fingers into two groups. We decided that the index finger controls the first group and the little finger controls the second group. Fig. 4.6 shows some of the different training postures for DLR hand. If more sensors are available, the individual fingers could easily be controlled separately. For this hand we obtained an average RMS DoF error of 1.15 degrees.

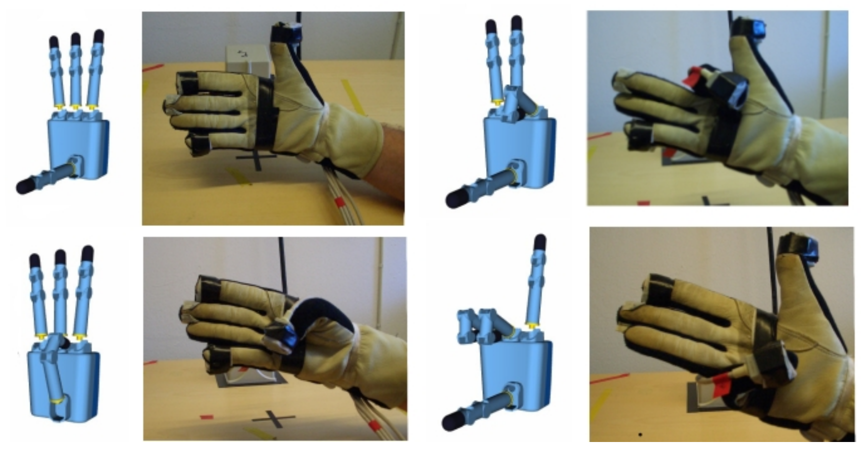


Figure 4.6: Some of the training postures for the DLR hand: open hand, thumb closed, index finger closed and little finger closed that controls two fingers in the DLR hand.

A problem in mapping with the DLR is that the fingers of the manipulator hand may collide even when the fingers of human hand do not collide, due to spread angle motion, see Fig. 4.7. This is a problem since if one of the fingers when the hand is closed locks another finger, then the fingers have to be "unfolded" in the correct order, i.e., the finger on top should be opened first. But if the user unfolds his/her grasp in another order, it results in a situation in which the manipulator hand is closed while the human hand is open, then the movements of human hand can not be performed until the robot hand is unlocked. To avoid this problem we have limited the spread angle range and therefore there are less possibilities of collision between fingers, see Fig 4.8.

4.1.3.3 Robonaut Hand

The NASA Robonaut (Lovchik and Diftler, 1999) hand shown in Fig. 4.4(c), has a total of fourteen degrees of freedom. There are two DoFs for the wrist, and five fingers with in total twelve degrees of freedom. A dexterous work set allows the hand to maintain a stable grasp while manipulating an object. The mapping results on this hand are particularly good because control is intuitive and simple, due to the obvious similarities to the human

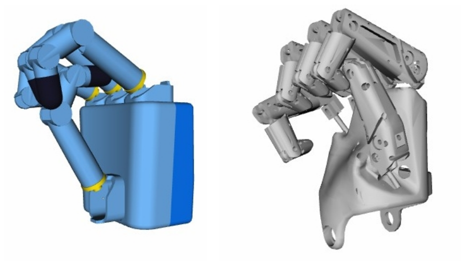


Figure 4.7: Two collision examples for the DLR and Robonaut hand. Here, we can see that thumb finger is locked. To solve the situation the fingers have to be unfolded in correct order.

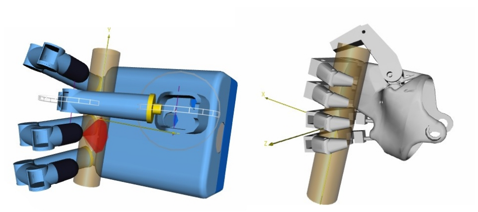


Figure 4.8: Two examples of mapping to avoid collisions for the DLR and Robonaut hand while grasping a cylinder. Here, the thumb finger is not locked by the others.

hand. Even in 14-DoF control the ANN has no problems to map the finger positions of the human hand to joint values. The training scheme is similar to the DLR-scheme, where the thumb sensor controls the Robonaut thumb and the other four finger can be divided into sections. In our training scheme the little finger or the index finger has to control three fingers simultaneously. Of course, it is possible to demonstrate different behaviors, for example, letting the index and little finger control two fingers each. As for the DLR hand, we are constrained here by the number of sensors. The underlying methodology can easily be extended if more sensors are available. For this hand we obtained an average RMS DoF error of 1.11 degrees.

For this manipulator there is also a risk of collision. To cope with this, some DoF limitations are given to the direct mapping. Some of the training postures for the Robonaut hand can be seen in Fig. 4.9.

4.1.4 Object Grasping

While the free grasp mapping allows the user to form any posture with the robot hand, its accuracy may not be sufficient to grasp objects. The average DoF error increases when all fingers are closed simultaneously. To reduce the error, additional training examples are provided, that show the robot specific grasp mappings, not just hand posture mappings.

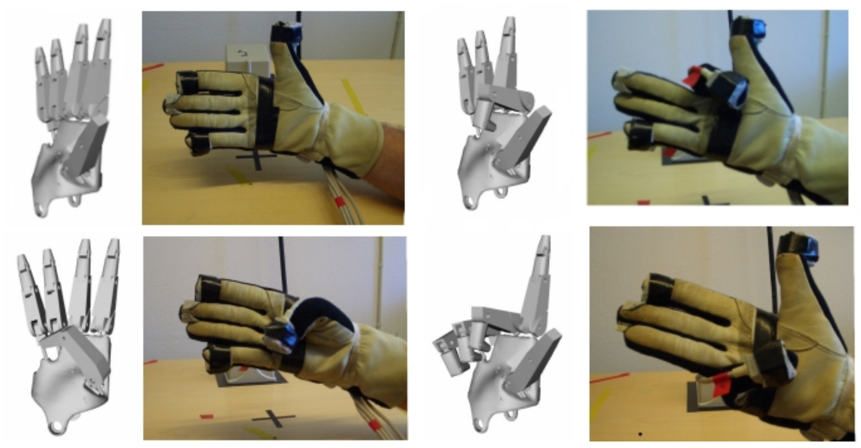


Figure 4.9: Example training postures for the Robonaut hand: open hand, thumb closed, index finger closed and little finger closed.

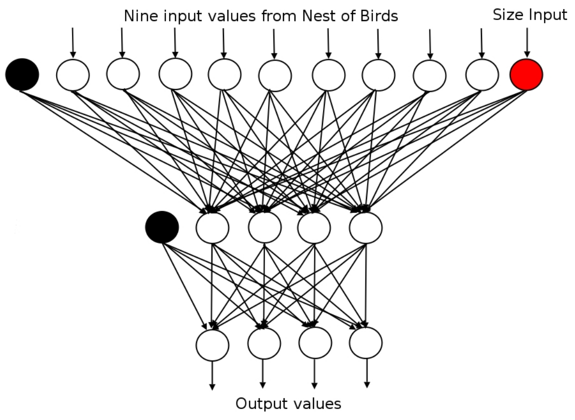


Figure 4.10: The extended network, with an added input neuron for the size of the object.

However, this will only teach the robot how to grasp that specific object. To obtain a more generalized network, we introduce a 10th input neuron, indicating the object size as shown in Fig. 4.10.

At this point we assume the object to consist of a basic shape with the form of a cylinder or a cube. The size indicates the object radius for a cylindrical object, or the side of a cubic object. Thus, we need separate networks for grasping cylinders and cubes. Examples of the additional training images are shown in Fig. 4.11. Four cylindrical and five cubic objects were used to train the system.

Fig. 4.12 show the test errors for three different grasps using the Barrett hand. As seen,

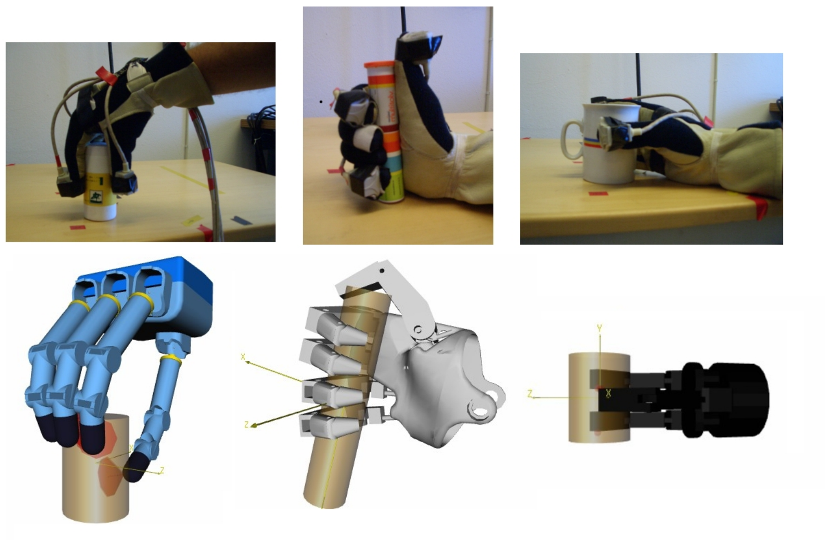


Figure 4.11: Three examples of training postures for grasping cylinders. For each hand, the system is trained on several cylinders.

the errors are lower for the mapping based on object shape. To further emphasize this, we present Fig. 4.13 which shows the posture of the Barrett hand as it is grasping a cube.

4.1.5 Conclusion

In this section, we have presented a way of mapping human hand postures to robot hand postures using an artificial neural network. The results show that for the grasping task, the system benefits from having training examples of different grasps, in addition to different hand postures. This system could be used for performing object manipulation in a teleoperative setting. However, a more advanced data glove with haptic feedback is probably necessary for a robust system, as the user would then know when the object has been grasped by the robot. This also requires the robot hand to be equipped with tactile sensors. The next section is also related to teleoperation, but with a higher degree of robot intelligence. The robot executes the grasping task autonomously just given a human example grasp execution.

4.2 Autonomous Grasping Based on Human Advice

Automatic grasping of objects is one of the most challenging research topics within robotics. The grasping process depends on many different modules working together, and each module is a hard research problem. The following modules must be present to enable a successful grasp of an arbitrary object:

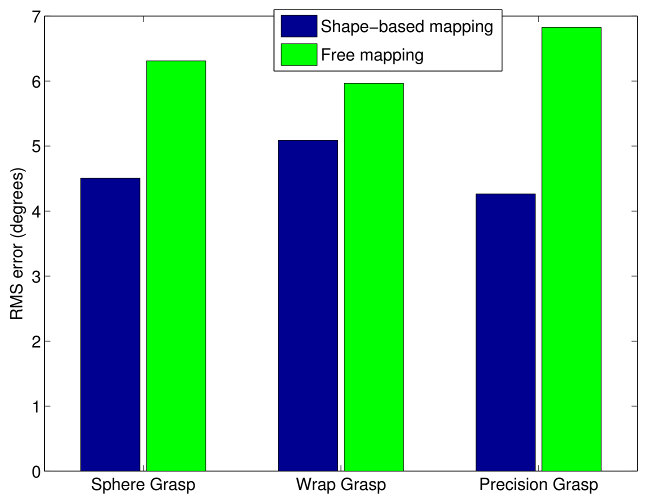


Figure 4.12: The RMS error for three Barrett grasp types on a cube. As seen the error is lower for the shape-based mapping. The two right bars correspond to the scene depicted in Fig. 4.13.

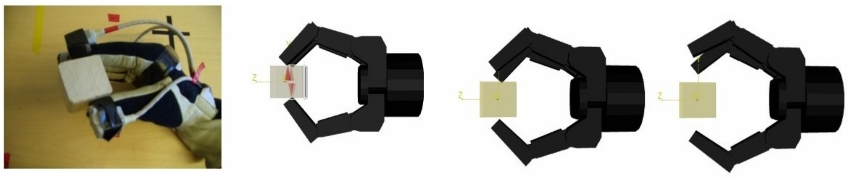


Figure 4.13: Training and test results for Barrett precision grasp on a cube. The two left images show the training postures. The two right images show the result using shape-based mapping (low error) and free mapping (high error).

- Object Detection/Recognition - Before the robot can begin to manipulate objects, it has to find the object.
- Pose Estimation - To successfully place the robot fingers at the correct location, the pose estimation of the object must be very exact. In this work, we simplify the problem by assuming the object to be standing on a table. Thus, it is only necessary to estimate three DoFs.
- Tactile Sensing - Vision alone is often not sufficient to grasp an object. The robot must be able to adjust its grasp during the whole manipulation sequence in order

to secure the grasp without causing the object to fall. If the object is heavy it may slip. The robot should automatically detect such a problem and change the grasp accordingly.

These methods rely on the assumption that the object is known to the robot in advance. Grasping an unknown object is much harder. The problem is that it is not easy to estimate the pose of an unknown object, unless it belongs to some known primitive shape class, e.g., a box. Furthermore, grasping an object without knowing its pose is quite hard, and it is also difficult to automatically evaluate whether a grasp is successful or not. In this work we have mostly concentrated on the simpler, but still very difficult problem of grasping known objects.

In this chapter, robotic grasping sequences are performed combining a learning by demonstration framework with semi-autonomous grasping. Let us start by a short motivation for the system design. Consider a human and a robot each standing in front of a table, on which a set of objects are placed, Fig. 4.14. A specific action is then demonstrated to the robot. That action may be moving (*pick up/move/put down*) an object. The robot recognizes which object has been moved and where using visual feedback. The magnetic trackers on the human hand, provide information that enables the robot to recognize the grasp type used. The robot should then reproduce or imitate the action induced by the human. As we showed in Ekvall and Kragic (2005b), a set of actions can be recorded as a whole task. This task can then be repeated by the robot at later time point.

We design and evaluate a system for automatic grasp generation and fine control, that can be used in the above scenario. The approach is evaluated in simulation using the Barrett hand and the Robonaut hand, presented in Section 4.1. In addition, it is shown how dynamic simulation can be used for building grasp experience and for the evaluation of grasp performance.

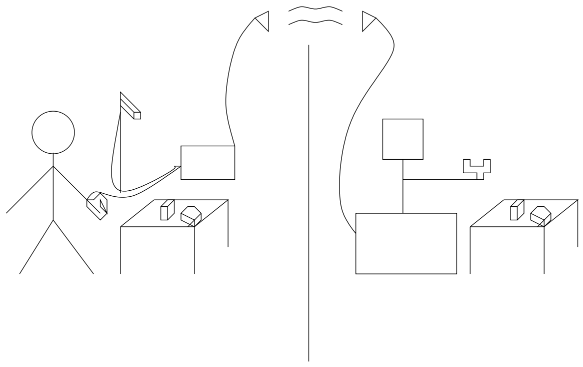


Figure 4.14: Left: A human demonstrates object manipulation actions to the robot. A camera and data glove equipped with magnetic trackers provide sensory input for task recognition. Right: The robot uses this information to reproduce the demonstrated action using its own frame of reference.

We shortly review the components currently used in our system:

1. Object Recognition and Pose Estimation
   Estimation of the objects' poses before and after an action enables the system to identify *which* object has been moved *where*. For object recognition and pose estimation, we used the RFCH-techniques described in Section 3.4. In this work, it is assumed that the objects are resting on a table. The pose can hence be represented by three parameters (x, y and ϕ).

2. Grasp Recognition
   A glove with magnetic trackers provides hand postures which are in a grasp recognition system. The system is described in detail in Section 4.3. The position of the hand is used to segment the grasp actions.

3. Grasp Mapping
   A fixed defined grasp mapping scheme maps the human grasps to robot grasps as presented in Section 4.4.2.

4. Grasp Planning
   The robot selects a suitable grasp controller. The object will be approached from the direction that maximizes the probability of reaching a successful grasp. This is presented in more detail in Section 4.4.

5. Grasp Execution
   A semi-autonomous grasp controller is used to control the hand from the planned approach position until a force closure grasp is reached, Section 4.4.

The evaluation of the system proposed in this work is performed using a modified and extended version of the robot grasp simulator GraspIt! (Miller and Allen, 2000) to allow for repetitive experiments and statistical evaluation. We strongly believe that the results of the experimental evaluation facilitate further development of robot grasping systems.

4.3 Grasp Recognition

Grasp recognition can be based on either the final posture of the hand or the entire grasp sequence including the arm motion. While the former alternative seems as more natural and is commonly used by other researchers we consider the latter alternative important since it allows grasp *intention* recognition. Also, the decision is made on the measurements obtained during the whole manipulation sequence which adds robustness to the system. There is no need to identify the best time point for hand posture recognition.

The steps involved in grasp recognition are the following: i) Obtain information about the user's fingertip locations or finger joint values, ii) Extract the key information and obtain features, iii) Demonstrate different grasps and build a model for each grasp, iv) Classify new grasps by comparing the features extracted with stored models. Each of these steps are described in detail in the following sections.

4.3.1 Applications

Grasp recognition is of great importance for many applications:

- **Programming by Demonstration**
  Grasp Recognition is used in simple PbD systems where the system not only records where an object has been moved, but also with what grasp the object was grasped with. Depending on the object's *affordances*, certain grasp types are more useful. This knowledge can be transferred to the robot through grasp recognition.

- **Prosthesis Construction**
  One of the longterm goals in robotics and neuroscience is to interpret the signals directly from the human brain and use them to control a prosthesis. Today, there is a range of electrically driven hands available. Many of these are activated by the user contracting a single muscle with the resulting electromyographic signal activating the opening or closing of a single degree-of-freedom device. Related to our work, if the user's intention can be recognized, the prosthesis may automatically prepare for the grasp by shaping the end effector to match the object. With the ambition to improve the prostheses, (Ferguson and Dunlop, 2002) have performed grasp recognition from myoelectric signals, measured on intact muscles. Six basic grasps from Cutkosky's grasp hierarchy (Cutkosky, 1989) are recognized, with an 80 % classification rate.

- **Object recognition**
  While object recognition generally is based on visual feedback, we believe the robustness may be increased by integrating a visual system with a grasp recognition system. A cup on the table may be recognized using, for example, its shape and color information. Integrating this with the recognition of *action* that can be performed on the object will further increase the robustness. This is closely related to the current research trend in *perception-action mapping*, (Ernst and Bulthoff, 2004).

4.3.2 Related Work on Grasp Recognition

Various methods for grasp recognition have been tried, but few have investigated recognition based on fingertip positions. Friedrich *et al.* (1999) present an approach for grasp classification using neural networks and Cutkosky's taxonomy. Similarly, Kang and Ikeuchi (1993) classify user-demonstrated grasps using an analytical method. They recognize grasp classes roughly similar to, but more elaborate than those proposed by Cutkosky. The common drawback of these techniques is that only static posture analysis is performed which means that an ideal time point for analysis of the hand configuration has to be extracted from the demonstration by other means before classification can be done.

Grasp recognition, gesture or sign language recognition commonly use Hidden Markov Models (see Section 2.2.1) for sequence analysis. In an approach similar to ours, Bernardin *et al.* (2003) recognize dynamic grasp sequences using HMMs. They present a system that uses both hand shape and contact point information obtained from a data glove and tactile sensors to recognize continuous human grasp sequences. The sensor fusion, grasp classification and task segmentation are made by a HMM based recognizer that distinguishes 14 grasp types from Kamakura's taxonomy. The HMM is based on finger joint values, from the Cyber Glove. An accuracy of up to 92.2% for a single user system, and 90.9% for a multiple user system is achieved. It should be noted here that in the multiple user case, the system has to be trained for every user. One particular contribution of our work is that the recognition rates we report consider system evaluation on users who did not participate in the training process.

A problem similar to grasp recognition is gesture recognition. Liang and Ouhyoung (1998) presents a Taiwanese sign language recognition system that consists of 4 HMMs, each recognizing the gesture from posture, position, orientation and motion features, respectively. The HMMs are fed with features extracted from the data glove. Posture features consist of a flexion of 10 finger joints of the hand. Azimuth, elevation, and roll of palm reported by a Polhemus 3D tracker are used for orientation estimation. Another approach is taken by Lee and Kim (1999), who designed a vision based system to recognize continuously executed sequences of gestures without prior detection of breakpoints. The segmentation is done automatically by a HMM recognizer and a set of 10 gestures is classified. Starner and Pentland (1995) report the use of HMMs for the recognition of the American sign language (ASL).

4.3.3 Grasp Recognition: Two Methods

In this section, we present two methods for grasp classification: i) using HMMs with fingertip locations and ii) observing the hand movement trajectory during different grasps. Here, Cutkosky's grasp taxonomy (Cutkosky, 1989) has been used for the definition of different grasp classes. In this hierarchy, grasps are sorted into 16 different classes. Each class is either a power- or precision-grasp, circular or prismatic. Six of these classes are redundant using our measuring device, and therefore 10 grasp types are considered as shown in Fig. 4.15. In this work, it is assumed that the start and the end time point of a grasp sequence are known. In (Ekvall and Kragic, 2004), we have shown how automatic segmentation can be done using continuous probability estimation.

4.3.4 Grasp Classification Based on Fingertip Positions

A grasp sequence is modelled as: i) Prepare: the hand is opened and formed to match the object to be grasped. ii) Approach: the hand is moved until palm contact is detected (power grasps) or until the object lies between the fingers (precision grasps). iii) Hand closure: the fingers are wrapped around the object until the contact force exceeds a certain value. We have to note here that, if only the final measurements of the fingertip positions are considered, the classification would be highly dependent on the object size. Here, a statistical, model based approach using Hidden Markov Models (see section 2.2.1) is considered to model each grasp as a sequence of hand postures. By analyzing each posture during the grasp sequence, increased robustness is gained.

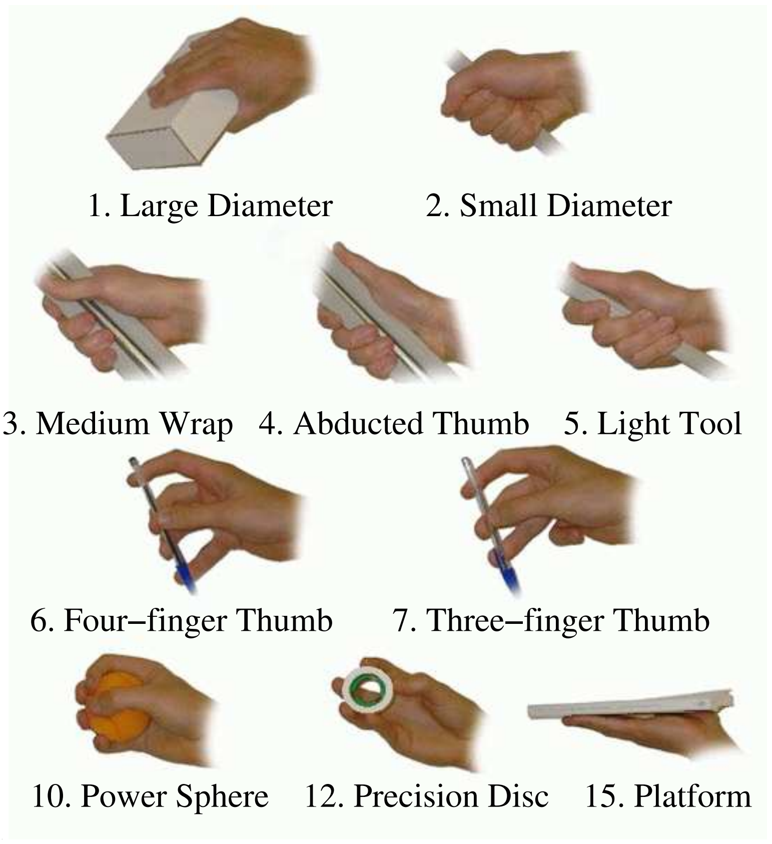


Figure 4.15: Grasps considered in our system: the numbers correspond to their position in Cutkosky's grasp taxonomy.

4.3.4.1 Grasp Modeling

The three fingertip locations are quantized into one of 50 cluster positions. Each cluster is 9-dimensional, since we have three fingertip locations with three dimensions each. The clusters are found using K-means clustering on the training data. The number of clusters should be low enough to gather enough statistical data of each posture, but high enough to facilitate good accuracy. We have experimentally found that 50 clusters satisfy both of the above requirements.

We assign a left-to-right structure to our HMM and each model operates in parallel. Each state in the HMM represents a hand posture and a grasp sequence is considered as a sequence of hand postures. With 50 different hand postures, a single grasp sequence is typically constructed of 5 postures. Thus, we have used 5 states in each HMM. The Baum-

Welch algorithm is used to train the HMM and only a few iterations are required to obtain stable values for $\lambda_j(A_j, B_j, \pi_j)$. Once the HMM is trained, it describes the probability of observing certain hand postures in different states. Since the state transitions are also modelled, the velocity of a performed grasp is not an issue.

For classification, we select the most probable model, given the observations, i.e., the most probable grasp, given the sequence of hand postures. Since no *a-priori* model information is available, the model with the highest likelihood is chosen:

$$class = \underset{i}{\operatorname{argmax}} \left(P(\mathbf{o}_1 ... \mathbf{o}_N | \lambda_i)\right) \tag{4.3}$$

4.3.5 Grasp Classification Based on Arm Movement Trajectories

In the second grasp classification approach considered here, we evaluate the importance of the arm trajectory in the grasping process. The reference sensor used in the HMM based method, mounted on the back side of the hand, is also used for trajectory recording. Six parameters are captured during the grasp: x, y and z describing the position, and three Euler angles (azimuth, elevation and roll) describing the orientation of the sensor. In addition, the velocities of each of these parameters are calculated at each time step. An *ArmState* is described by these 12 parameters, and is represented as $\mathbf{v}$ = (v_1, v_2,..., v_{12}).

4.3.5.1 Position and Orientation Invariant Trajectories

Besides the grasp type, the grasping trajectory depends on the initial arm configuration, as well as the object position. Ideally, we would like the grasp trajectory of a certain grasp to be the same when the user is sitting and standing up and independent of the position of the object to be grasped. The final position and orientation of the hand provides information of the object's location, and by transforming all trajectories to the relative coordinate system given by the final hand pose $(x, y, z, \alpha, \beta, \gamma)$ of each trajectory, trajectories with different start- and end-positions may be compared.

4.3.5.2 Similarity of Two Trajectories

An arm movement trajectory is a sequence of *ArmStates*. Let us first consider the trajectory of a single parameter in *ArmState*, i.e. $v_1(n)$. If the trajectory consists of N time steps, it be viewed as a point in an N-dimensional space. The naive dissimilarity measure $D(\mathbf{x}, \mathbf{y})$ for two trajectories $\mathbf{x}$ and $\mathbf{y}$ of equal length, is the Euclidean distance, defined as

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{n=0}^{N-1} (x(n) - y(n))^2} \tag{4.4}$$

The lower $D(\mathbf{x}, \mathbf{y})$, the more similar trajectories $\mathbf{x}$ and $\mathbf{y}$. The above dissimilarity measure has a couple of disadvantages. First, it is very sensitive to time shifting. A shift in time for one trajectory will make the two trajectories uncorrelated. Second, it does not handle shifts in space very well either, i.e. a position trajectory will have low similarity to an identical trajectory, shifted a few cm.

By first transforming the trajectory to the Fourier domain, the above problems are avoided. Transformation is performed using the Discrete Fourier Transform (DFT), defined as

$$X(\mu) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{\frac{-\mu n 2i\pi}{N}} \tag{4.5}$$

where $X \in [0,\ 1, ...,\ N-1]$ are the frequency components of the signal x(n). The dissimilarity measure used for the two transformed trajectories is

$$D(\mathbf{X},\mathbf{Y}) = \sum_{\mu=1}^{fc} |(X(\mu) - Y(\mu)|^2 \tag{4.6}$$

By ignoring the first component $X(0)$, which corresponds to the average position, space shift invariance is achieved. Also, $X(\mu)$ is not time dependent, so time shifts is not a problem. Furthermore, only the first $fc < N$ coefficients are taken into account, which corresponds to a low-pass filtering of the signal $x(n)$. Extensive experiments have shown that the useful information regarding arm movement trajectories is concentrated to the first few Fourier coefficients. Using cross validation, we found $f_c = 2$ to be optimal.

The Nest of Birds measurement system provides samples with approximately constant rate, 30 Hz. However, for the DFT this is not satisfactory as the samples have to be taken at equidistant intervals. Since this is not possible, we have used interpolation to calculate the assumed measurement values at the desired time points. As arm movements are in general slow compared to the measurement rate, it is realistic to interpolate the measurements. A fixed number of samples k are used, so the same number of Fourier components will be available for each trajectory. We found $k = 20$ to be optimal, using cross validation.

The total dissimilarity function for two trajectories $\mathbf{u}$ and $\mathbf{v}$ is described by

$$D(\mathbf{u},\mathbf{v}) = \sum_{i=1}^{12}\sum_{\mu=1}^{fc} |(X_i(\mu) - Y_i(\mu)|^2 \tag{4.7}$$

If we, in addition, allow each parameter i to be weighted with weight w_i, the dissimilarity function is

$$D(\mathbf{u},\mathbf{v}) = \sum_{i=1}^{12} w_i^2 \cdot \sum_{\mu=1}^{fc} |(X_i(\mu) - Y_i(\mu)|^2 \tag{4.8}$$

However, the Fourier transform is not scale-invariant. In order to compare trajectories taken by different users, with different arm lengths, a normalization step is required. (4.9) describes the general normalization procedure for all position parameters. The maximum and minimum parameter values were determined from the training data. These values represent the grasp workspace, which has the approximate size of 1 m^3.

$$v_i^{norm}(n) = \frac{v_i(n) - \min(v_i)}{\max(v_i) - \min(v_i)} \tag{4.9}$$

4.3.5.3 Parameter Weighting

To evaluate which of the parameters are the most important for grasp recognition, we assigned weights to the parameters to let their importance vary. By optimizing the weights using cross validation, the recognition rate for the test set increased. For weight optimization, we used a procedure similar to threshold accepting (Dueck and Scheuer, 1990), in which random changes are made in order to find the global maximum.

The final weights determine the importance of different parameters for grasp recognition. One drawback with threshold accepting is that it is not guaranteed to find the global maximum. The risk of getting stuck in a local maximum is reduced because of the random steps taken. However, the cross-validation score is still a good predictor of how high the recognition rate will be. Therefore, we have tried 50 stable weight sets, and selected the set with the highest cross-validation score.

4.3.5.4 Trajectory Classification

For classification of arm trajectories, we have chosen an instance-based approach in which all training examples are considered when a trajectory **u** is to be classified. A training example **v** votes for its corresponding object class i in vote slot c_i, with a vote calculated from the dissimilarity function (4.8) as

$$c_i = c_i + \frac{1}{\sigma\sqrt{(2\pi)}} e^{\frac{-D(\mathbf{u},\mathbf{v})}{2\sigma^2}} \tag{4.10}$$

where σ is the standard deviation of the Gaussian kernel. We found $\sigma = 0.1$ to be optimal, using cross validation. When all training examples have voted, the object class is estimated by selecting the grasp type with the highest number of votes:

$$class = \operatorname*{argmax}_{i=1}^{N_{objects}} c_i \tag{4.11}$$

4.3.6 Experimental Evaluation

The methods are evaluated separately. As mentioned, 10 grasp types from Cutkosky's grasp taxonomy have been considered in two different settings.

In the first setting, the grasps are trained and tested by the same subject. For each grasp type, 20 grasps are executed, from which 10 are used for training and the remaining 10 are used for testing. In the second setting, training is done by two users, each demonstrating every grasp 10 times. The testing is then performed on a third subject, this time executing each grasp 20 times. The latter experiment evaluates how well the system generalizes to new users with different hand shapes.

During training and testing, the same grasp is taken on several different objects, illustrated in Fig. 4.16. For example, the *Large Diameter* grasp is taken on both the binder and the box.

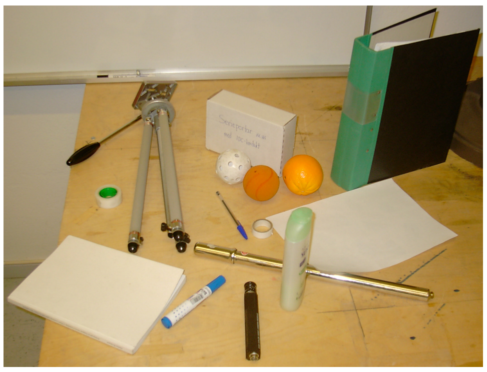


Figure 4.16: The objects grasped in the experiments.

4.3.6.1 Optimal Parameter Weights

From the training examples, the optimal parameters for the trajectory based method were found using threshold accepting. Fig. 4.17 shows the importance of the different 12 parameters.

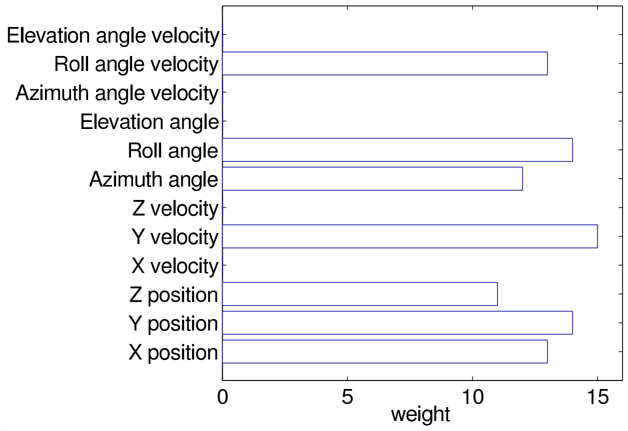


Figure 4.17: The final parameter weights used by the trajectory based method.

It can be seen that some of the parameters are completely ignored, while the others have about the same weights. It is interesting to note that in terms of the position velocities, only the Y velocity is important, which is the approach velocity to the object. The Z position corresponds to up and down motion, and the X position corresponds to left and right motions, relative to the object.

4.3.6.2 Performance

We have evaluated the performance of each method with respect to the number of different grasps to recognize. As the number of possible grasps increases, the difficulty of the recognition task increases. Fig. 4.18 shows the recognition rate for the single user setting.

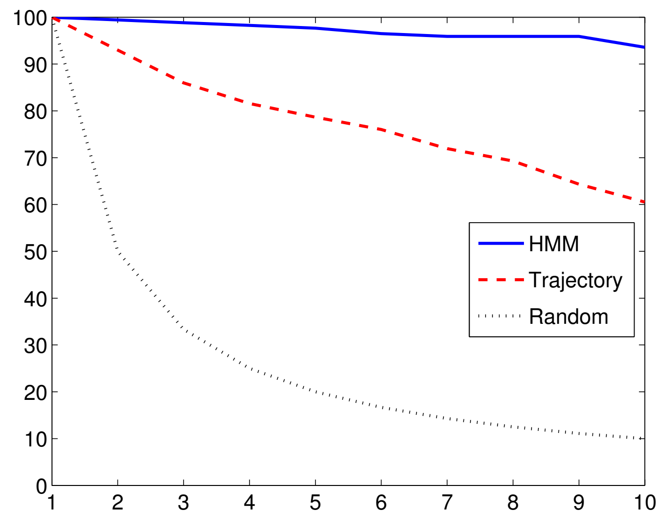


Figure 4.18: The recognition rate in the single user setting, mapped to the number of grasp types available.

As expected, the HMM based method which relies on fingertip locations outperforms the trajectory based method. The HMM method is able to recognize 94% of the grasps when there are 10 possible grasp types to choose from. For comparison, we also tested with a simple posture classification approach, in which just the mean fingertip location of the final posture were stored during training. Instead of analyzing the entire grasp sequence, we compared the final posture of the test sequence with the final postures of the training sequences. The grasp is then classified as the nearest neighbor in the training data, measured by the Euclidean fingertip distance. With this approach we achieved a recognition rate of 86%.

Further on, we evaluated the performance in the multiple user setting. In this setting, two users trained the system, which then was tested on a third user. Fig. 4.19 shows the recognition rate.

The performance is not as good as in the single user setting. The system performs well for up to five grasp types, with a recognition rate of about 80%. For 10 grasp types, the recognition rate is 65%. However, the system was trained on a single user and fur-

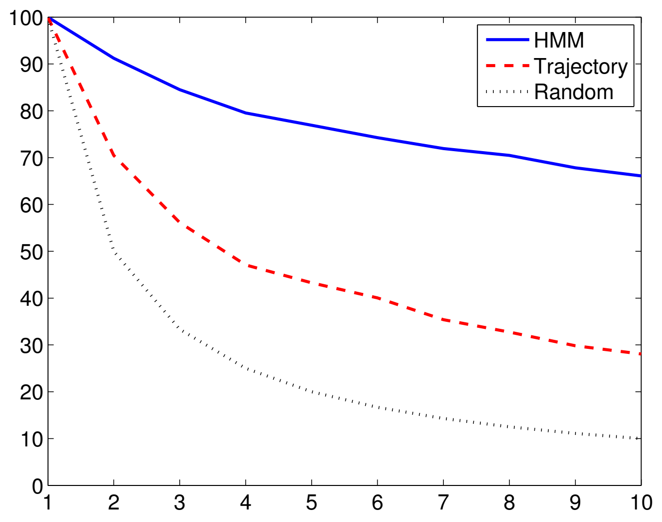


Figure 4.19: The recognition rate in the multiple user setting, mapped to the number of grasp types available.

ther experiments are required to evaluate if training on multiple users will increase the generalization capabilities of the system.

4.3.6.3 Misclassification Analysis

As seen in Fig. 4.20, most misclassifications are the *Power Sphere* and the *Precision Disc* being classified as the *4-Finger Thumb*. It is easy to reason why this happens. First, the grasp types are very similar in terms of fingertip positions, see Fig. 4.15. Second, the way the users take these grasps vary among the subjects. Some users have a wide spread angle when taking the *4-Finger Thumb* grasp, while others keep their fingers close together. This, combined with the inaccuracy of the sensors (up to 1 or 2 cm), and the different hand shapes of the subjects, makes it very hard to distinguish between these grasps using only fingertip positions.

4.3.6.4 Grasp Intention

We have also investigated how well the methods recognize grasp intention. When a human grasps an object, the hand starts to prepare for the grasp immediately. This behavior can be exploited for grasp intention recognition. In these experiments we have only used five grasp types, as some are very similar in the beginning of the grasp sequence. We have evaluated how well the methods perform when trained and tested only on the first part of

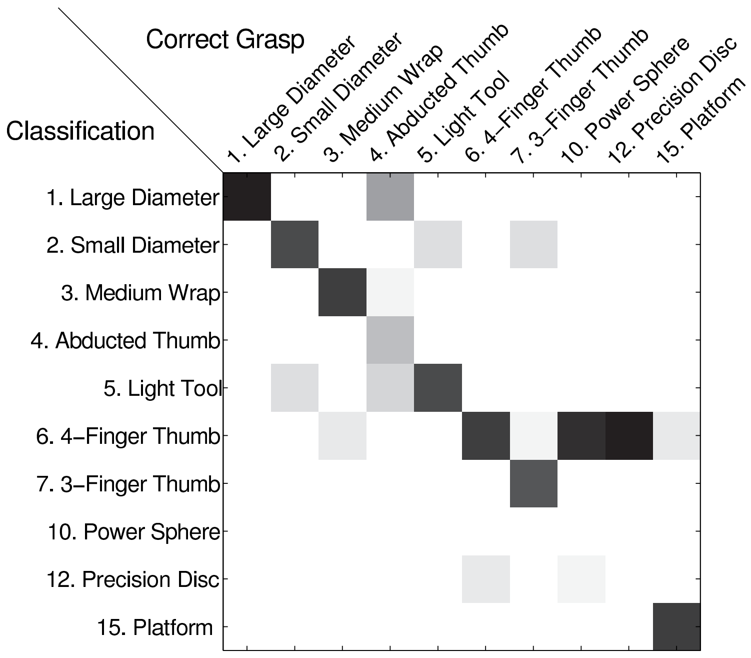


Figure 4.20: The confusion matrix, for 10 grasp types, and multiple users.

the grasp. As seen in Fig. 4.21, the grasps can be recognized with a 95% accuracy already at 60% of grasp completion, in the single user setting. With multiple users, 80% of the grasp has to be seen before the system performs satisfactory. Note that this evaluation was done using five specific grasps, while the results depicted in Fig. 4.18 were generated as the mean over the results from all the possible subsets of grasp types drawn from the total ten grasp types.

4.3.7 Conclusion

In this section, we have presented two methods for grasp recognition. The first method relies on fingertip movements where only three fingers have been considered due to limitations of the measurement device. The second method is based on the hand trajectory taken during the grasp. The position and orientation as well as the velocity of the hand is tracked during grasp execution. The experimental evaluation has shown that as expected, the hand posture is more important than the trajectory taken, but surprisingly good results were obtained from the hand trajectory.

We have also found that the user reveals much information during the grasp and this information can be used to estimate the grasp type before it is taken. When the system is

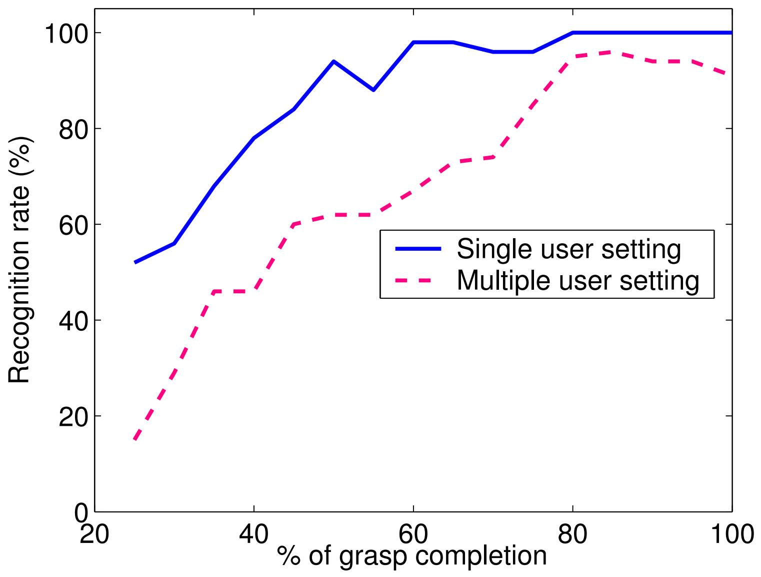


Figure 4.21: The recognition rate mapped to the percentage of grasp completion, for five grasp types.

used on the same user that has trained the system, it can recognize almost all grasps at just 60% completion. Grasp recognition is further used in Section 4.4 in order to aid the robot in choosing a grasp type.

4.4 Autonomous Grasping Inspired by Human Demonstration

One of the main challenges in the field of robotics is to make robots ubiquitous. To intelligently interact with the world, one of the key abilities that robots need to have is to manipulate objects. Typical environments in which robots will be deployed, such as a house or an office, are dynamic and it is very difficult to equip robots with an ultimate and general grasp planning capability. Planning a grasp is difficult due to the large search space resulting from all possible hand configurations, grasp types, and object properties that occur in regular environments. Another important question is how to equip robots with capabilities of gathering and interpreting the necessary information for novel tasks through interaction with the environment in combination with minimal prior knowledge.

In this section we present a method for automatically grasping known objects. The entire grasp sequence is thoroughly evaluated in a simulated environment, from learning a grasp to actually reaching it, including dynamic simulation of the grasp execution. We also discuss the necessary requirements for evaluating this approach in a real setting. It should be noted here that the problems arising are not only related to the mapping between different kinematic chains for the arm/hand systems but also to the quality of the object pose estimation delivered by the vision system.

The contributions of the work presented in this section are:

- A suitable grasp is related to object pose and shape and not only to a set of points generated along its outer contour. This means that we do not assume that the initial hand position is such that only planar grasps can be executed as proposed in (Morales *et al.*, 2004). In addition, grasps relying only on a set of contact points may be impossible to generate on-line since the available sensory feedback may not be able to estimate the exact same points on the object's surface once the pose of the object is changed.

- The choice of the suitable grasp is based on the *experience*, i.e. it is learned from simulated grasps on the object, and also from the human by defining the set of most likely hand preshapes with respect to the specific object. A similar idea was investigated in (Miller *et al.*, 2003) but only one robotic hand and four grasp preshapes were considered. We evaluate both Barrett (Townsend, 2000) and Robonaut hand, (Lovchik and Diftler, 1999). Since grasp preshapes are generated based on recognition of human grasps it makes them more natural. This is, of course, of interest for humanoid robots where the current trend is to resemble human behavior as closely as possible.

- Finally, we evaluate the quality of different grasp types with respect to inaccuracies in pose estimation. This is an important issue that commonly occurs in robotic systems. The reasons may be that the calibration of the vision system or hand-eye system is not exact or that a detailed model of the object is not available. We evaluate how big pose estimation error different grasp types can handle.

4.4.1 Related Work on Grasping

Considering specifically object manipulation tasks, the work on automatic grasp synthesis and planning is of significant relevance, (Miller *et al.*, 2003; Pollard, 2004; Morales *et al.*, 2004; Platt Jr *et al.*, 2003). The main issue here is the automatic generation of stable grasps assuming that the model of the hand is known and that certain assumptions about the shape of the object can be made. Example of assumptions may be that the full and exact pose of the object is known in combination with its (approximate) shape, (Miller *et al.*, 2003). Another common assumption is that the outer contour of the object can be extracted and a planar grasp applied, (Morales *et al.*, 2004). The work on contact-level grasps synthesis concentrates mainly on finding a fixed number of contact locations with no regard to hand geometry, (Bicchi and Kumar, 2000; Ding *et al.*, 2000).

Taking into account both the hand kinematics as well as some *a-priori* knowledge about the feasible grasps has been acknowledged as a more flexible and natural approach towards automatic grasp planning (Pollard, 1994; Miller *et al.*, 2003). In (Pollard, 1994), a method for adapting a given prototype grasp of one object to another object, such that the quality of the new grasp would be at least 75% of the quality of the original one was developed. It has to be, however, pointed out that this process required a parallel algorithm running on supercomputer to be computed efficiently. The method proposed in (Miller *et al.*, 2003) presents a system for automatic grasp planning for a Barrett hand by modeling an object as a set of shape primitives, such as spheres, cylinders, cones and boxes in a combination with a set of rules to generate a set of grasp starting positions and pregrasp shapes.

With respect to dynamic grasping and manipulation control, previously presented results include catching a ball or playing the piano using the robotic DLR Hand (Borst *et al.*, 2003). Exchanging a light bulb has been shown using the Utah/MIT hand (Jägersand, 1997). High speed grasping has also been demonstrated in (Namiki *et al.*, 2003). In terms of grasping systems, relevant ideas have been presented in (Horswill, 2000).

4.4.2 Grasp Mapping

The ANN-based hand posture mapping presented in Section 4.1 can not be used in this scenario. It is designed to be used online in a teleoperative manner, while in this section we deal with autonomous grasping. Instead, we define a fixed grasp mapping scheme based on grasp type similarity. During execution of the robot grasp, the robot's *virtual fingers* (Kang and Ikeuchi, 1997) acts much in the same way as the human fingers. It has been argued that grasp preshapes can be used to limit the large number of possible robot hand configurations. This is motivated by the fact that, when planning a grasp, humans unconsciously simplify the grasp choice by choosing from a limited set of prehensile postures appropriate for the object and task at hand (Napier, 1956). Related to robotics, Cutkosky (1989) classified human grasps needed in a manufacturing environment and evaluated how the task and object geometry affect the choice of grasp.

As shown in Section 4.3 the current grasp recognition system can recognize ten different grasp types. The mapping scheme showed in Table 4.1 was defined. This table is visualized in Fig. 4.22.

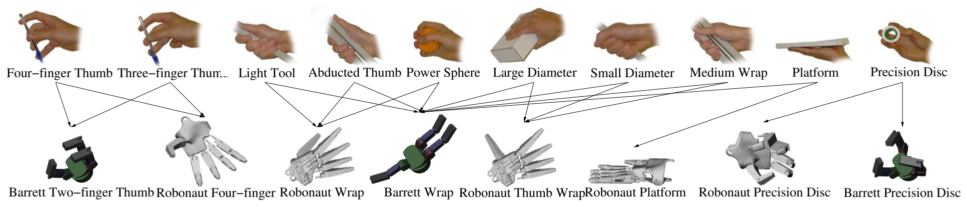


Figure 4.22: Initial robot hand postures for different grasp types.

It has to be noted here that the robot grasp types do not refer only to hand postures, but to grasp execution schemes. Such a scheme includes the initial position, the *approach vector*, the robot hand closing sequence, controllers for corrective movements, etc. Hence, different strategies are used to grasp an object dependent on the grasp type. Fig. 4.22 illustrates the initial hand postures for each of the controllers.

Human Grasp	Barrett Grasp	Robonaut Grasp
Large Diameter	Barrett Wrap	Robo. Thumb Wrap
Small Diameter	Barrett Wrap	Robo. Thumb Wrap
Medium Wrap	Barrett Wrap	Robo. Thumb Wrap
Abducted Thumb	Barrett Wrap	Robonaut Wrap
Light Tool	Barrett Wrap	Robonaut Wrap
Four-finger Thumb	Barrett Two-finger Thumb	Robo. Four-finger
Three-finger Thumb	Barrett Two-finger Thumb	Robo. Four-finger
Power Sphere	Barrett Wrap	Robonaut Wrap
Precision Disc	Barrett Precision Disc	Robo. Precision Disc
Platform	Barrett Wrap	Robonaut Platform

Table 4.1: The mapping of human grasps to robot grasp controllers. The left column is a selection of human grasps from Cutkosky's grasp hierarchy.

4.4.3 Grasp Controllers

There are two basic grasp controllers in the system: Power Grasp and Precision Grasp. There are eight variations of these, three for the Barrett hand and five for the Robonaut hand. The difference lies in the initial grasping position and the finger control during closure. The two basic controllers are described below.

- **Power Grasp**
  First the initial hand posture is set according to the grasp type recognized from the human demonstration. Then the hand approaches the object with the palm towards the object. Once contact is detected, all fingers close simultaneously. When a fingertip contact is detected, that finger stops its closure. This grasp type will in general give a contact at the palm for a more stable grasp. Depending on the grasp type, the joint angle speed may be different for each joint, causing for example the thumb to close more slowly.

- **Precision Grasp**
  This controller is similar to the Power Grasp, but with an added dimension. Once contact is detected, typically at one of the finger tips, the hand can retract a predefined distance and then close all fingers simultaneously. This allows the robot to combine tactile sensing with computer vision.

The grasp approach vector is defined relative to the object's pose and center. Other object shapes may require evaluation of several approach vectors, e.g. the object top and bottom, or one or more for each object feature.

For both these controllers, a grasp modification step would increase the grasp success rate. Once the grasp is established, the robot could calculate the grasp quality, estimate the local surface patches of the object, and correct its finger positions for a more stable grasp. However, this is a difficult research problem, and outside the scope of this work.

4.4.4 Grasp Planning

Grasp planning is the process of finding stable grasps for a certain object without actually grasping them. The planning is done for each object and each robot end-effector, in the grasping simulator GraspIt! A 3D-model of the object is approached with the end-effector model from many different angles, and a grasp quality measure is used to measure the grasp obtained from each angle. All results are stored in a grasp experience database. Our approach to grasp planning is similar to the work described in (Miller and Allen, 2000) and (Morales *et al.*, 2006a). However, those approaches require the object to be composed of object primitives, explained later in Section 4.4.4.3. For each primitive, a set of predefined candidate grasps and approach directions are evaluated. In this work, we do not use predefined approach vectors but instead evaluate many approach vectors for each object. The vectors may target the center of the object or one of the primitives, but we believe the information gained from previous grasping of a primitive has limited value when the primitive is attached to an object. The most significant contribution compared to the above papers is that we evaluate how the grasps perform under imperfect pose estimation. We also evaluate how the primitive representation affects the results as the real object is grasped.

The planning is performed using a simple search technique where many different approach vectors are tested on the object. The training can be performed on either the primitive object model or the full object model, and in the experiments we have evaluated both methods. A more detailed model will in general result in higher grasp quality on the real object.

For power grasps, three parameters (θ, ϕ, ψ) are varied describing the approach direction and hand rotation. For precision grasps, a fourth parameter d, that describes the retract distance when contact is detected, is added. The number of evaluated values for the variables are θ=9, ϕ=17, ψ=9, d=6. For the precision grasps the search space was hence 8262 grasps which required about an hour of training using kinematic simulation.

4.4.4.1 Grasp Quality Measures

To evaluate grasps, the 6-D convex hull spanned by the forces and torques that the grasp can resist is analyzed using GraspIt! (Miller and Allen, 1999). The ε-L1 quality measure is the smallest maximum wrench the grasp can resist and is used for power grasps. For precision grasps, a grasp quality measure based on the volume of the convex hull was used, volume-L1. These grasp quality measures obviously require full knowledge of the world, and can thus only be used in simulation.

4.4.4.2 Grasp Retrieval

At run-time, the robot retrieves the approach vector that result in the highest quality grasp from the grasp experience database. As the highest quality grasp is not necessarily the most robust with respect to position and model errors, the grasp should be chosen taking also those parameters into account, see Section 4.4.5.2. Because of robot kinematic constraints and possible non-free paths toward the object, all approach directions are not suitable at task execution time. Thus, the robot searches the database only for directions that are applicable in the current situation. In a Programming by Demonstration scenario, the mapping from human to robot grasp is one-to-one. But if the robot acts autonomously, i.e. *explores* the environment and performs grasps on unknown objects, the grasp type is not defined and the best grasp can be chosen from among all possible grasps.

4.4.4.3 Training on Object Primitives

As mentioned earlier, a model of each object is necessary for training the grasp planner. It is not likely that the robot will be able to acquire these models by itself within the near future. However, it is not unrealistic to assume that it will be possible to extract *shape primitives* using computer vision. The idea is to represent each object by its appearance and shape primitives. The appearance is used for recognition, and the primitives for grasp planning. Several primitives build up an object, and the primitives can be of different basic shapes: truncated cone, sphere, box, cylinder etc. To evaluate such an approach, we have designed a primitive representation of each object in our scenario, see Fig. 4.23. The grasp controllers are then evaluated in the GraspIt! simulator. For evaluation purposes, we have modelled all objects in 3D, also seen in Fig. 4.23.

4.4.5 Experimental Evaluation

The experimental evaluation presented here focuses on the gripping task, as the grasp recognition and object recognition models have been evaluated in previous chapters. The grasping evaluation is also done in the simulator, since we do not have access to the real manipulator hands. Five objects shown in Fig. 4.23 were modelled and put in the simulator. The real objects were placed on a table, with a mounted camera to monitor the world state. Objects and grasps were recognized and fed to the virtual robot, which then grasped the same object with the desired grasp. In this section we provide both qualitative experiments that show how some human grasps are mapped to the manipulator, and quantitative experiments that show how small errors in pose estimation affect the end result. As pointed out by Morales *et al.* (2006b), the robustness of a grasp to positioning the effector has not been widely addressed in the literature.

4.4.5.1 Grasp Mapping Examples

Fig. 4.24 shows some examples of the best grasps obtained when the robot is free to choose any approach direction. Fig. 4.24(h) shows an example of a failed grasp, due to a simulated error in pose estimation.

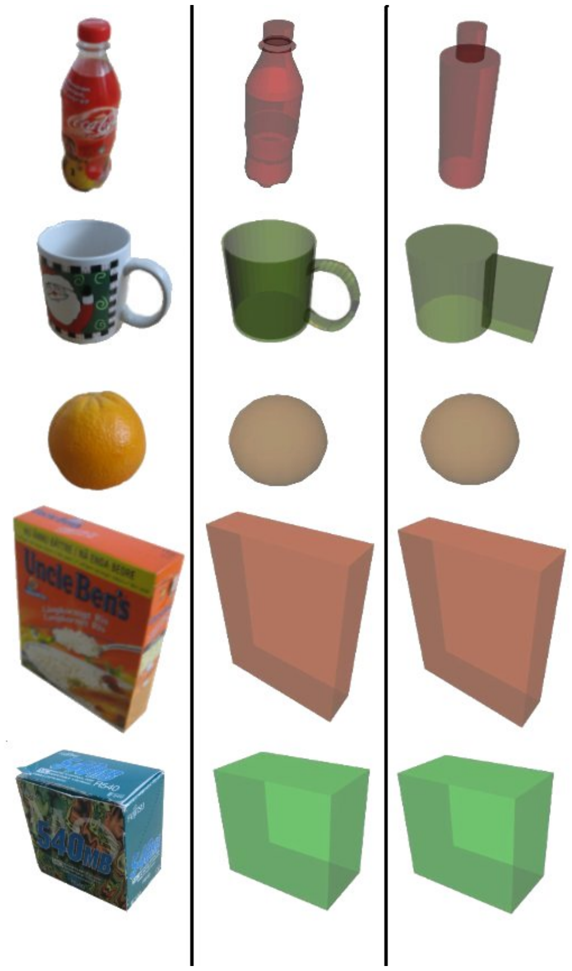
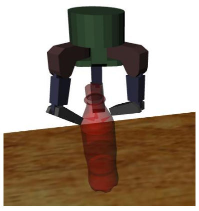


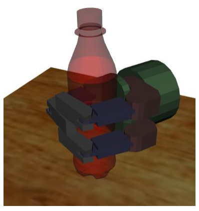
Figure 4.23: Left: The real objects. Center: The modelled objects. Right: The object primitives used for training.

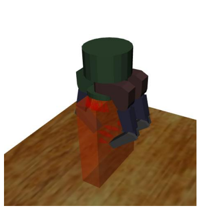
4.4.5.2 Simulated Pose Estimation Inaccuracies

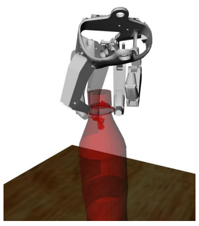
To evaluate the performance under imperfect pose estimation, we have simulated errors in pose estimation by providing an object pose with an offset. As pointed out in (Morales *et al.*, 2006b), the robustness of a grasp to positioning the end-effector has not been widely addressed in the literature.

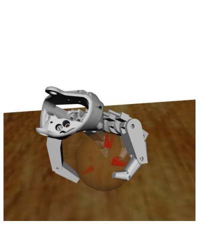
In the experiment, the target object was placed on the table and the robot performed 50 grasps using different approaches. The robot hand position was between each grasped translated a certain distance in a random direction. As a result, the robot interpreted the

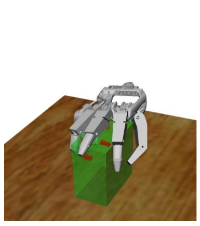

(a) Barrett Precision Disc

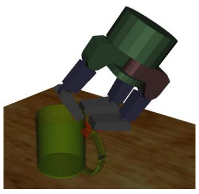

(b) Barrett Wrap

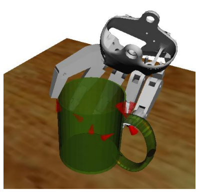

(c) Barrett Wrap


(d) Robonaut Precision Disc


(e) Robonaut Thumb Wrap


(f) Robonaut Thumb Wrap


(g) Barrett 2-finger Thumb


(h) Robonaut 4-finger Thumb

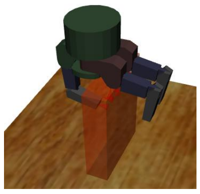

(i) Failed Barrett Wrap

Figure 4.24: Examples of grasp executions for various grasp types and objects. (a)-(h) shows successful grasps, while (i) shows a failed grasp due to a simulated error in pose estimation. The contact friction cones are plotted in red.

situation as if the object (and possibly table) was in another position than that for which the grasp was planned. This was repeated for five different vector lengths: 0, 1, 2, 3, and 4 cm. In total, the robot grasped the object 250 times from a total of 201 different positions.

Fig. 4.25 - 4.27 show the grasp success rates for various grasps and objects, under

increasing error in position estimation. As expected, power grasps are more robust to position errors than precision grasps. The precision grasps target details of object, e.g., the bottle cap or the ear of the mug. Thus, the grasps are much more sensitive to position inaccuracies.

It is clear that the Barrett hand is more robust than the Robonaut hand, likely due to its long fingers. The exception is the grasping of the mug, Fig. 4.26, where the Robonaut Four-finger Thumb grasp is the best.

The bottle and the mug have been trained both using a primitive model and using the real model (see Fig. 4.23). Training on the primitive model does not decrease the grasp success rate much, especially not for the bottle. However, the primitive model of the mug is, unlike the real mug, not hollow, which causes problems for some of the precision grasps trained on the primitive.

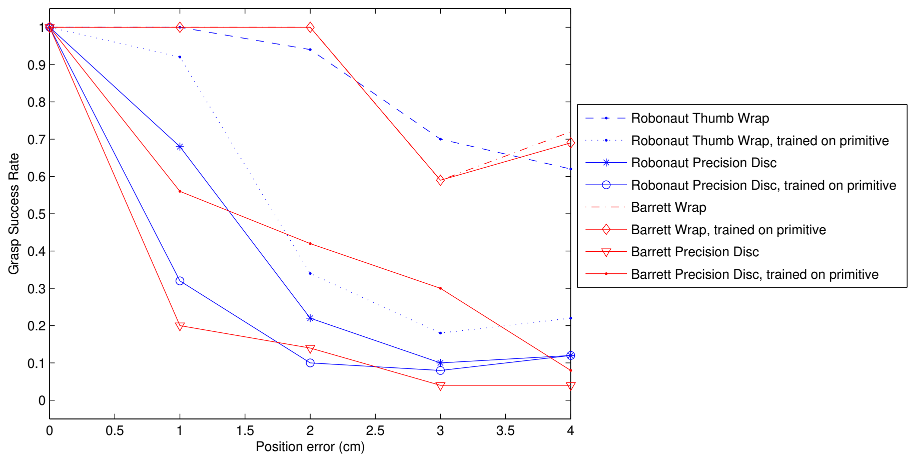


Figure 4.25: Grasp bottle.

In (Tegin *et al.*, 2006) we present a force controller for the Barrett hand. Using this, grasping in a dynamic environment was simulated. The results in Fig. 4.28 shows that this allows for even worse pose estimation while still finding successful grasps.

We have also evaluated how a simulated rotation estimation error affects the result. For each object and grasp type, we tested much the object could be rotated before the grasp failed. As expected, for symmetric objects like the orange and the bottle this type of error has no effect. However, for the other objects we found that the difference in rotation error robustness is big. Table 4.2 shows the rotation tolerance for various objects and grasp types. For two of the Robonaut grasps on the mug, the rotation is no problem, with a perfect success rate. For one of the Barrett grasps on the mug, the rotation is absolutely crucial and cannot withstand a small rotation inaccuracy. Thus, this type of grasp should be avoided for this object.

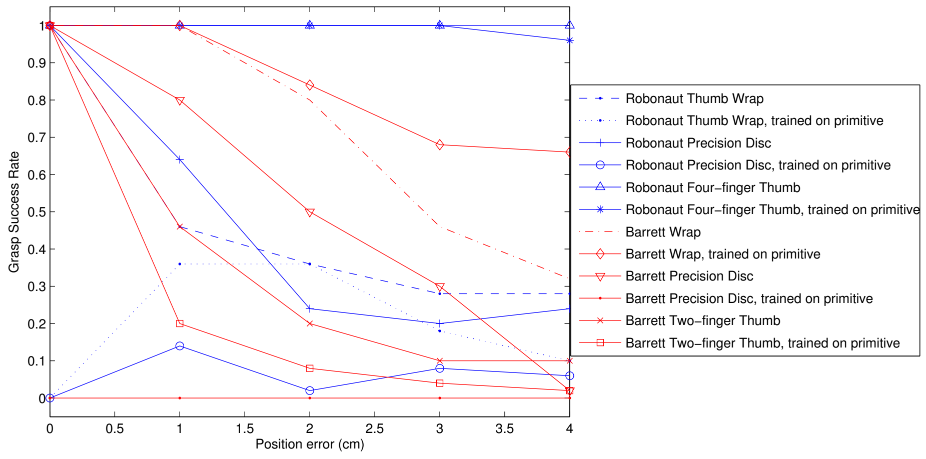


Figure 4.26: Grasp mug.

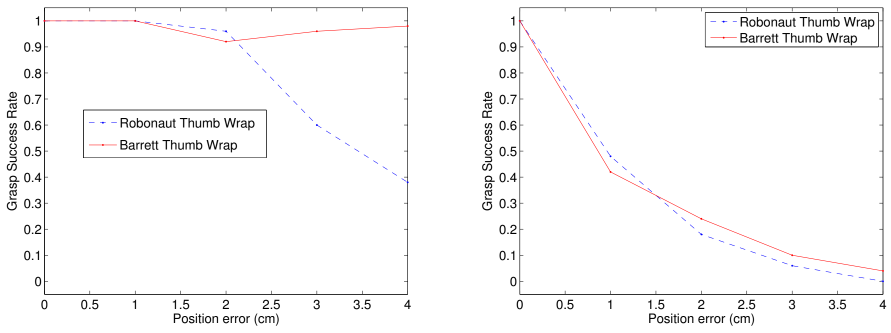


Figure 4.27: Left: Grasp orange. Right: Grasp zip disk packet.

4.4.6 Conclusion

The success rate of the presented system depends on the performance of four subparts: i) object recognition, ii) grasp recognition, iii) pose estimation of the grasped object, and iv) grasp execution. As demonstrated in Section 3.4, the object recognition rate for only five objects is around 100 %, and the grasp recognition ratio is about 94 % for ten grasp types. Therefore, the performance in a static environment may be considered close to perfect with respect to the first steps. As the object pose and possibly the object model is not perfectly known, some errors were introduced that indicate the needed precision in the pose estimation given a certain grasp execution scheme. The results suggest that for certain tasks stable grasping is possible even when the object's position is not perfectly known.

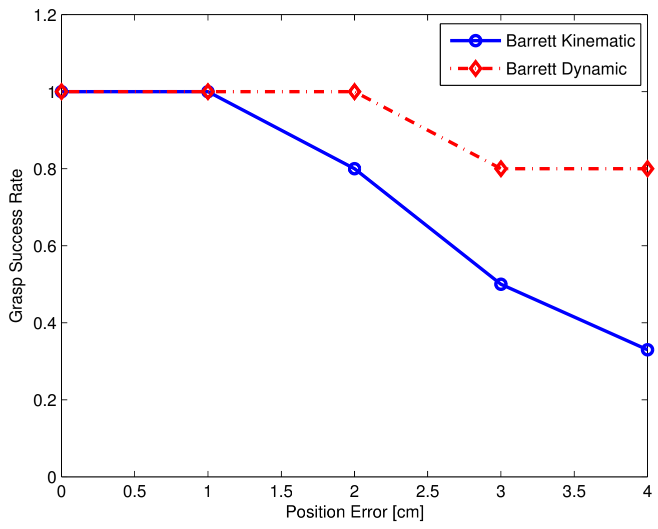


Figure 4.28: Grasp success rates for grasping of the rice packet, using different controllers. As seen, a force controller (Barrett Dynamic) improves the result and allows the hand to stabilize the grasp even if the approach vector is poor. (Due to some problems with the simulator, a limited number of samples was used in the evaluation of dynamic grasping. For the 0, 1, 2, 3, and 4 cm random displacement, the number of trials were 50, 14, 18, 18, and 12 respectively (instead of 50). Still, these samples were truly random and we believe that the number of trials is high enough to draw conclusions.)

If a high quality dynamic physical modeling is essential, for example when grasping compliant objects or for advanced contact models, other simulation tools may be more suitable, see e.g. (Ferretti *et al.*, 2006). But since grasping often can be performed rather slowly, and as model errors for mass properties, sensors, friction, and in actuator and gear models are often quite large, second order dynamic effects can be ignored in the control design and instead considered as small disturbances (Prattichizzo and Bicchi, 1998).

4.5 Discussion

In this chapter, we have presented two approaches for robotic grasping. In the first, we focus on robot control and teleoperation using a natural device: a data glove which let the user control the robot hand with his own hand. In the second approach, the grasping is autonomous but the choice of grasp is inspired by a human demonstration, from which the grasp type is automatically recognized. This grasp type is then mapped to the robot and the grasp is executed in a simulated environment.

Object and Grasp Type:	Rotation Error Tolerance (degrees):
Mug, Robonaut Precision Disc	4
Mug, Robonaut Thumb Wrap	180
Mug, Robonaut Four Finger Thumb	180
Zip Disc Packet, Robonaut Thumb Wrap	17
Rice Packet, Robonaut Thumb Wrap	2
Zip Disk Packet, Barrett Thumb Wrap	3
Rice Packet, Barrett Thumb Wrap	17
Mug, Barrett Thumb Wrap	12
Mug, Barrett Precision Disc	0
Mug, Barrett Two Finger Thumb	6

Table 4.2: The rotation error tolerance for different objects and grasp types.

Methods for generating robot grasps based on object models, shape primitives and/or human demonstration have been presented and evaluated. While many factors are important, the focus lies on one of the main challenges in automatic grasping; the choice of the approach vector. The approach vector depends on the object as well as on the grasp type. Using the proposed methods, the approach vector is chosen not only based on perceptional cues, but on experience that some approach vectors will provide useful tactile cues that result in stable grasps. Moreover, a methodology for developing and evaluating grasping schemes has been presented. Focus lies on obtaining stable grasps under imperfect vision, something that has not been thoroughly investigated in the literature.

Simulating results was necessary for generating insight into the problem and for performing the statistical evaluation for the grasp experience, since i) the world must be reset after each grasp attempt, and ii) computing the grasp quality measure requires perfect world knowledge. The proposed strategies have been demonstrated in combination with tactile feedback and hybrid force/position control of a robot hand. The functionality of the proposed framework for grasp scheme design has been shown by successfully reaching force closure grasps using a Barrett hand and dynamic simulation.

The grasp experience database contains not only a record of success rates for different grasp controllers but also the object-hand relations during an experiment. In this way, we can specify under what conditions the learned grasp strategy can be reproduced in new trials.

The results of the experimental evaluation performed in a simulated environment suggest that the outlined approach and tools can be of great use in robotic grasping, from learning by demonstration to robust object manipulation. Future work includes further grasp execution scheme development and implementation. Furthermore, to ensure truly secure grasping outside the simulator, the grasping scheme must also comprise a grasp quality evaluation method that does not use information available in simulation only. Preferably such a measure would also depend upon the task at hand.

Chapter 5

Task Level Learning from Demonstration

Robot task learning has during the past years received significant attention (Kuniyoshi *et al.*, 1994; Atkeson and Schaal, 1997; Schaal, 1999; Matarić, 2000; Ogawara *et al.*, 2002; Breazeal and Scassellati, 2002; Friedrich *et al.*, 1999; Ehrenmann *et al.*, 2001; Aleotti *et al.*, 2004; Ekvall and Kragic, 2005a) and it has been recognized that more natural programming interfaces are necessary to allow ordinary users to teach robots new tasks. Motivated by the fact that imitation enables humans to easily learn skills others have already mastered, the robotics community has taken upon this idea and used it in the design of some of the recent task learning systems for robots. From the viewpoint of task learning in humans it is known that such a strategy where a teacher's demonstration is used as a starting point of learning significantly speeds up the process and reduces the amount of trial-and-error steps. In the AI community, much of the work has concentrated on the high-level planning and conceptual representations of skills and state changes using propositional or first-order logic. In robotics, such an approach to learning has been considered in frameworks of Learning by Imitation and Programming by Demonstration (PbD), (Friedrich *et al.*, 1999; Chen and Zelinsky, 2003) where sensory based task representation and task analysis have been recognized to represent the basis for development of robust and flexible learning systems. Apart from being able to learn from and imitate humans, improving robots' capabilities through longterm training and feedback has been recognized as the milestone in both communities.

For the work presented here, we consider a service robot scenario in which we want to enable a robot to learn and refine representations and understanding of complex domestic tasks. The robot should be able to learn either by a direct learning process with a human teacher, or through observation where the robot is able to extract new information and learn new skills just by looking at a human. A PbD interface can be used to interpret teacher's demonstration and generate control commands required by the robots for completing the task. Such a programming interface is natural for humans since it does not require any programming skills and can potentially be used to program very complex tasks.

Naturally, an important issue to deal with is that the initial task setting will change between the demonstration and execution time. A robot that has to set up a dinner table may have to plan the order of handling plates, cutlery and glasses in a different way that previously demonstrated by a human teacher. Hence, it is not sufficient to just replicate the human movements but the robot i) must have the ability to recognize what parts of the whole task can be segmented and considered as subtasks so to ii) perform online planning for task execution given the current state of the environment.

The main contribution of this chapter is the use of a task planning approach in combination with robot learning from demonstration strategies. The important problem considered here is how to instruct or teach the robot the essential order of the subtasks for which the execution order may or may not be crucial. As an example, in a table setting scenario, the main dish plate should always be under the appetizer or a soup plate and the order in which these are placed on the table is important. One way of addressing this problem is to demonstrate a task to the robot multiple times and let the robot learn which order of the subtasks is essential. In relation, additional contributions of this chapter are the state generation and constraints identification methodologies based on multiple human demonstrations.

The proposed methodologies are evaluated in the framework of robotic object manipulation tasks. Learning such tasks is considered a hard problem since robots have a very limited world knowledge to start with and are mainly constrained by the type of available sensory modalities. For humans, much of the background knowledge is innate and one demonstration is often sufficient. This is not a case when considering a robot. There are two possible directions here: either we let the robot assume that the actions can be executed in any order, or that the actions have to be executed in the same order as the demonstration. The first alternative requires that the human instructs the robot of all the possible task constraints during the demonstration. In this chapter, we have chosen the latter alternative since it allows the robot to learn from multiple observations and improve the task model over time, without the risk of violating a contraint.

5.1 Motivation and Related Work

For humans, one of the fundamentals of social behavior is the understanding of each others intentions, skill transfer and learning through interaction and observation. Skill learning in humans have been well studied and most common forms of teaching are observation, demonstration, physical guidance or verbal instruction, (Schmidt and Lee, 1999). When learning a complex task, a significant amount of information has to be processed. It has been shown that in such cases, the viewer does not know which details are important for the task outcome and that the appropriate level of detail should be provided for successful learning, (Schmidt and Lee, 1999).

In a similar way, the robot task learning have to be made easy and flexible. Some of the open questions are:

- How should the robot be instructed complex tasks when the temporal order for some of the subtasks is important but unimportant for others?

- How should objects and actions that can be performed on them be represented?
- Should the task goal be represented by the final goal state or should it be learned by considering temporal dependencies between the subtasks?
- How does the representation and number of demonstrations facilitate the generalization of task models?

These are some of the question we are trying to answer in the work presented here. In general, we would like to teach the robot some useful tasks such as how to set up a dinner table, slice a cucumber or put in dishes into a dishwasher. Setting up a dinner table task can be viewed as a sequence of pick-and-place object manipulation subtasks, (Ekvall and Kragic, 2005b). For this task, the robot is required to recognize objects, grasp them and put them on the table in specific relation to each other. The relationship between objects can be represented relative to one object, e.g. main plate. Chopping a cucumber is more difficult since the robot has to learn that a knife should be held in a specific way related to the cucumber. Different from the first example, the relative relationship between objects changes during task execution. Mobile manipulation tasks such as mail delivery (Jensfelt *et al.*, 2005) include constraints, for example that the mail has to be collected before it can be delivered but the order of delivery may be irrelevant. In summary, for some of the tasks a specific order of subtasks is required and for some it is not. One contribution of our work is that the problem of learning tasks that include object manipulation is solved by identifying the goal state and the spatio-temporal constraints of the task.

Many of the current robot instruction systems that deal with programming by demonstration are based on a single demonstration. However, the robot should be able to update the initial task model by observing humans or another robot performing the task multiple times. In other words, we need a task level learning and planning system that builds constraints automatically identified from multiple demonstrations. This problem has previously been considered in regard to sub-optimality in demonstration (Friedrich and Dillmann, 1995; Chen and Zelinsky, 2003) where different sources of sub-optimality have been recognized: where the human demonstrates unnecessary, incorrect or unmotivated actions; where there is a choice of scenario regarding when to apply an action; where the user does not know enough about the task and thus the actions are demonstrated wrongly. Some of the solutions to these problems have been studied in (Chen and Zelinsky, 2003) where the sub-optimality on the task-level is considered as noise and removed before any programming of the robot takes place. Differently from the work presented in (Chen and Zelinsky, 2003; Lefebvre *et al.*, 2005) that generates plans autonomously using geometric properties of objects or instructions provided by the demonstrator, we deal with learning and refinement of high-level tasks based on a set of underlying capabilities already available to the robot. In particular, we are interested in evaluating scenarios where the robot is able to reproduce a task based only on a desired outcome or final goal of the task, preferably also generalizing from multiple demonstration trials.

A similar problem has been studied in a robot navigation scenario, (Nicolescu and Mataric, 2003) where a task is represented by the alternate paths shown during the teaching phase. Compared to our work, the robot is still required to follow one of the human demonstrations unless the task is refined. In the work presented here, we focus on object manipulation tasks which require that objects are represented in relation to each other. Thus, our work differs both in the state representation and the task generalization.

In (Ogawara *et al.*, 2002), generation of task models based on multiple human demonstrations is presented. *Essential interactions* that represent the important hand movements during a manipulation task are identified. Then the relative trajectories corresponding to each essential interaction are generalized by calculating their mean and variance. High variance means arbitrary motion is allowed, while low variance means strictly precise motion is required. The learned trajectories are stored in the task model, which is used to reproduce a skilled behavior. It is important to point out that trajectories are related to the essential interactions with the manipulated objects, meaning that many different trajectories corresponding to the same object manipulation are represented. This makes the method less flexible, as it requires the world state to be roughly the same as during the demonstration. In our work, we do not store the hand trajectories, but instead what has been done. The robot can then reproduce the results of the human demonstration at execution time by planning a sequence of actions to reach the goal state.

5.2 System Description

The general outline of the system is shown in Fig. 5.1. The teacher demonstrates the task and and the robot makes observations based on visual input. After a learning trial, explained in more detail later on, the robot first plans and then executes the task using visual input and grasp planning.

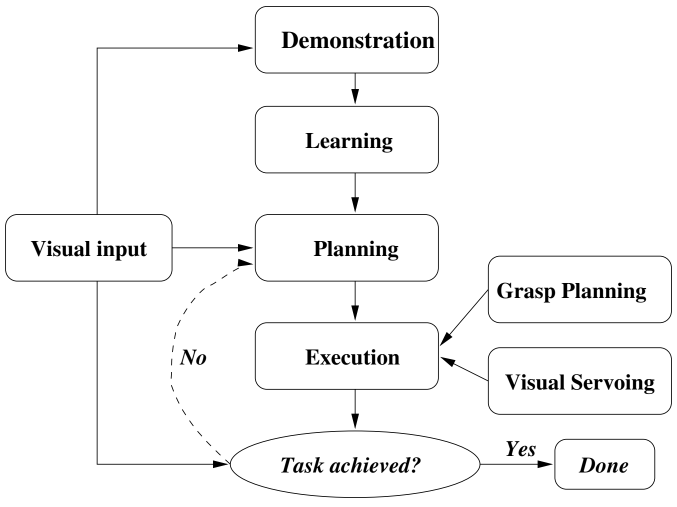


Figure 5.1: Integration of task learning from demonstration and task level planning.

In the current system, task learning can be performed in three different ways:

- **Imitation Learning**
  The term *imitation learning* is commonly used to represent task learning at a low level by considering reproduction of trajectories and/or robot joint configurations, (Ude, 1999; Riley *et al.*, 2000; Ruchanurucks *et al.*, 2006). In our work, with the imitation learning we denote the task reproduction process where the robot is given only the task goal and it tries to achieve the same result.

- **Learning in Dialog with Teacher**
  In human teaching, it is common that the teacher demonstrates the task once, while continuously explaining each step. We investigate a similar approach in our system by allowing the teacher to add some constraints to the task while performing it. By doing this, the robot is explicitly instructed what not to do and it is thus able to avoid solving the task in a wrong or inefficient way.

- **Generalizing from Multiple Observations**
  We also believe that the robots should be able to improve or learn new tasks not only through a direct teaching process but also by observing humans performing tasks in everyday settings. Multiple observations of the same task can be utilized to form a more general and thus flexible model of the task by autonomously identifying the spatio-temporal constraints of the subtasks or detect the irrelevant subtasks.

In this chapter, we model and evaluate all of the above approaches.

5.2.1 Experimental Platform

The experimental platform is a PowerBot from MobileRobots Inc. It has a non-holonomic differential drive base with two rear caster wheels. The robot is equipped with a 6-DoF robotic manipulator on the top plate. It has a SICK LMS200 laser scanner mounted low in the front, 28 Polaroid sonar sensors, a Canon VC-C4 pan-tilt-zoom CCD camera with 16x zoom on top of the laser scanner and a fire wire camera on the last joint of the arm, see Fig. 5.2.

5.3 Task Level Planning

Task planning is the problem of finding a sequence of actions to reach a desired goal state, (Russel and Norvig, 2003). This is a classical AI problem which is commonly formalized using a suitable language to represent task relevant actions, states and constraints. Naturally, the robot has to be able to plan the demonstrated task before executing it if the state of the environment has changed after the demonstration took place. The objects to be manipulated are not necessarily at the same positions as during the demonstration, and thus the robot may be facing a particular starting configuration it has never seen before.

Our task planner is inspired by the STRIPS planner, (Fikes and Nilsson, 1971), and is based on operations which contain several preconditions and effects. These describe

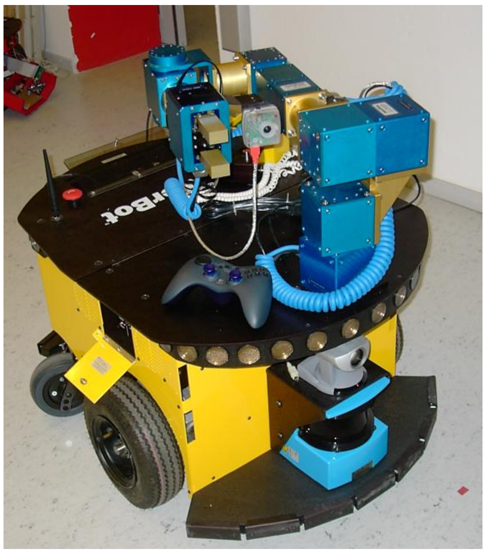


Figure 5.2: The experimental platform: PowerBot from MobileRobots Inc.

the changes of the world state once an action has been executed. The second part of the planner is the problem file, which is designed to reflect the current world state. The file contains all objects in the current scene, their locations and task defined destinations. It is automatically generated at run-time.

The domain and problem is provided to the planner in XML. The domain is formalized using first order logic, which allows us to model the changing state of the world. Currently, there are only two types of operations available to the planner, defined below.

Definition 1. Domain Operations

- *pickUp(o, l, g)* - Grasp the object *o* at location *l* using grasp type *g*.
- *putDown(o, l, g)* - Put down the object *o* at location *l* using grasp type *g*.

The world state is described with a list of predicates.

Definition 2. Domain Predicates

- *handEmpty* - indicates whether the hand is empty or not.
- *holding(o)* - indicates if the hand is currently holding object *o*.
- *objAtLoc*(o, l) - indicates if location *l* is occupied by the object *o*.
- *graspable*(l, g) - indicates if the object at location *l* is graspable with grasp type *g*.

- *collision(l1, l2, g)* - indicates if a collision would occur if a grasp *g* is applied to location *l1*, and both location *l1* and *l2* are occupied.

- *mustOccurBefore(o1, l1, o2, l2)* - A special predicate to handle temporal constraints. Indicates if object *o1* must be placed at location *l1* before object *o2* can be placed at location *l2*.

Below we define the preconditions and effects for the operators.

Operation *pickUp(o, l, g)*

Precondition: *handEmpty* ^ *objAtLoc(o, l)* ^ *graspable(l, g)* ^ ¬∃*(obj, loc)* (*collision(l, loc, g)* ^ *objAtLoc(obj, loc)*)
Effect: *holding(o)* ^ ¬*handEmpty* ^ ¬*objAtLoc(o, l)*

Operation *putDown(o, l, g)*

Precondition: *holding(o)* ^ *graspable(l, g)* ^ ¬∃*(obj) objAtLoc(obj, l)* ^ ¬∃*(obj, loc)* (*collision(l, loc, g)* ^ *objAtLoc(obj, loc)*) ^ ¬∃*(obj, loc) mustOccurBefore(obj, loc, o, l)*
Effect: *handEmpty* ^ ¬*holding(o)* ^ *objAtLoc(obj, loc)*) ^ ∀*(obj, loc)* ¬*mustOccurBefore(o, l, obj, loc)*

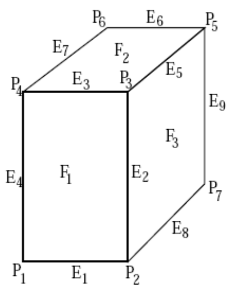
Thus, the grasp type for an object is selected automatically at run-time, based on the predicates provided to the planner by the vision system. As seen, we operate with a limited number of predefined grasps. For planning, a grasp must be chosen so that is does not cause collisions with the other objects, and also so that it is within the robot's reach. A grasp *g* at location *a* is not possible if there is a nearby location *b* occupied by another object, that would cause a collision when grasp *g* is applied to *a*. For all location pairs, the robot tests all grasps against all objects for collisions and reachability. This approach allows the robot to select the best grasp depending on the target pose of the object.

The locations are limited to the source and target locations of each object, plus a number of additional free locations that can be used for freeing up the workspace. As an example, the source location for a box is labeled *loc_box_s*, and similarly the target location is labeled *loc_box_t*. Section 5.5.1 provides an example of how the planner operates. We note here that more advanced logical languages will be considered once more complex tasks are to be modelled. One example is the Linear Dynamic Event Calculus proposed in (Steedman, 1997), a logical language that combines aspects of situation calculus with linear and dynamic logics.

In this chapter, we work with polyhedral objects but the presented methodology can be applied to a large set of shapes as long an accurate pose estimate is available. The next section shortly describes the pose estimation process.

5.3.1 Pose Estimation

As stated, pose estimation of objects is performed automatically. Here, we use a slightly different method to the one presented in Section 3.6.1. Because only a few objects that occupy a rather large part of the image are used, there is no need to first search for objects. The objects are modelled using a set of geometric primitives as shown in Fig. 5.3 where only the size has to be known in advance. In the simplest case, the primitives are the apparent object edges modelled using points, lines and polygons defined both in the camera (3D) and image (2D) space. Given the current pose of the object, hidden primitive removal is performed using back face culling (Foley *et al.*, 1990).



points	lines	polygons
$P_1(E_1,E_4,\ldots)$	$E_1(P_1,P_2)$	$F_1(E_1,E_2,E_3,E_4)$
$P_2(E_1,E_2,E_8)$	$E_2(P_2,P_3)$	$F_2(E_3,E_5,E_6,E_7)$
$P_3(E_2,E_3,E_5)$	$E_3(P_3,P_4)$	$F_3(E_2,E_8,E_9,E_5)$
$P_4(E_3,E_4,E_7)$	$E_4(P_4,P_1)$	

Figure 5.3: An object is represented with points, lines and polygons.

Due to the rich textural properties of the objects, pose estimation cannot be performed by using solely the outer contours of the object. This is why in an off-line learning stage, we store a single file representing a set of SIFT points originally presented in (Lowe, 1999). With the stored image, we also store the pose of the object corresponding to that particular view of the object. At run time, the SIFT feature detector is applied to the whole image. The detected features are then matched to the stored set of points defined for each of the objects. For planar objects, a homography based matching with robust outlier rejection (RANSAC) is used for pose estimation, as presented in (Kyrki and Kragic, 2005). The reason for this approach is that for each point on the surface, it is enough to know which facet it belongs to — the exact 3-D position of each individual point is not required. Since the pose of the object for the stored view is known, the problem of scale ambiguity related to homography decomposition is easily solved. Examples of the pose estimation process are shown in Fig. 5.4 and Fig. 5.5. We note that the pose estimation process is not perfect in some cases when only one side of the object is visible, Fig. 5.5 (left).

5.3.2 Detecting Object Collisions

The work here relates also to the path planning problem, (Bohlin and Kavraki, 2000). Compared to the task level planning, a path planner searches for a path in robot's configuration space to reach a desired configuration while avoiding obstacles, self-collisions, etc. In contrast, a task planner performs high level operations and rely on an existing low-level

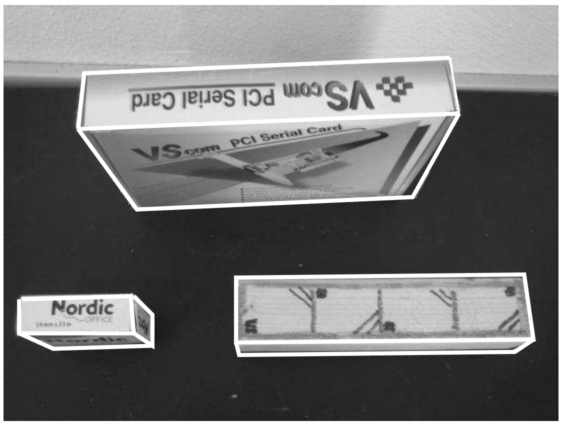


Figure 5.4: Examples of estimated poses overlaid in white.

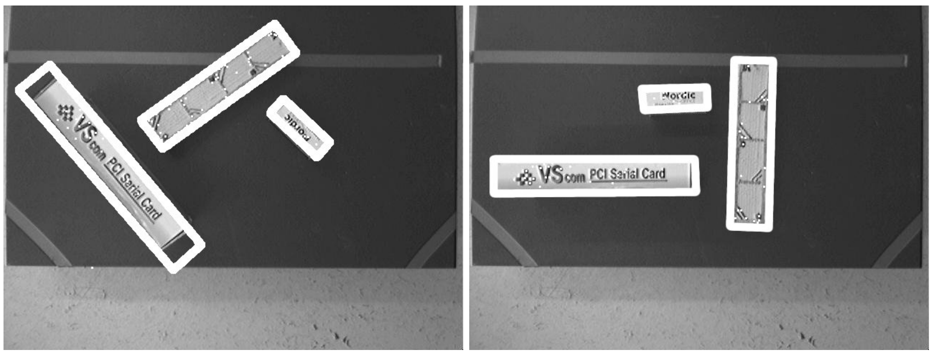


Figure 5.5: Examples of estimated poses overlaid in white. As seen in the left image, the pose estimated from a single image is not always perfect.

robot controller to carry out the necessary operations. Using a path planner for planning the entire task is not feasible as the complexity of the task would make the planner infeasible. Also, it is hard to accurately incorporate the dynamics of the actual grasping into the path planner. For task planning, it is however important to also consider the reachability of the robot. While a path planner only explores locations within the robot's workspace, a task planner operates in the task space and must at each step check that the specific world location is reachable. The robot used in this work has a very limited workspace, as shown in Fig. 5.6.

Before the planner is initiated, all possible object collisions that can occur during task execution are evaluated. This is a fast process for settings with a few objects. If many

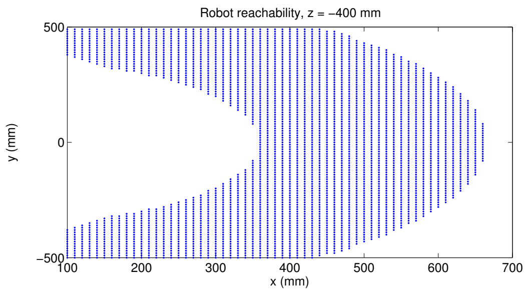


Figure 5.6: The workspace of the robot, visualized for the plane 40 cm below the base cube, where the experiments were conducted.

objects are involved in a task, techniques that limit the number of collision checks may have to be utilized, e.g., only checking nearby locations for collision. Fig. 5.8 visualizes the data available to the robot before performing collision checks. For each object, there are a number of grasping configurations that can be applied to each object and a suitable one is chosen based on the initial and destination position of the object and the current position of other objects in the environment, see Fig. 5.7. A collision check results in a list of grasps which are applicable to the object in the initial location, if its destination location is currently occupied. The collision check is performed in three steps. First, it is tested whether the two objects would collide with each other. In that case, no grasp is possible. If not, each grasp for the object in the first location is checked for collision with the object in the second location. If the grasp still does not cause a collision, it is checked whether the robot can actually perform this grasp considering its kinematic constraints and limited reachability.

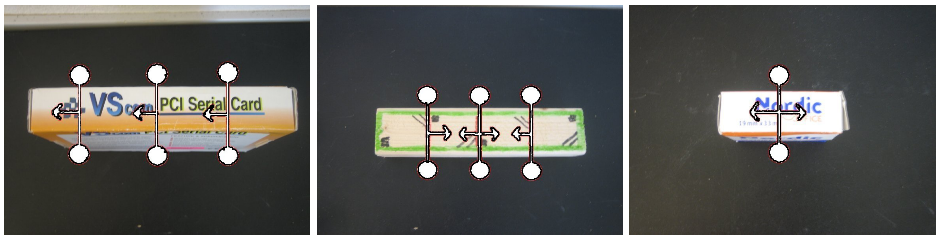


Figure 5.7: The predefined grasp types for the different objects. The box has three grasps, the wooden block has four, and the tape only two.

The collision checks are currently performed as following: two objects collide if their bodies occupy the same part of the environment. This check is done in 2D since all objects are assumed to be placed on a table in a vertical position. A grasp collides with an object if

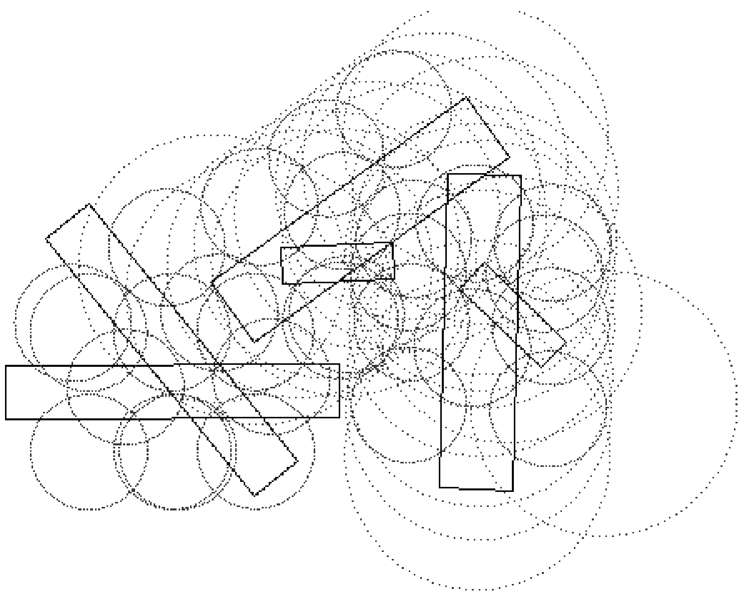


Figure 5.8: A visualization of what the robot sees and knows. The thick lines are the object positions including both their current position and their target position from the images seen in Fig. 5.5. The dotted circles are the collision spheres from the different grasps. The small circles origin from the gripper fingers, while the large circles arise from the gripper itself and the camera mounted on the end effector, see Fig. 5.9. These larger spheres are actually on a higher altitude and only cause collisions if a small object is being grasped next to a tall object.

any of its collision spheres intersect with the object's box. Since the robot knows the height of each object and each sphere has an altitude, this collision check is done in 3D. A grasp may have several collision spheres. For the parallel-jaw gripper we are using, there are one small sphere for each of the fingers, one large sphere on a higher altitude for the hand, and another large sphere to model the camera mounted on the hand. Fig. 5.9 shows a typical error that occurs if the camera is not modelled - there is a collision between the camera and one of the objects. Thus, the robot will have to plan grasps such that the camera does not collide. Due to the large camera and the limited workspace, the planning task is not trivial.

5.3.3 Finding Free Space

The initial and destination locations of objects are usually not enough for the planner to solve the task. The workspace is narrow and for many tasks, the robot needs free space to unload objects temporarily. These locations are found by searching in the workspace

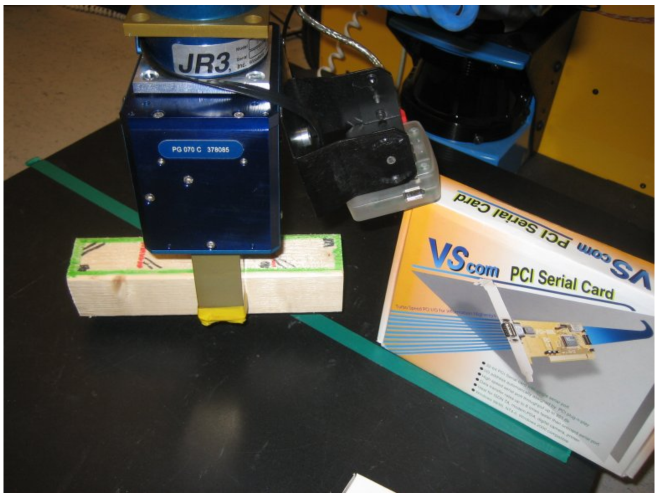


Figure 5.9: The choice of grasp type is very important. Here, the camera has not been modelled and thus the robot thinks it is fine to put down the wooden block next to the box. However, this causes a collision between the tall box and the camera.

visualized in Fig. 5.8 for locations which has the fewest number of collisions for all objects and grasps. This location is then treated as any other location and any possible collision that may still occur is provided to the planner. Since searching every possible location would be too time consuming, the search is limited to every 5 cm and every 60 degree rotation. This yields 210 possible locations, but of course this can easily be increased at the expense of increased search time.

5.3.4 Taking Profit from Human Advice

We have also modelled tasks in which the human has the possibility to instruct the robot that one of the objects (*tape*) has to be manipulated (*placed on the table*) before any other object. We note here that there is currently no verbal communication - instead the expected result of such a system was encoded in the planner. The instruction "The tape should be placed first" adds two constraints: the tape should be at its target location before the box, and also before the block. These are fed two the planner as predicates, e.g, *mustOccurBefore(tape, loc_tape_t, box, loc_box_t)*.

5.4 Automatic Generalization from Multiple Examples

If the teacher does not instruct the robot about the constraints of the task directly, the robot should still be able to detect these by observing the task performance several times. Beside this, in the current system the robot can calculate the average location of each object and does not need to place them at the exact locations from a particular demonstration. The ultimate goal is to have a robot autonomously moving around in the environment, observing humans performing their everyday tasks and so learn new or update models of the existing tasks. Then, without further instruction, the robot can ideally acquire the knowledge to perform the tasks itself.

With the work presented here, we aim to automatically identify the underlying constraints of the task. Generalizing from multiple examples requires more components than when simply imitating a human task. The necessary steps include:

Segmentation - The segmentation of the task into isolated operations or *primitive tasks* is a research issue that has been studied before, (Kuniyoshi *et al.*, 1994; Atkeson and Schaal, 1997; Matarić, 2000; Friedrich *et al.*, 1999). The task as a whole is unlikely to be observed again because of the minor variations that occur from demonstration to demonstration. We view the task as a composite of specific actions, *primitive tasks*, which can be easily recognized.

In this work, the task segmentation process is performed manually. Kang and Ikeuchi (1995) showed how automatic segmentation can be performed by using a change detection algorithm on the volume sweep rate, which is the product of the hand speed and the finger-tip polygon area.

State Generation - To enable generalization over multiple demonstrations, the subtasks are modelled as states, describing the impact of a certain action to the current world state, e.g., *"Knife moved 10 cm to the right of the plate"*. The *state generation* block takes all demonstrations into account and searches for similar subtasks which are represented by the same state. The similarity is measured in terms of effects on the world state.

Task Generalization - This process is used to identify which states must occur before others and possibly which states that are irrelevant for the task goal. From a single demonstration, the task is carried out in the exact same order unless some prior knowledge is available. From multiple demonstrations, the robot acquires more knowledge about the task and achieves the goal by assembling its own action sequence from a combination of all demonstrations.

5.4.1 Example Task

For easier understanding of the system implementation details, let us study a specific task we would like to teach a robot, *chop-a-cucumber*. The following objects are considered in the task: a cutting board, a cucumber and a knife. Given that the objects' poses are estimated, this task can be learned incrementally as shown in Table. 5.1. Here, object positions can be represented given either absolute world coordinates or relative to other objects already manipulated. The demonstrated tasks are segmented, and each subtask is quantized to a state. A demonstration is then represented as a state sequence. Another example task used later on in the chapter is *setting up a dinner table*. This task consists of placing plate, knife, fork, spoon, glass, food and napkin on a dinner table.

Table 5.1: Task *chop cucumber* as modelled in our system (z-axis anti-parallel to gravity).

Object	Relative Position	Relative Orientation	(x,y,z,θ,ϕ,ψ) Pose [cm, degrees]
Cutting board	None	None	(393, 123, 0, 0, 0, 0)
Cucumber	Cutting board	Cutting board	(10, 15, 1, 90, 0, 0)
Knife	Cucumber	Cucumber	(25, 0, 6, 90, 90, 0)
Knife	Cucumber	Cucumber	(25, 0, 0, 90, 90, 0)
Knife	Cucumber	Cucumber	(25, 0, 6, 90, 90, 0)
Knife	Cucumber	Cucumber	(24, 0, 6, 90, 90, 0)
Knife	Cucumber	Cucumber	(24, 0, 0, 90, 90, 0)
...	...	...	....

5.4.2 State Generation

In this section we provide more information of how continuous measurements are quantized from the operations. For the tasks considered in our work, the placement of certain objects can be defined relative to other objects (*Place glass to the left of the main plate*) but some objects are to be placed to a specific position defined in absolute coordinates, i.e. robot centered coordinate system or some world coordinate system. To decide if the position should be regarded absolute or relative, we compute the minimum variance with respect to already placed objects:

$$relobj_i = \underset{\forall j \; moved}{\operatorname{argmin}} \; |cov(\mathbf{x_i} - \mathbf{x_j})| \tag{5.1}$$

where $\mathbf{x_i}$ is the position of object i. If $relobj_i = i$, then the position should be regarded as absolute. The same procedure is done for the orientation, meaning that an object can have a relative position to one object and a relative orientation to another object. For some tasks, there may be several positions that are valid for a certain object. A difficult problem is how to automatically decide when a position should be regarded as a new state, and when it should be regarded as a variation of an existing state. We use K-means clustering to quantize the position and orientation for a specific object into a number of subgroups. This quantization method is good even though the amount of data is low which is in general the case in PbD systems. In detail, each position can be considered as a point in N-dimensional space, N being the number of DoFs for the object. If the same object is placed at approximately the same location in several demonstrations, the corresponding points will lie close to each other. K-means clustering automatically finds groups of points (see Section 2.4.2.1), which can then be labeled as the same state. The optimal number of subgroups is the one which yields the lowest maximum variance. However, the clusters are not allowed to lie closer than a certain threshold to each other, to prevent the scenario of a single cluster for each measurement. The improved algorithm becomes:

$$relobj_i = \underset{\forall j\ moved,\ c\ \in\ [1,N_{demo}]}{\operatorname{argmin}} \max_{k=1}^{c} |cov(\mathbf{x_i^k} - \mathbf{x_j^k})| \tag{5.2}$$

The best value is sought over all objects and cluster possibilities. Here, x_i^k denotes subset k when object i is clustered into c clusters. With this approach, we are able to identify multiple suitable positions and orientations for a single object, e.g., for a *set table* task, the spoon can be either above or to the right of the plate.

5.4.3 Task Generalization

After the demonstrations have been abstracted to state sequences, the robot can analyze all sequences to build a general task model. Fig. 5.10 illustrates how this is done.

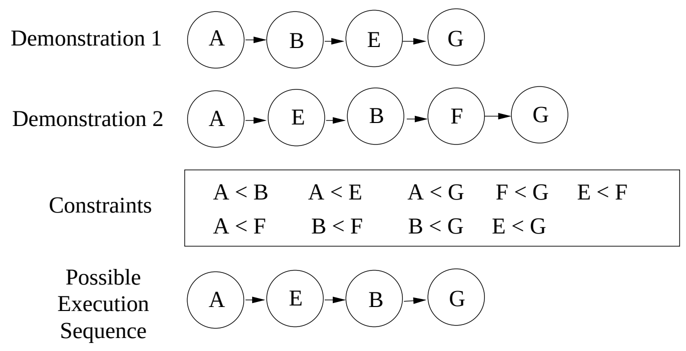


Figure 5.10: Top: Two demonstrations given to the robot. Center: Nine constraints are identified. Note that state B and E are not constrained. Bottom: One of the possible sequences to follow at execution time

In this example, there are two demonstrations. From these, nine *constraints* are identified. Initially, all actions are constrained to the order they were demonstrated. When two or more constraints contradict each other, they are removed. Thus, in the example above the constraints $B < E$ and $E < B$ have been removed. The robot is then free to reach any of the goal states demonstrated, as long as it does not violate any of the constraints. As more demonstrations are added, the list is modified. We then utilize our planner to calculate a sequence of actions to achieve the goal under the constraints. Note that as the sequence is calculated at run-time, the robot does not have to follow any of the human examples.

Another method for task generalization is presented in (Nicolescu and Mataric, 2003). The method is based on the longest common subsequence (LCS) of the state sequences, and the LCS of several demonstrations constitute the generalized task model, in which the other actions appear as alternative paths. However, this approach is not suitable for manipulation tasks. Many pick-and-place tasks can be performed in arbitrary order, so the

LCS for those tasks may be as short as a single state. Instead, we propose to build up a list of constraints that describes which states must occur before others.

Among the constraints generated in the example above, some are unnecessary, e.g., $A < G$, when the constraints $A < B$ and $B < G$ are present. These types of constraints can be removed, but they actually serve a purpose: they make the planning go faster. The planner does not have to try $G-A-B$, which is a dead end.

5.5 Experimental Evaluation

Throughout the experiments presented here, we have operated in three dimensions. Each object is assumed to be placed on a table, and only the x, z and θ pose parameters are used. The robot also knows the height of each object. We used three objects, which is enough to complicate the situation for the planner. The robot has a very limited workspace, about 40x20 cm as indicated in Fig. 5.6, so the choice of grasp type and move order sequence is crucial for the task success.

5.5.1 Planning Example

Here we give an example of how the planner operates. We ran the planner on the two test images seen in Fig. 5.5. The generated list of predicates consists of three parts. First, the collisions are listed:

collision(loc_box_s, loc_box_t, box_left)
collision(loc_box_s, loc_box_t, box_center)
collision(loc_box_s, loc_box_t, box_right)
collision(loc_block_s, loc_tape_s, block_center)
...

Then, the reachability of each grasp is listed:
graspable(loc_box_s, box_left)
graspable(loc_box_s, box_center)
graspable(loc_box_s, box_right)
graspable(loc_block_s, block_center)
...

Finally, the location of each object is listed:
objAtLoc(box, loc_box_s)
objAtLoc(block, loc_block_s)
objAtLoc(tape, loc_tape_s)

and of course *handEmpty*. The goal state is listed separately as
objAtLoc(box, loc_box_t)
objAtLoc(block, loc_block_t)

objAtLoc(tape, loc_tape_t)

The planned solution is provided below.

pickUp(box, loc_box_s, box_center)
putDown(box, loc_box_t, box_center)
pickUp(tape, loc_tape_s, tape_center_180)
putDown(tape, free1, tape_center_180)
pickUp(block, loc_block_s, block_center)
putDown(block, loc_block_t, block_center)
pickUp(tape, free1, tape_center)
putDown(tape, loc_tape_t, tape_center)

Thus, the system identifies a free location and uses it to store the tape while the other objects are moved into place. The planner chooses the grasps and a move sequence such that the task can be completed without collisions.

5.5.2 Imitation Learning

To evaluate the imitation learning and the overall performance of the system, we placed the three objects in six different configurations according to Fig. 5.11. For each configuration, the robot was asked to reconfigure the objects to one of the other configurations. In total, the robot had to plan and execute 30 tasks using its 6-DoF arm and parallel-jaw gripper. In the imitation setting, a “demonstration” is simply an image of the target configuration, however the starting configuration varied slightly for each experiment since it is not possible to place the objects exactly according to the image. Table 5.2 shows the results.

The table should be interpreted as follows.

- S - success, all objects were successfully moved to their new position.
- F1 - failure type 1. One of the objects in the source or target configurations was not found.
- F2 - failure type 2. The grasping of an object was not as expected, which caused collision when it was put down. For example, the wooden block is heavy and may slip, or the box may tilt when being grasped.
- F3 - failure type 3. Imprecise pose estimation caused the gripper to collide with the object when grasping.

In this experiment, 11 of 30 tasks were successfully completed. Out of the 19 failures, 16 occurred due to pose estimation errors, that is, due to the imperfect visual input. Since we only use one image for pose estimation, we believe that these errors can be avoided by using several images for pose estimation. For example, 10 of the unsuccessful task executions are related to the image shown in the center of the bottom row in Fig. 5.11.

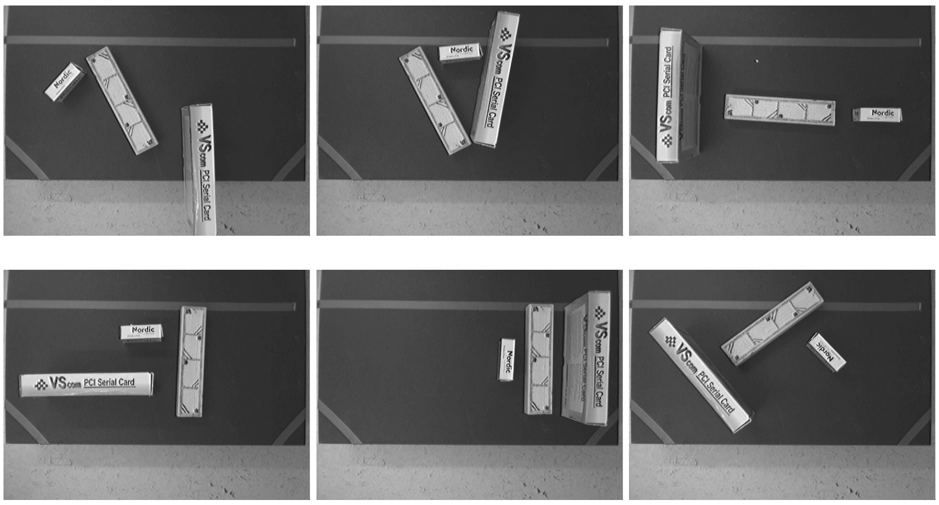


Figure 5.11: The six different object configurations used in the experiments, seen from the robot's point of view.

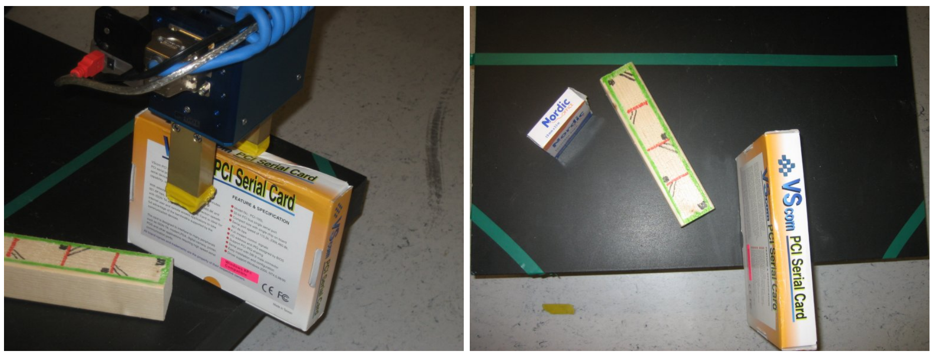


Figure 5.12: Left: The robot gripper as it about to put down the box and finish the task. Right: The final configuration of the objects, arranged by the robot. When compared with the demonstrated task, Fig. 5.11 (top-left), the configurations seem identical.

Apart from pose estimation errors, type F2 failures are not easily detectable and hard to predict. One solution is that the robot visually verifies the final configuration of the manipulated object. Then, if the result is not as expected, the robot re-plans. This is currently not implemented in the system.

Table 5.2: The outcome of each task in the experiments.

Target Conf. / Source Conf.	1	2	3	4	5	6
1	-	F2	F3	F3	F1	S
2	S	-	S	S	F1	F1
3	F3	F3	-	S	F1	F3
4	S	F2	S	-	F1	S
5	F1	F1	F1	F1	-	F1
6	S	F2	S	S	F1	-

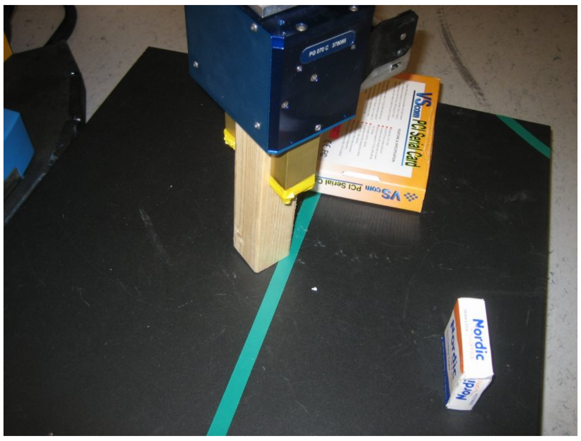


Figure 5.13: An unexpected error which the system can not detect or recover from. The wooden block is heavy and sometimes slips from the gripper when grasped on the side. The gripper still holds on to the block but due to its weight it rotates, and when being put down it causes a collision.

5.5.3 Learning from Human Advice

This experiment is similar to the one presented in Section 5.5.2. This time, the human instructs the robot that the tape is to be in place before the other objects, which adds two constraints to the planner. This results in a bit longer plan:

pickUp(box, loc_box_s, box_center)
putDown(box, free1, box_center)
pickUp(block, loc_block_s, block_left)
putDown(block, free2, block_left)

pickUp(tape, loc_tape_s, tape_center_180)
putDown(tape, loc_tape_t, tape_center_180)
pickUp(block, free2, block_left)
putDown(block, loc_block_t, block_left)
pickUp(box, free1, box_center)
putDown(box, loc_box_t, box_center)

As seen, the robot fulfills the constraint of placing the tape at the target location before the other objects are placed at their target locations.

5.5.4 Generalizing from Multiple Examples

We have also performed experiments to evaluate our approach of generalizing from multiple examples. The first experiment is performed in a virtual environment and evaluates the state generation module with several objects. The second experiment shows how the task generalization module operates by automatically identifying a hidden constraint from several examples in the real world.

5.5.4.1 Generalizing in a Virtual Environment

In this experiment, the *set table task* described in Section 5.4.1 is considered. The task was demonstrated by the user three times in a virtual environment where each object was only allowed to be moved, not rotated. The state of each object can then be represented using only its position which makes this experiment easy to analyze.

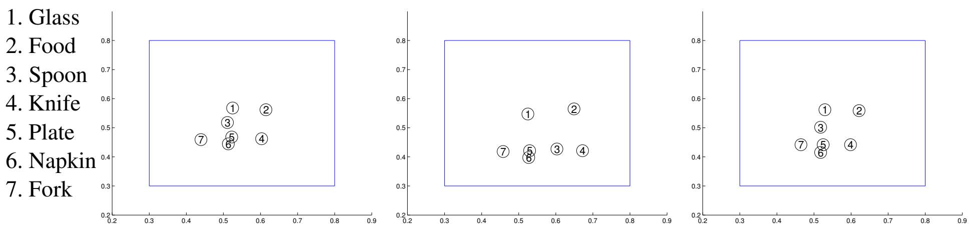


Figure 5.14: Example demonstrations in the virtual environment.

Fig. 5.14 shows the result of each demonstration. Demonstration 1 and 3 were similar but the objects were not moved in the same order. Demonstration 2 was different because of the spoon being put to the right of the plate, instead of behind it. Table 5.3 shows the states generated by the system. As expected, the knife, fork and napkin are specified relative to the plate. Because the glass position varied too much relative to the plate, its position is specified in absolute coordinates. The system correctly identifies the two possible placements of the spoon (state D and H). State J arises since the fork actually has lower variance towards the glass compared to the plate. In the first demonstration the fork was put down before the glass and positions can only be specified towards already placed objects. Table 5.4 shows the generated state sequences for the demonstrations. From these, a total of 32 constraints were identified. In this example, the constraints make sure that when an object is to be placed, its 'relative' object is already in desired position.

Table 5.3: Experiment in virtual environment: The states generated from the demonstrations.

State	Object	Relative Position	(x,z)
A	Plate		(0.52, 0.44)
B	Knife	Plate	(0.1, 0)
C	Fork	Plate	(-0.07, 0)
D	Spoon	Knife	(-0.08, 0.04)
E	Glass		(0.52, 0.56)
F	Food		(0.62, 0.56)
G	Napkin	Plate	(-0.01, -0.02)
H	Spoon	Plate	(0.02, 0.04)
I	Knife	Spoon	(0.08, -0.04)
J	Fork	Glass	(-0.07, -0.12)

Table 5.4: The state sequences found in the demonstrations (virtual environment).

Demonstration 1	A-B-C-D-E-F-G
Demonstration 2	A-E-H-I-J-F-G
Demonstration 3	A-C-B-F-E-D-G

5.5.4.2 Generalizing From Real Examples

The method has also been evaluated in a real scenario where the robot observed the human performing the task three times. The start configuration was different each time, which led to three different observations, shown in Table 5.5. The task was to organize the objects according to test image 6, and the starting configurations were according to test image 1, 2 and 3 from Fig. 5.11. The user had an underlying constraint when performing the task, that the tape must be placed at the target location before the other objects reach their locations.

From the observations, five states were automatically generated, as shown in Table 5.6. It is interesting to see that the system has correctly identified that the block should be placed relative to the tape, and that the box should be placed relative to the block.

Table 5.5: The three observations perceived by the system. (real environment)

Observation 1	move box to (214, -40, -10)
	move tape to (12, -76, 57)
	move block to (-120,-39,-35)
	move box to (-174, 49, -121)
Observation 2	move box to (220, -60, 4)
	move tape to (13, -55, 52)
	move block to (-109, -25, -36)
	move box to (-106, 64, -122)
Observation 3	move block to (268, -82, -138)
	move tape to (23, -59, 52)
	move block to (-94, -16, -41)
	move box to (-93, 72, -126)

Table 5.6: Experiment in real environment: The states generated from the observations.

State	Object	Rel. Position	Rel. Orientation	(x,z,θ)
A	box			(217, -50, -3)
B	block			(268, -82, -138)
C	tape			(16, -63, 54)
D	block	tape	tape	(-124, 37, -91)
E	box	block	block	(-17, 88, -86)

The observations are then remapped to the identified states. This results in state sequences, as shown in Table 5.7.

Table 5.7: The state sequences found in the observations.

Observation 1	A-C-D-E
Observation 2	A-C-D-E
Observation 3	B-C-D-E

From these, three constraints, $C < D, C < E$ and $D < E$, and one common goal, C, D, E

are identified. The goal is the final position of each object. Considering that this is a pick-and-place task, only the final positions are important. For other types of tasks the robot may have to perform either A or B as well. Note that both underlying constraints were identified but also an additional constraint $D < E$. Although not intentionally demonstrated, that constraint is necessary in order to be able to align the box next to the block. The planned solution is the same as in Section 5.5.3, although the locations are slightly different since they are calculated from several observations.

5.6 Discussion

In this chapter, we have presented a task learning system where a robot learning by demonstration scenario is integrated with a task level planning system. Three learning techniques have been considered: task learning from imitation, learning from human advice and learning from multiple observations where a task is represented by its goal configuration and task constraints. A task planner for manipulation tasks and reaching the goal state given different initial world states has been demonstrated. We have also addressed the issue of task constraints. If there are some underlying constraints that must be fulfilled the knowledge of just the final goal is not sufficient for task execution. We have proposed two techniques for constraint identification. In the first case, the teacher can instruct the system and, in the second case, the constraints are identified by the robot itself through the merging of multiple observations. The constraints are then considered in the planning phase, allowing the task to be executed without violating any of them.

In large complex tasks, there may be intermediate goals that must be met; the success of the task does not just rely on the final goal. Our system can be used for such complex tasks, however we then require the teacher to specify the intermediate goals. The planner will then treat the task as a sequence of simpler subtasks.

The experimental evaluation has been performed both in a virtual environment and with a robot manipulator. It has been demonstrated that the system is able to perform tasks with real objects. We believe that the proposed framework is easily extendable to tasks involving more complex objects. The current system requires that all objects are visible at the planning stage.

Chapter 6

A Service Robot Application

The problem studied in this chapter is the application of the Receptive Field Cooccurrence Histogram described in Section 3.4 to a mobile robot that autonomously navigates in a domestic environment, builds a map as it moves along and localizes its position in it. In addition, the robot detects predefined objects, estimates their position in the environment and integrates this with the localization module to automatically put the objects in the generated map. Thus, we demonstrate one of the possible strategies for the integration of spatial and semantic knowledge in a service robot scenario where a simultaneous localization and mapping (SLAM) and object detection/recognition system work in synergy to provide a richer representation of the environment than it would be possible with either of the methods alone.

The added mobility enables the robot to learn a whole new set of tasks. Before task learning, the robot can acquire room and place knowledge through communication with the human teacher. At the same time, the robot can learn to recognize objects and store their latest locations in the map. The spatial and semantic knowledge is a prerequisite for learning higher level mobile tasks. An example of such a task may be the delivery of mail in an office. Here, the robot could learn the task goals and plan a path to achieve those goals, in a similar way to the pick-and-place task learned in the last chapter. Here, fetch-and-carry is the corresponding operation to a pick-and-place operation. To learn such a task, the robot would have to link verbal or typed instructions to its spatial-semantic knowledge, as it would be very difficult to correctly interpret object interactions from vision alone in such uncontrolled learning scenarios.

To enable SLAM, we have integrated our work with the work presented in (Jensfelt *et al.*, 2006). Thus, the navigation examples presented here are not a contribution of this chapter.

6.1 Motivation and Related Work

The importance of robotic appliances both in economical and sociological perspective regarding the use of robotics in domestic environments as help to elderly and disabled has been well recognized. The AAAI Mobile Robot Challenge has demonstrated that the development of an interactive social robot represents a clear research challenge for the future. Such a robot should be able to easily navigate in dynamic and crowded environments, detect as well as avoid obstacles, have a dialog with a user and manipulate objects. It has been widely recognized that, for such a system, different processes have to work in synergy: high-level cognitive processes for abstract reasoning and planning, low-level sensory-motor processes for data extraction and action execution, and mid-level processes mediating these two levels.

A successful coordination between these levels requires a well-defined representation that facilitates anchoring of different processes. One of the proposed modeling approaches has been the use of *cognitive maps*, (Kuipers, 1983). The cognitive map is the body of knowledge a human or a robot has about the environment. In (Kuipers, 1983), it is argued that topological, semantic and geometrical aspects are important for representation of spatial knowledge. This approach is closely related to Human-Augmented mapping (HAM) where a human and a robot interact so to establish a correspondence between the human spatial representation of the environment and the robot's autonomously learned one, (Kruijff *et al.*, 2006). Both of the above have strongly influenced our current work where the integration of object recognition and map building represents a basis for longterm reasoning and planning in an autonomous robot system.

The ability to automatically detect rooms is important as it enables the robot to learn labels as it is being showed a new environment. Here, the scenario is that the robot automatically follows the human while learning labels through verbal communication. There are several levels of labels, the robot can learn that inside the kitchen there is a place called the kitchen table. Both during the mapping phase and during robot task execution, object detection can be used to augment the map of the environment with objects' locations. We see several scenarios here: while the robot is building the map it will add information to the map about the location of objects. Later, the robot will be able to assist the user when s/he wants to know where a certain object X is. As object detection might be time consuming, another scenario is that the robot builds a map of the environment first and then when no tasks are scheduled for execution, it moves around in the environment and searches for objects.

The same skill can also be used when the user instructs the robot to go to a certain area to fetch object X. If the robot has seen the object before and it already has it in the map, the searching process is simplified to re-detection. By augmenting the map with the location of objects we also foresee that we will be able to achieve place recognition. This provides valuable information to the localization system as well as it greatly reduces the problem with symmetries in a simple geometric map. This would be an alternative approach to the visual place recognition presented in (Pronobis *et al.*, 2006) and the laser based system in (Martínez Mozos *et al.*, 2005). Furthermore, along the way by building up statistics about what type of objects typically can be found in, for example, a kitchen, the robot might not only be able to recognize a certain kitchen but also potentially generalize to recognize a room it has never seen before as probably being a kitchen.

Although there exists a large body of work related to mobile robots, there are still no fully operational systems that can operate robustly and long-term in everyday environments. The current trend in development of service robots is reductionistic in the sense that the overall problem is commonly divided into manageable sub-problems. During the last few years, there have been a few examples of systems where the robot can acquire and facilitate semantic information, (Theobalt *et al.*, 2002; Galindo *et al.*, 2005). Different to our approach, the work presented in (Theobalt *et al.*, 2002) is mostly concentrated on linguistic interaction with a human and the robot is not using its sensors to retrieve semantic information. The anchoring approach, presented in (Galindo *et al.*, 2005), deals mostly with the problem of integrating semantic and spatial levels where a special type of representation is used to achieve this.

6.1.1 Active Vision

The vision system design in this work is based on the *active vision* paradigm, (Ballard, 1991) where, instead of passively observing the world, viewing conditions are actively changed so that the best results are obtained given the task at hand. As shown in Fig. 6.1, if the object is too far away from the camera (left), no adequate local information can be extracted. Therefore, the main idea pursued in this work is to use a global appearance-based method to generate a number of hypotheses of the whereabouts of the object. The robot then investigates each of these hypotheses by moving closer to them, or as in our case, by zooming with a pan-tilt-zoom camera. Once the object appears large enough, it can be recognized with the local feature-based method. One of the contributions in this chapter is that we make use of both approaches in a combined framework that let the methods complement each other. If the robot recognizes the object from two different locations, it can use geometric triangulation to calculate the approximate world position of the object and store it in its map.

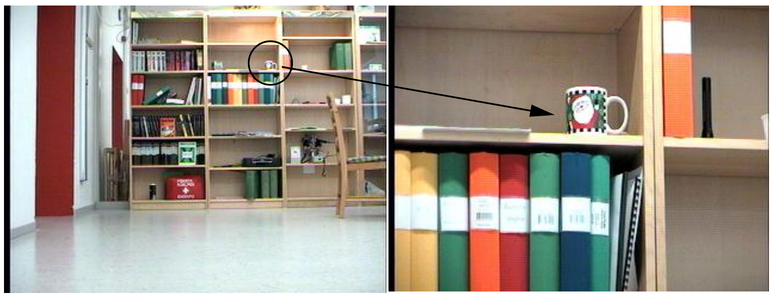


Figure 6.1: Left: The robot cannot recognize the mug located in the bookshelf. Right: Minimum size of the mug required for robust recognition.

6.2 Building a Map of the Environment

For automatic acquirement of semantic structure of the environment, automatic map building and its integration with object/place identification is a basic requirement. For increased flexibility, the robot should both be able to build a map and use it for localization. Many of the methods for SLAM, including the one used in this chapter, have their roots in the work by Smith *et al.* (1988). Most of the work in SLAM focuses on creating a map from sensor data but not on how this data is created or how to use the map afterwards. In this chapter, we want the robot to use the map to carry out different tasks. Some of these tasks may require that the user communicates with the robot using common labels from the map such as *"that room"* or *"that window"*. A natural way to achieve this is to let the robot follow the user around the environment in the initial stage when the map is being built. This allows the user to put labels on things and places of interest: certain locations, areas or rooms. This is convenient for example when instructing the robot later on to go to a certain area to fetch something or when asking the robot for information about where something might be.

The SLAM algorithm uses a laser sensor and details can be found in (Folkesson *et al.*, 2005). A feature based map (e.g., 2D line map as in our case) is rather sparse and does not contain enough information for the robot to know how to move from one place to another. Only structures that are modelled as features will be placed in the map and there is thus no explicit information about where there is free space such as in an occupancy grid. In this work we use a technique inspired by (Newman *et al.*, 2002) and build a navigation graph while the robot moves around. When the robot has moved a certain distance, a node is placed in the graph at the current position of the robot. Whenever the robot moves between two nodes, these are connected in the graph. The nodes represent the free space and the edges between them encode paths that the robot can use to move from one place to another. The nodes in the navigation graph can also be used as references for certain important locations such as, for example, a recharging station. Fig. 6.5 shows an example of a navigation graph as connected stars. For a more detailed description of the navigation graph and how it can be used for space partitioning space, see (Kruijff *et al.*, 2006).

6.3 Active Object Recognition

Despite the large body of work on vision based object recognition, few have investigated strategies for object recognition when distance to the object (scale) changes significantly. Similarly, there are very few object recognition systems that have been evaluated in a mobile robot setting. In (Gopalakrishnan and Sekmen, 2005), a robot behavior similar to ours is presented, but with somewhat limited vision algorithms. A mobile, self-localizing robot wanders in an office environment and can learn and recognize objects encountered. However, the recognition algorithm cannot cope with a cluttered environment and it works only for a very few objects since the neural-network-based vision algorithm only uses object shape information as input.

Impressive recognition results have been achieved with methods based on Scale Invariant Features (SIFT) (Lowe, 1999) and alike. The most significant drawback of these methods is that the reliable features can only be found when the object occupies a significant part of the image. It is very hard to recognize objects that are far away from the camera. On the other hand, the main drawback with RFCHs turned out to be the false positive rate. They are excellent for providing the most probable object location in a cluttered environment, but when it comes to determining whether the object is actually in the scene, the performance is mediocre. We have solved this problem by both making use of a pan-tilt-zoom camera and a global method prior to a feature-based one to generate hypotheses. By zooming in on a number of a probable object locations provided by the hypotheses generation step, objects far away from the camera can be recognized. We propose to use RFCHs for generating hypotheses of object locations, and then use a SIFT-based method for object recognition once the object is zoomed-in.

6.3.1 Active Object Learning from Demonstration

For the robot to put a new object into the database, we have adopted a very natural approach with an ordinary end-user in mind: the object is shown to the robot by placing it in front of the camera. During this "teaching" step, features are extracted from the image and it is crucial that only features from the object are extracted and thus learned by the classifier. If the background is visible in the image, that information will be learned as well and will therefore increase the number of false positives in the online recognition stage. The common way of segmenting the object is to manually process the image in an editor and crop the object from the background. However, this is a tedious step which may be difficult for a regular user to perform. In our framework, training images are are generated from human demonstrations in two steps. First, the robot captures an image of the scene without the object being present in it. Then the operator places an object in front of the camera and the object is separated using image differencing. Thus, the user is relieved of the process of manually extracting the object. However, simple image differencing is prone to noise and the result is not a perfect image of the object as it contains many holes in the object and also some of the background. To cope with this problem, a number of morphological operations is performed to achieve better segmentation (errode - dilate - errode, (Gonzalez and Woods, 1992)). These operations are performed using information from the original image, i.e., a growing effect (covering holes) will not add black pixels, but pixels from the original image. The result of this step can be seen in Fig. 6.2. The final image may still not have a perfect segmentation of the whole object but this is not a problem for any of the vision algorithms we apply as long as most of the object is visible.

Another problem with image differencing is the choice of a threshold θ that determines if a pixel is part of the background or not. If θ is set too high, too much of the background will remain. If θ is set too low, significant parts of the object may be omitted. In our work, we use an automatic adjustment of θ based on the result of the differencing performance. If image differencing was successful, the remaining pixels should be concentrated to a single area where the object has been placed. If the differencing has failed, the pixels are mostly scattered around the entire image. Thus, the success is measured in terms of detection

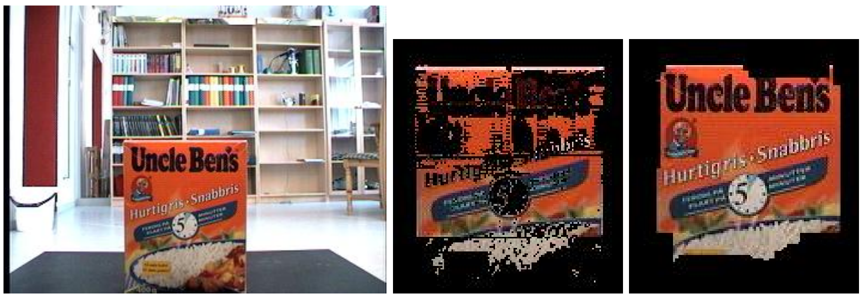


Figure 6.2: Left: The original image. Center: The result after simple image differencing (only a part of the image is shown). Right: The result after morphological operations.

variance. In addition, a penalty is added that is linearly proportional to the number of pixels remaining, to cover the case of very few remaining pixels that have a low variance but are not sufficient for the object representation. The algorithm tests every θ from 1 to 150, to find the optimal setting with the lowest score.

6.3.2 Hypotheses Generation

Object hypotheses are generated by scanning the image with a small search window, resulting in a vote matrix as explained in Section 3.3.4. If a vote is higher than a certain object-dependent threshold, the corresponding location for that vote is considered a hypothesis.

The threshold value provides a trade-off between search time and detection probability. If the threshold is low, many hypotheses are generated and evaluating them all is time consuming. On the other hand, if the threshold is high, there is a risk that the object is missed. There are many factors influencing the match value of the searched object, e.g specular reflections, illumination variations, occlusion. However, an extensive experimental evaluation has shown that the method is not very sensitive to the value of the threshold. In this work we found that $\gamma = 0.2$ was suitable for relatively small objects (approximately 8-10 cm in height), while the larger objects could use $\gamma = 0.25$ for a faster search.

6.3.3 Hypotheses Evaluation Strategy

Given our pan-tilt-zoom camera and a set of hypothesized object locations, the task is to efficiently determine and zoom on the generated hypotheses. To speed up the process, we decided to first use an intermediate level of zoom and then use the appearance-based object detector for final verification, before applying the somewhat slower feature-based method. Finding the best image locations to zoom on is not a trivial task. We quantize the view space into the same size as the vote matrix. For each vote cell, we calculate which hypotheses would still be visible if one would zoom on that location using zoom factor z. Then, the problem is to find the smallest set of zoom locations that cover all hypotheses. Here, a simple greedy approach is followed: Select the location that covers most hypotheses, then remove these from the list and calculate the zoom locations once again. Continue until all hypotheses are covered. See Fig. 6.3 for an example. This

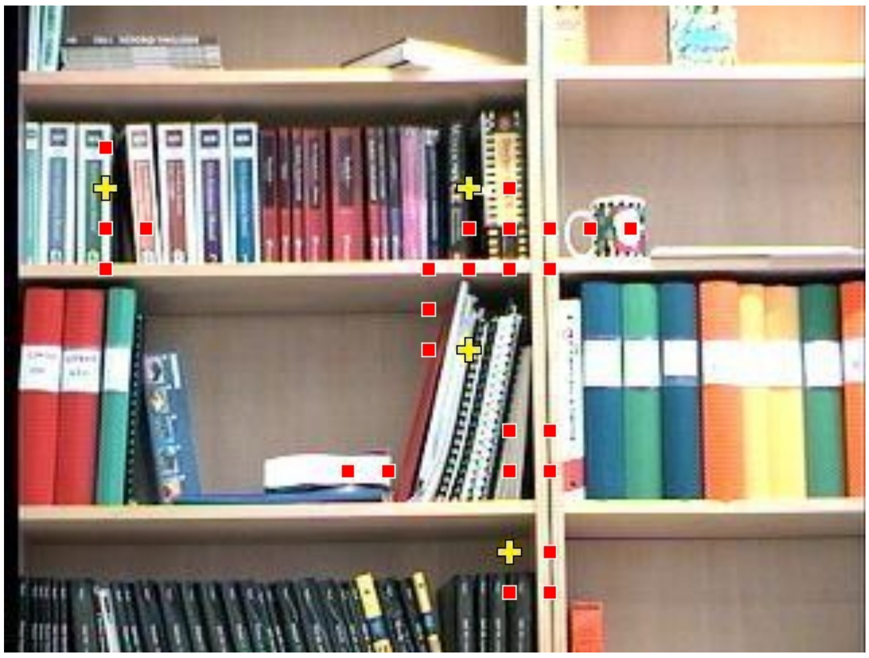


Figure 6.3: An example of the greedy search strategy used while searching for the mug at the intermediate zoom level. Squares represent possible object locations, and crosses are the calculated zoom locations that cover all the hypotheses. There are 4 zoom locations and 30 hypotheses in this example.

approach has proven to work well and is much more efficient compared to evaluating all of the hypotheses. The method is used both for high and the intermediate zoom levels. The zoom factor z decides how much to zoom in so to reach the next zoom level. A large z makes objects larger in the image and thus gives the detector more information but it in turn means that fewer hypotheses can be evaluated simultaneously. To account for this, we set z based on the distance measured by the laser scanner in the direction of the object. If the distance is small, the far zoom level is skipped, and the algorithm starts at the intermediate level. If the distance is very small, about 1 m or less, the intermediate level is also skipped. Experimental evaluation will show that the object recognition method works well even if the object appears larger in the image compared to training images.

Once a hypothesis is zoomed in, we again use RFCH for matching. If the match value exceeds the threshold, we perform SIFT-matching to verify the hypothesis. The more SIFT-matches found in an image, the more likely it is that the image contains the object.

If the number of matches exceeds an object-dependent threshold, the object is considered recognized. Some objects have more features than others and are thus easier to recognize. To minimize the number of false positives, the threshold depends on the number of features found during training. If multiple objects are being searched for, the hypotheses for each object may be combined at each zoom level. This way, the number of zoom-in steps can be reduced, compared to searching for the objects in sequence. For each zoom-in operation, only those objects that generated the visible hypotheses are considered.

6.4 Integrating SLAM and Object Recognition

In the current scenario, we focus on the integration of SLAM and object recognition modules. We note here that the main strength of this work is the ability of the system to provide richer maps of the environment than it would be possible to achieve with any of the techniques alone. Here, the robot follows the user through a new environment so that the user can show the robot around. The robot is considered to be our *guest* that is getting a tour of the environment. The user can attach labels to areas/room, i.e. instruct the robot that *this* is the living room, *this* is the kitchen, (Kruijff *et al.*, 2006). These labels can then be associated with a part of the navigation graph. As the object recognition is moderately fast, we let the robot add objects to the map after the user has shown the extent of the environment. This is then carried out fully autonomously. Including objects into the map also serves to address another important problem in SLAM, namely, loop closing. In (Newman and Ho, 2005), a laser range scanner is used to build a map and vision-based Maximally Stable Extremal Regions (MSER) are used to detect loop closing situations. The objects in our map could be used in a similar way.

Some objects can be detected from more than one position. This allows for triangulation to estimate not only the bearing to the object but also the approximate position. Even though an object has only been detected once, the map contains information from where each object has been detected and in what direction which allows for re-detection. Objects are stored on an area by area basis. We should stress that we do not propose to try to estimate the exact position of an object here. When the time comes to interact with the object, for example to pick it up, it has to be re-detected anyway to confirm that it is still there and then use visual servoing techniques to pick it up using the same technique that has been proposed in (Petersson *et al.*, 2002).

6.5 Experimental Evaluation

The experimental platform used in this work is the same as in the last chapter, see Section 5.2.1. In this section, we first evaluate how long it takes for the robot to locate an object in a room, and also, how often the robot fails to find the object. Then, we give an example of how a search in multiple rooms is performed. Finally, a “fetch object”-command is given to the robot, and we provide an example execution of that command.

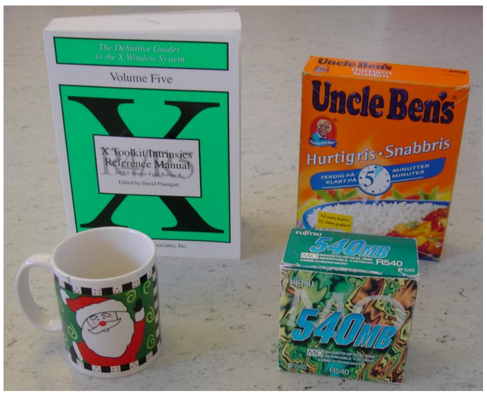


Figure 6.4: Some of the objects from the database.

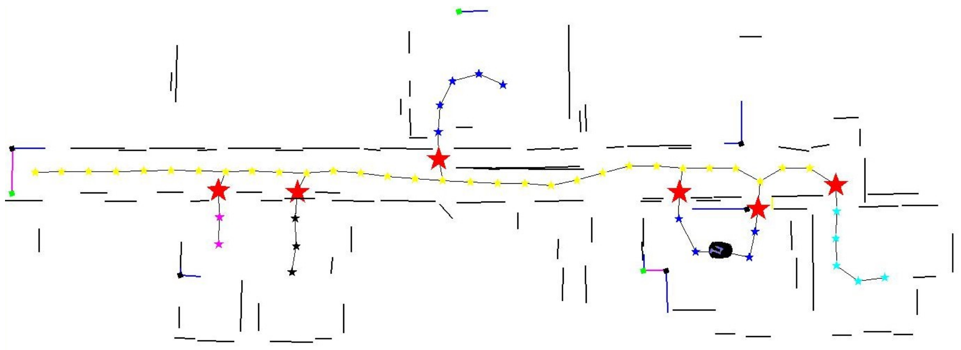


Figure 6.5: A partial map of the 7th floor at CAS/CVAP. The stars are nodes in the navigation graph. The large stars denote door/gateway nodes that partition the graph into different rooms/areas.

6.5.1 Evaluating the Search Effectiveness

We started by testing the effectiveness of our system in an office environment. Here, the robot was presented four objects (see Fig. 6.4) which were then placed in a room in six different configurations. In the training stage, the robot was given two views of each object: one close-up view for SIFT-training and one at a lower scale for RFCH-training. We note here that still a single image is used for the object, just at different scales.

The robot performed a search for the objects at each node in the navigation graph. This search was done with two different robot rotations, separated by 180°. Four different pan angles for the camera were used to cover the field of view for each of the two robot

Table 6.1: Object Recognition Results.

Object	ATD (min)	ATST (min)	Detect
Rice	1:10	3:52	6/6
Book	0:40	2:55	6/6
Mug	3:52	11:22	5/6
Zip-disks	3:28	7:01	4/6

orientations. In this specific experiment, the navigation graph consisted of four nodes in a single room (see Fig. 6.6). We note here that this strategy is not optimal and that there is room for improvement by, for example, considering some of the work on view planning, (Allen *et al.*, 1998).

For each object, we measured the average time for detection (ADT), average total search time (ATST) and the number of detections. Not all object locations were visible from all node positions so we counted the number of times the robot missed the object completely, i.e. did not see it from any of the node positions. As seen in Table 6.1, this only happened three times. The rice package and book were the easiest to detect, which can be seen from the average time for detection. This is not because of their appearance but rather due to the size: the rice package and book are both large, so their features are easier to detect when the object is far away. To spot the zip-packet or the mug, the robot usually had to be less than 3 m away from the object. We found that setting the SIFT-threshold to $1/20$ of the number of features found during training, a high detection rate and no false positives were achieved. The main reason for the required recognition time and failure is that the camera sometimes needed several seconds to focus after moving the camera or the robot, with the result that the images were blurry, causing the robot to miss the objects. In Fig. 6.6, the map of the room is shown with one of the object configurations with all four objects detected. Since all objects were detected from several nodes, their position estimates can be improved using triangulation.

To visualize the performance of the hypothesis generation step we have generated a few test images in which we varied both the initial distance to the object and object's rotation relative to the camera. In this scenario, the robot is searching for the rice package. The odd rows in Fig. 6.7 show the example images, while even rows show the vote matrices used for generate the hypotheses for the attention process. Strong hypotheses are shown with darker colors. We see that the rotation of the object around the camera axis does not effect the outcome of the hypotheses generation significantly. However, the rotation around the vertical axis does effect the hypothesis generation (evident in Fig. 6.7(b) and (f)). The reason for this is that the object is viewed from the side and the stored representation of the object uses the frontal image. However, the object is still one of the strongest hypothesis generated even if we, on purpose, placed it in an environment where there are other items with similar textural properties presents (orange folders). It can also be seen in Fig. 6.7(j) and (n) that once the hypothesis is examined more closely, the votes are stronger and facilitate correct detection. One of the limitations of the current method is thus not so much in the hypotheses generation but in hypotheses verification since SIFT points are

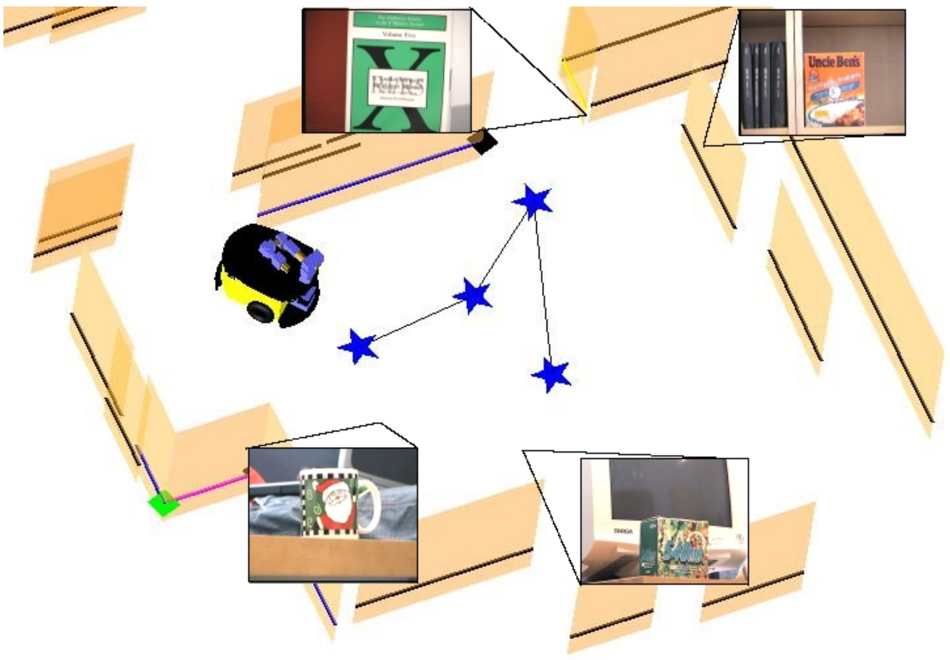


Figure 6.6: The results of a robot detecting four objects in a living room and estimating their positions.

invariant to rotations of up to 30-35°, (Lowe, 1999) which means that even if the correct hypothesis is generated the system may fail to detect the object when the SIFT based hypothesis generation process fails. This can be addressed by using more than one view of the objects.

6.5.2 Searching for Objects in Several Rooms

In this experiment, the search for objects is not limited to a single room. However, we skip door nodes and nodes directly adjacent to doors as stopping in these places blocks the way for people moving around. We used only two objects in this experiments, a soda can and the rice package. Fig. 6.8 shows the situation after the robot has visited two of the rooms. One instance of each of the objects were placed in these two rooms. The lines extending from close to the graph nodes starts in the camera position at the time of detection and is directed toward the observation of the object. As can be seen the objects are often spotted from more than one location. A rice package showed to the far right in the map was placed on a table. It has been detected three times and it can be seen that a triangulation would place the object closer to the camera than the laser which is only able to detect the distance to the wall behind the table. In this figure, it can also be seen that the robot has correctly

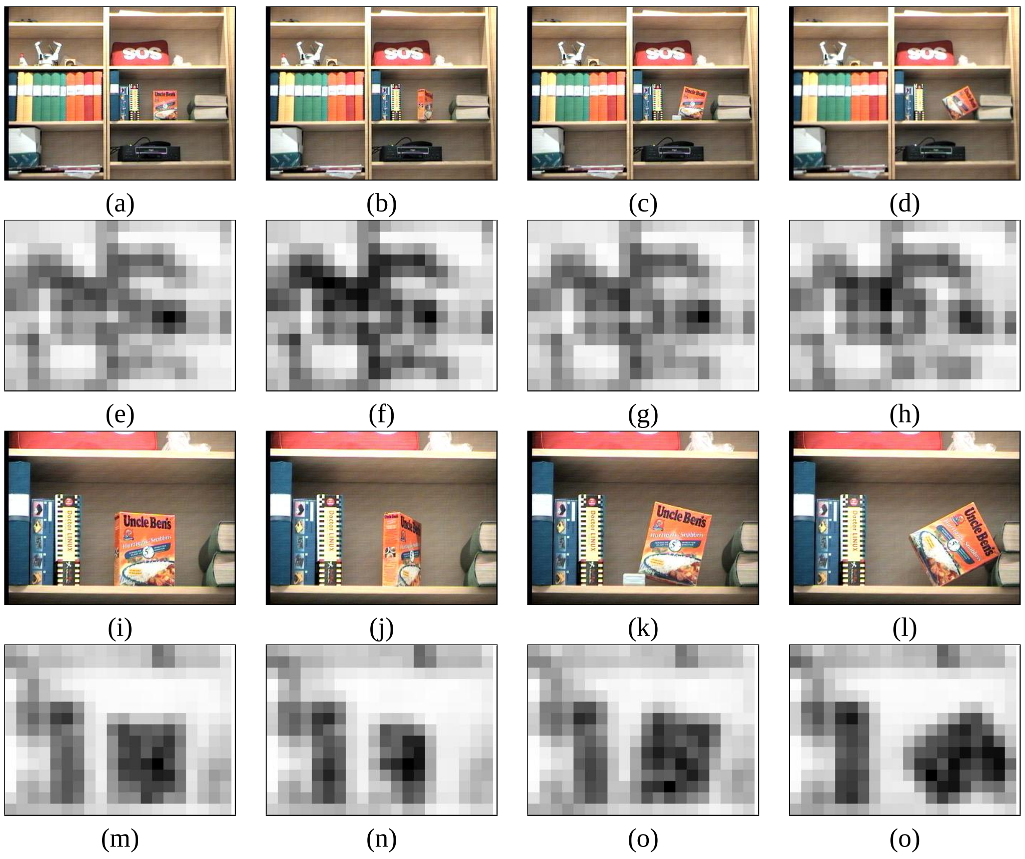


Figure 6.7: A few example images that demonstrate the outcome of the hypotheses generation process for cases when the rotation of the object relative to the camera changes. It is evident that the rotation around the vertical axis affects the results (figure (b) and (f)) for cases when the object is far away from the camera. A more detailed discussion is provided in the text.

detected the three doors in this part of the environment (marked as large stars), two of them leading to the same room from the corridor and thus correctly partitioned the space into rooms. This demonstrates the possibility of instructing the robot to fetch object X in room Y.

6.5.3 Fetching Objects

In section we demonstrate how the robot uses the acquired world knowledge to perform tasks. After the map is built and the robot has performed a multi-room search, it is in-

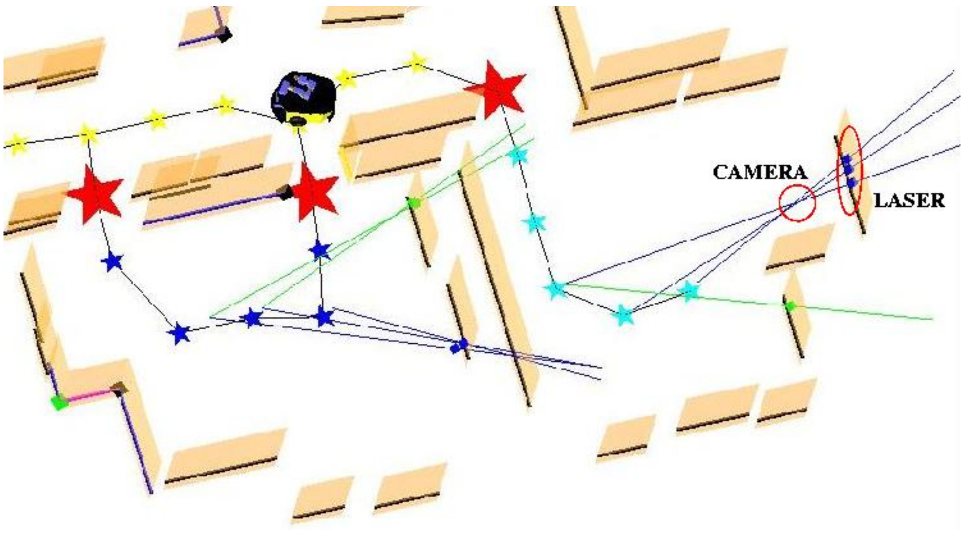


Figure 6.8: Searching for two objects in two rooms: a rice package placed on a dinner table is precisely localized by camera compared to laser.

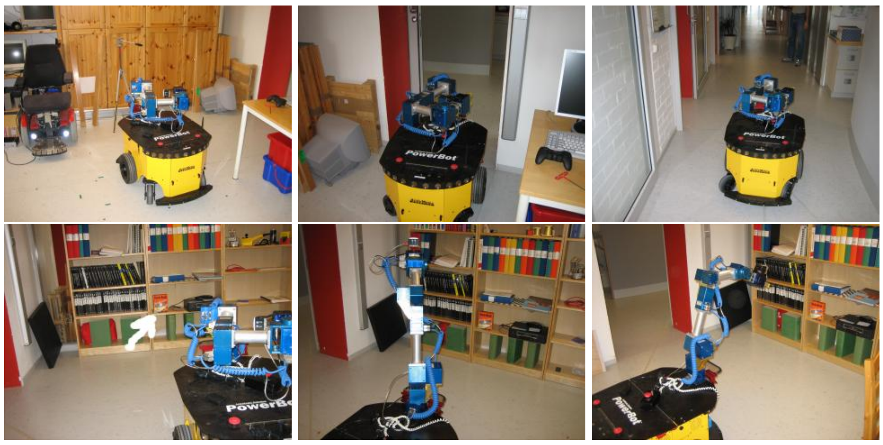


Figure 6.9: The robot is instructed to fetch the rice from the living room. It first plans a path and then follows it while avoiding obstacles. Once the goal position is reached, it verifies that the object is still there (marked with the white arrow). The robot signals that it has found the object by pointing to it with the arm.

structed to go to a specific room and pick up an object. The initial position of the robot is shown in the upper left corner in Fig. 6.9 in the room called *the manipulator lab*. This room is to the right in Fig. 6.8. The task the robot has to perform is to fetch the rice package from the living room (the left room on the map in Fig. 6.8). The robot first plans a path using the navigation graph and starts moving through the door to the hallway. It then enters the living room and moves to the closest point from which it has previously seen the object. At this point, it verifies that the object is still there. Then, it raises its arm towards the object, signaling that the object has been found. If the object was not found at the expected location, a new search for object is automatically initiated. A few example images taken during robot task execution are shown in Fig. 6.9. We note here that actual object pick up is outside the scope of this chapter. One method that could be used for this is described in (Petersson *et al.*, 2002).

6.6 Discussion

In this chapter, we have presented methods that enable the robot to learn mobile tasks. The integration of spatial and semantic information enables the robot to reason beyond simple geometrical level. The navigation module enables the robot to follow the teacher and learn room and place labels on the way, and the automatic object detection enables the robot to learn statistics over where certain objects are likely to be situated.

Through an extension to the RFCH-method from Section 3.4 we have shown that we can robustly detect and recognize objects in cluttered home and office environments, even though the objects are far away from the robot. We have also shown how the robot can learn to recognize new objects through interaction with the teacher.

The added mobility enables the robot to learn a whole new set of tasks. We believe the same strategy as in Chapter 5 can be used, although it has to be complimented with verbal or typed communication. We have shown how we build upon a SLAM system using different types of features (lines, points, SIFT) and sensors (laser, camera) which is able to construct a navigation graph that allows the robot to find its way through the feature-based map. A new method suitable for object detection has been presented. It is based on an appearance-based method, RFCH, used together with a feature-based method, SIFT. The experimental evaluation shows that the robot is able to successfully use the detection algorithm to detect and recognize objects in cluttered scenes.

Finally we have shown how to augment a SLAM map with information about object positions. One of the limitations of the current approach is that the system does not use the knowledge of the latest position of the object when searching for it in the frames node. This put some of the unnecessary burden to the search process and could easily be optimized. In a longer run, we believe that the system could also allow us to determine what type of objects are typically found in certain types of rooms which furthermore could help recognizing the function of a room that the robot has never seen before such as a kitchen or a workshop.

Chapter 7

Summary and Future Work

In the future, we envision robots to enter our homes and help us in our everyday lives. Filling the dishwasher, setting the table or cleaning the floor are just some of the tasks we want the robot to accomplish. Since there are many different tasks robots should be able to help us with, they will not come preprogrammed for each of them but instead be able to learn tasks from human demonstration. The reason is that, at manufacturing time the robot does not know the shape of our table, the appearance of our plates etc. This information has to be learned once the robot arrives. Clearly, the tasks also depend on the objects available and we may also want the robot to set the table in different ways. Thus, the robot has to learn tasks from a human teacher in the home environment. Programming by Demonstration has been recognized as one way of achieving this. The idea is that average users without deep technical knowledge should be able to teach the tasks by demonstrating them to the robot. However, despite much research effort around the world the gap is still large between the vision and the current performance. The many challenges include robot vision, grasping of objects, and also hardware reliability and robustness. One particular difficulty is that the robot is to operate in uncontrolled, dynamic environments. In this thesis, we have presented several contributions related to robot task learning in such environments. The following section summarizes them.

7.1 Summary

In this thesis, we have presented contributions related to direct and indirect robot learning. The simplest form of direct learning is storing the movement trajectory as the human is performing the task by controlling the robot. This, however, leads to an inflexible system and can only be used in a controlled environment, where the objects do not change their position between learning and execution phase. On the other hand, a human controlling the robot can identify changes in the environment and make intelligent decisions on how to proceed. However, precision control of a high degree-of-freedom robot is difficult and task execution tend to be slow and require much effort from the operator. It has been shown that *virtual fixtures* can be used to aid the operator in performing the task. They reduce the execution time and increase the overall precision. In Chapter 2, we introduce *adaptive virtual fixtures* that can be used in a Human Machine Collaborative System. While standard virtual fixtures only allow the operator to follow a predefined trajectory, adaptive virtual fixtures allow the operator to slightly change the trajectory to fit a changed task environment.

A challenging assumption that is decisive for the indirect learning approach is that the task relevant objects are not necessarily at the same location at execution time as when the learning took place. Thus, it is not sufficient to learn movement trajectories and absolute coordinates. Instead, the robot must be able to, at run-time, detect the objects and possible obstacles. Many Programming-by-Demonstration solutions aim to teach the robot a specific task. It may be inserting a peg in a hole, or a "spindle insertion task" (Chen and McCarragher, 2000). In this work, our goal was to develop a general learning system for manipulative motions. The most critical module here is the vision system. In Chapter 3, we developed an appearance-based method for object detection, which is used together with a model-based method for pose estimation. While the object detection method is simple to train and use, the pose estimation system requires many training images. In Chapter 5, we instead used a pure feature-based method for pose estimation. As the results show, the vision task is still difficult despite the quite simple setting. An interesting future approach would be to integrate the appearance-based and feature-based methods, something we did for object detection in Section 6 but only for object detection, not pose estimation.

In Chapter 4 we investigated some aspects in the field of object grasping. Two scenarios were considered: i) robot grasping in collaboration with a human, and ii) autonomous-based robot grasping. For the first scenario, we developed a neural network based grasp mapping system, that translates the human grasp into a robot grasp. The system can be trained by examples. The operator can then control the robot grasping through a data-glove. In the second scenario, the robot grasps objects autonomously. Here, a full model of the object is necessary, so that the robot can try out different approach vectors before grasping the object. By recognizing the human grasp during a demonstration, the robot can choose hand configuration and grasp strategy, to fit the object and current task. In this chapter, all results were evaluated in simulation, in contrast to the other chapters where a real robot was used for evaluation. Simulation has the advantage that it allows us to experiment with many different robot hands not available at our lab. The disadvantage is that the methods are not guaranteed to work outside the simulator. Hence, the most important future work here is to evaluate the methods using a real robot.

In Chapter 6 we presented a service robot application. The robot is able to navigate to a room and search for an object on demand. Another scenario is that the robot wanders around and automatically searches for objects. Then, on user's demand, it can provide information on where the object was seen last. However, the robot would be more useful if it could actually grasp the object and perform fetch-and-carry operations. If the object is standing on a table or the floor, without anything blocking it, it is certainly possible to grasp it. But if it instead is situated in a bookshelf or in a box, the grasping task becomes quite complicated.

7.2 Future Work and Perspective

One drawback with adaptive virtual fixture approach is that the task is approximated with a set of linear movements. Although this was not experienced as a problem during the experiments, further evaluation is necessary to see how the system performs with more complex and curved trajectories. In the experiments, we operate in three dimensions, while most work on virtual fixtures in the literature is limited to two dimensions. However, many tasks require both translation and rotation in 3D, thus six dimensions. Theoretically, adaptive virtual fixtures should be able to operate in joint space instead of Cartesian space. Then, the task can still be divided into straight line segments. The evaluation of such an approach is one of the more important subjects for future work on this system.

Currently, the indirect learning system is limited to pick-and-place operations, but we believe that due to the general methods used, the system can be extended to perform other actions. As we mention in (Ekvall and Kragic, 2006), one continuation would be to add learned movement trajectories as *actions* to the task planning system. To exemplify, consider the task *chop cucumber* from Section 5.4.1. Here, the chopping motion can be represented as a trajectory, and the robot could learn what prerequisites that need to be met are required before that action can be executed. The planner is then utilized to achieve this subgoal (knife in hand, cutting board on table, cucumber on cutting board).

There are several other ways to extend the work presented in Chapter 5. First of all, the grasps are currently provided to the robot as *a priori* knowledge. Instead, the robot could utilize the grasp recognition and mapping from Chapter 4 to automatically retrieve possible grasps from the demonstration. Another assumption in Chapter 5 is that the 3D-models of the objects are available. These should also be taught to the robot by showing the object to it, similar to the object appearance learning presented in Section 6.3.1. However, to automatically retrieve 3D-models puts great demands on the vision system, which must also be able to separate the unknown object from the background.

A natural, and actually quite simple extension to the task planning and execution system presented in Chapter 5 is to integrate it with the navigation module from Chapter 6. This would allow the task location to be encoded into the task, so that when the robot is asked to set the table, it first automatically moves to the kitchen.

In this thesis, we have presented several ways of teaching robots how to perform various tasks. It is clear that although we are a few steps closer, we are still far from an autonomous home robot butler that can aid us in everyday tasks. In the different chapters, we have shown learning methods on different levels, from simple imitation to full autonomy. Each level of learning has both advantages and drawbacks. Imitation learning is easy to implement but is very restricted in that the world must be exactly the same at run-time, as when the learning took place. On the other hand, adding intelligence to the robot allows it to adjust its behavior to deal with dynamic worlds, but restricts the problem space which the robot can solve. Eventually, we end up with a robot that can solve a very specific task fully autonomously. The learning then is simply to teach the robot how to deal with variations of that task. We have also shown different semi-autonomous solutions where the robot has some intelligence and aids the human in performing the task. Such a solution is good if a broader problem space is desired, but we do not have the demand of full autonomy. We believe this kind of solution will be mostly applicable in the industry, where the problem is well-defined but unattractive for a human to perform. The task may be dangerous (mine clearing), require perfect precision (surgery), be very heavy (construction), to name a few reasons.

The conclusion is that it is not possible to select a single level of autonomy as the optimal level for all tasks. A future multi-purpose robot will probably be equipped with learning on many different levels.

Bibliography

M. Aizerman, E. Braverman, and L. Rozonoer. 1964. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837.

J. Aleotti, S. Caselli, and M. Reggiani. 2004. Leveraging on a virtual environment for robot programming by demonstration. In *Robotics and Autonomous Systems, Special issue: Robot Learning from Demonstration*, volume 47, pages 153–161.

P. K. Allen, M. K. Reed, and I. Stamos. 1998. View planning for site modeling. In *Proc. DARPA Image Understanding Workshop*, pages 1181–1192.

H. Araujo, R. L. Canceroni, and C. M. Brown. 1996. A fully projective formulation for Lowe's tracking algorithm. Technical report 641, The University of Rochester, CS Department, Rochester, NY.

Ascension Tech. 2006. *Nest of Birds*. http://www.ascension-tech.com/products/nestofbirds.php.

C. G. Atkeson and S. Schaal. 1997. Robot learning from demonstration. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97) (ed. D. H. Fisher Jr.)*, pages 12–20.

D. H. Ballard. 1991. Animate vision. *Artificial Intelligence*, 48(1):57–86.

Y. Bar-Shalom and Y. Li. 1993. *Estimation and Tracking:Principles, techniques and software*. Artech House.

K. Bernardin, K. Ogawara, K. Ikeuchi, and R. Dillmann. 2003. A hidden markov model based sensor fusion approach for recognizing continuous human grasping sequences. In *Third IEEE Int. Conf. on Humanoid Robots*. URL `http://www.cvl.iis.u-tokyo.ac.jp/papers/all/0100.pdf`.

A. Bicchi and V. Kumar. 2000. Robotic grasping and contact: A review. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'00*, pages 348–353.

R. Bohlin and L. Kavraki. 2000. Path planning using lazy prm. In *Proceedings of the International Conference on Robotics and Automation*, pages 521–528.

C. Borst, M. Fischer, S. Haidacher, H. Liu, and G Hirzinger. 2003. DLR hand II: Experiments and experiences with an antropomorphic hand. In *Proceedings. IEEE International Conference on Robotics and Automation*, volume 1, pages 702–707.

H. Bourlard and N. Morgan. 1990. A continuous speech recognition system embedding MLP into HMM. *Advances in Neural Information Processing Systems*, 2:186–193.

C. Breazeal and B. Scassellati. 2002. Robots that imitate humans. *Trends in Cognitive Sciences*, 6(11):481–487.

L. Bretzner. 1999. *Multi-scale feature tracking and motion estimation*. PhD thesis, CVAP, NADA, Royal Institute of Technology, KTH.

C. J.C. Burges. 1998. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2:121–167.

B. Caputo. 2004. *A new kernel method for appearance-based object recognition: spin glass-Markov random fields*. PhD thesis, Royal Institute of Technology, Sweden.

A. Castellani, D. Botturi, M. Bicego, and P. Fiorini. 2004. Hybrid HMM/SVM model for the analysis and segmentation of teleoperation tasks. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2918–2923.

P. Chang and J. Krumm. 1999. Object recognition with color cooccurrence histograms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 498–504.

J. Chen and A. Zelinsky. 2003. Programming by demonstration: coping with suboptimal teaching actions. *International Journal of Robotics Research*, 22(5):299–319.

J. R. Chen and B. J. McCarragher. 2000. Programming by demonstration - constructing task level plans in a hybrid dynamic framework. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'00*, pages 1402–1407.

P. Chen, C. Lin, and B. Schölkopf. 2003. A tutorial on ν-support vector machines. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/nusvmtutorial.pdf`.

C. Cortes and V. Vapnik. 1995. Support-Vector Networks. *Machine Learning*, 20(3):273 – 297.

J. J. Craig. 1989. *Introduction to Robotics: Mechanics and Control*. Addison Wesley Publishing Company.

M. R. Cutkosky. 1989. On grasp choice, grasp models and the desing of hands for manufacturing tasks. *IEEE Transactions on Robotics and Automation*, 5(3):269–279.

D. DeMenthon and L. S. Davis. 1995. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141.

E. Dickmanns and V. Graefe. 1988. Dynamic monocular machine vision. *Machine Vision and Applications*, 1:223–240.

D. Ding, Y.-H. Liu, and S. Wang. 2000. Computing 3-d optimal formclosure grasps. In *Proc. of the 2000 IEEE International Conference on Robotics and Automation*, pages 3573–3578.

G. Dueck and T. Scheuer. 1990. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175.

M. Ehrenmann, O. Rogalla, R. Zöllner, and R. Dillmann. 2001. Teaching service robots complex tasks: Programming by demonstration for workshop and household environments. In *Proceedings of the 2001 International Conference on Field and Service Robots(FSR)*, pages 397–402.

S. Ekvall. 2005. CODID - CVAP Object Detection Image Database. URL `http://www.nada.kth.se/~ekvall/codid.html`.

S. Ekvall and D. Kragic. 2004. Interactive grasp learning based on human demonstration. In *IEEE/RSJ International Conference on Robotics and Automation*, volume 4, pages 3519–3524.

S. Ekvall and D. Kragic. 2005a. Grasp recognition for programming by demonstration. In *IEEE/RSJ IROS*, pages 748–753.

S. Ekvall and D. Kragic. 2005b. Integrating object and grasp recognition for dynamic scene interpretation. In *IEEE International Conference on Advanced Robotics, ICAR'05*, pages 331–336.

S. Ekvall and D. Kragic. 2006. Learning task models from multiple human demonstrations. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN'06*, pages 358–363,.

A. Elgammal, V. Shet, Y. Yacoob, and L. S. Davis. 2003. Learning dynamics for exemplar-based gesture recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 571–578.

M. O. Ernst and H. H. Bulthoff. 2004. Merging the senses into a robust percept. *Trends in Cognitive Sciences 8(4)*, pages 162–169.

S. Ferguson and G. Dunlop. 2002. Grasp recognition from myoelectric signals. In *Australian Conference On Robotics And Automation - ACRA, Auckland*, pages 78–83.

G. Ferretti, G. Magnani, P. Rocco, and L. Viganò. 2006. Modelling and simulation of a gripper with dymola. *Mathematical and Computer Modelling of Dynamical Systems*, 12:89–102.

R. E. Fikes and N. J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*, pages 189–205.

J Foley, A. van Dam, S. Feiner, and J. Hughes, editors. 1990. *Computer graphics - principles and practice*. Addison-Wesley Publishing Company.

J. Folkesson, P. Jensfelt, and H. I. Christensen. 2005. Vision SLAM in the measurement subspace. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA'05)*, pages 30–35.

Y. Freund and R. E. Schapire. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory*, pages 23–37.

H. Friedrich and R. Dillmann. 1995. Obtaining good performance from a bad teacher. In *Workshop: Programming by Demonstration vs Learning from Examples; International Conference on Machine Learning*. URL `citeseer.ist.psu.edu/kaiser95obtaining.html`.

H. Friedrich, R. Dillmann, and O. Rogalla. 1999. Interactive robot programming based on human demonstration and advice. *Sensor Based Intelligent Robots*, 1724:96–119.

H. Friedrich, S. Münch, and R. Dillmann. 1996. Robot programming by demonstration (rpd): Supporting the induction by human interaction. *Machine Learning*, 23:163–189.

G. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J.A. Fernández-Madrigal, and J. González. 2005. Multi-hierarchical semantic maps for mobile robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2278–2283.

V. Gengenbach, H.-H. Nagel, M. Tonko, and K. Schäfer. 1996. Automatic dismantling integrating optical flow into a machine-vision controlled robot system. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'96*, volume 2, pages 1320–1325.

R. C. Gonzalez and R. E. Woods. 1992. *Digital Image Processing*. Addison Wesley Publishing Company.

A. Gopalakrishnan and A. Sekmen. 2005. Vision-based mobile robot learning and navigation. In *IEEE International Workshop on Robot and Human Interactive Communication*, pages 48–53.

I. D. Horswill. 2000. *Behavior-Based Robotics, Behavior Design*. Technical report CS 395, Northwestern University.

K. Hyunsoo and P. Haesun. 2004. Prediction of protein relative solvent accessibility with support vector machines and long-range interaction 3D local descriptor. *Proteins: Structure, Function, and Bioinformatics*, pages 557–562.

M. Jägersand. 1997. *On-line Estimation of Visual-Motor Models for Robot Control and Visual Simulation*. PhD thesis, Univ. of Rochester.

P. Jensfelt, S. Ekvall, D. Kragic, and D. Aarno. 2005. Integrating slam and object detection for service robot tasks. In *IEEE International Conference on Intelligent Robots and Systems, Workshop on Mobile Manipulators: Basic Techniques, New Trends and Applications"*. URL `http://www.nada.kth.se/~patric/publications/workshop2005.pdf`.

P. Jensfelt, J. Folkesson, D. Kragic, and H. I. Christensen. 2006. Exploiting distinguishable image features in robotic mapping and localization. In Henrik I. Christensen, editor, *1st European Robotics Symposium (EUROS-06)*, pages 143–157.

S. Kang and K. Ikeuchi. 1993. Toward automatic robot instruction from perception-recognizing a grasp from observation. *IEEE Transactions on Robotics and Automation*, 9:432–443.

S. Kang and K. Ikeuchi. 1995. Toward automatic robot instruction from perception - temporal segmentation of tasks from human hand motion. *IEEE Transactions on Robotics and Automation*, 11(5):670 – 681.

S. Kang and K. Ikeuchi. 1997. Toward automatic robot instruction from perception-mapping human grasps to manipulator grasps. *IEEE Transactions on Robotics and Automation*, 13:81–95.

D. Kragic and H.I. Christensen. 2002. Model based techniques for robotic servoing and grasping. In *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, pages 299–304.

D. Kragic, P. Marayong, M. Li, A. M. Okamura, and G. D. Hager. 2005. Human-machine collaborative systems for microsurgical applications. *International Journal of Robotics Research*, 24:731–741.

G. M. Kruijff, H. Zender, P. Jensfelt, and H. I. Christensen. 2006. Clarification dialogues in human-augmented mapping. In *Proc. of the 1st Annual Conference on Human-Robot Interaction (HRI'06)*, pages 282–289.

A. B. Kuang, S. Payandeh, B. Zheng, F. Henigman, and C.L. MacKenzie. 2004. Assembling virtual fixtures for guidance in training environments. In *12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, HAPTICS '04.*, pages 367 – 374.

B. J. Kuipers. 1983. The cognitive map: Could it have been any other way? *H. L. Pick, Jr. and L. P. Acredolo (Eds.), Spatial Orientation: Theory, Research, and Application, New York: Plenum Press*, pages 345–359.

Y. Kuniyoshi, M. Inaba, and H. Inoue. 1994. Learning by watching, extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10:799–822.

V. Kyrki and D. Kragic. 2005. Integration of model-based and model-free cues for visual object tracking in 3d. In *IEEE International Conference on Robotics and Automation, ICRA'05*, pages 1566–1572.

H. K. Lee and J. H. Kim. 1999. An HMM-based threshold model approach for gesture recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 21:961–973.

T. Lefebvre, H. Bruyninckx, and J. De Schutter. 2005. Task planning with active sensing for autonomous compliant motion. *International Journal of Robotics Research*, 24(1): 61–81.

M. Li and A. M. Okamura. 2003. Recognition of operator motions for real-time assistance using virtual fixtures. In *11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'03)*, pages 125–131.

M. Li and R. H. Taylor. 2004. Spatial motion constraints in medical robot using virtual fixtures generated by anatomy. In *IEEE International Conference on Robotics and Automation, ICRA '04*, volume 2, pages 1270–1275.

R. Liang and M. Ouhyoung. 1998. A real-time continuous gesture recognition system for sign language. In *IEEE International Conference on Automatic Face and Gesture Recognition*, pages 558–567.

O. Linde and T. Lindeberg. 2004. Object recognition using composed receptive field histograms of higher dimensionality. In *17th International Conference on Pattern Recognition, ICPR'04*, volume 2, pages 1–6.

C. S. Lovchik and M. A. Diftler. 1999. The robonaut hand: a dexterous robot hand for space. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 907–912.

D. Lowe. 1999. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV 99)*, pages 1150–1157.

J. B. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 1:281–297. University of California Press.

O. Martínez Mozos, C. Stachniss, and W. Burgard. 2005. Supervised learning of places from range data using adaboost. In *Proc. of the IEEE International Conference on Robotics and Automation, ICRA'05*, pages 1742–1747.

M. J. Matarić. 2000. Getting humanoids to move and imitate. *IEEE Intelligent Systems*, 15:18–24.

B. W. Mel. 1997. SEEMORE: Combining color, shape and texture histogramming in a neurally inspired approach to visual object recognition. *Neural Computation*, 9:777–804.

A. T. Miller and P. K. Allen. 1999. Examples of 3D grasp quality computations. In *Proceedings of the of the 1999 IEEE International Conference on Robotics and Automation*, pages 1240–1246.

A. T. Miller and P. K. Allen. 2000. Graspit!: A versatile simulator for grasping analysis. In *Proceedings of the of the 2000 ASME International Mechanical Engineering Congress and Exposition*, pages 1251–1258.

A. T. Miller, S. Knoop, P. K. Allen, and H. I. Christensen. 2003. Automatic grasp planning using shape primitives. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1824–1829.

A. Morales, P. Azad, T. Asfour, D. Kraft, S. Knoop, R. Dillmann, A. Kargov, C. Pylatiuk, and S. Schulz. 2006a. An anthropomorphic grasping approach for an assistant humanoid robot. In *International Symposium on Robotics (ISR)*. URL `http://www.sfb588.uni-karlsruhe.de/publikationen/2006/R1R3_Morales_ISR06.pdf`.

A. Morales, E. Chinellato, A. H. Fagg, and A. P. del Pobil. 2004. Using experience for assessing grasp reliability. *International Journal of Humanoid Robotics*, 1(4):671–691.

A. Morales, P. J. Sanz, A. P. del Pobil, and A. H. Fagg. 2006b. Vision-based three-finger grasp synthesis constrained by hand geometry. *Robotics and Autonomous Systems*, 54 (6):494–512.

H. Murase and S. K. Nayar. 1995. Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*, 14:5–24.

A. Namiki, Y. Imai, M. Ishikawa, and M. Kaneko. 2003. Development of a high-speed multifingered hand system and its application to catching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2666–2671.

J. Napier. 1956. The prehensile movements of the human hand. *Journal of Bone and Joint Surgery*, 38B(4):902–913.

P. Newman and K. Ho. 2005. SLAM-loop closing with visually salient features. In *IEEE International Conference on Robotics and Automation, ICRA'05*, pages 635–642.

P. Newman, J. Leonard, J. D. Tardós, and J. Neira. 2002. Explore and return: Experimental validation of real-time concurrent mapping and localization. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA'02)*, pages 1802–1809.

M. N. Nicolescu and M. J. Mataric. 2003. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the Second International Joint International Conference on Autonomous Agents*, pages 241–248.

J. T. Nolin, P. M. Stemniski, and A. M. Okamura. 2003. Activation cues and force scaling methods for virtual fixtures. In *11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'03)*, pages 404–409.

K. Ogawara, K. Hashimoto, J. Takamatsu, and K. Ikeuchi. 2003. Grasp recognition using a 3D articulated model and infrared images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1590–1595.

K. Ogawara, J. Takamatsu, K. Kimura, and K. Ikeuchi. 2002. Generation of a task model by integrating multiple observations of human demonstrations. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, pages 1545–1550.

S. Payandeh and Z. Stanisic. 2002. On application of virtual fixtures as an aid for telemanipulation and training. In *10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, HAPTICS '02*, pages 18–23.

M. A. Peshkin, J. E. Colgate, W. Wannasuphoprasit, C. A. Moore, R. B. Gillespie, and P. Akella. 2001. Cobot architecture. *IEEE Transactions on Robotics and Automation*, 17(4):377–390.

L. Petersson, P. Jensfelt, D. Tell, M. Strandberg, D. Kragic, and H. I. Christensen. 2002. Systems integration for real-world manipulation tasks. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2500–2505.

R. Platt Jr, A. H. Fagg, and R. A. Grupen. 2003. Extending fingertip grasping to whole body grasping. In *Proc. of the International Conference on Robotics and Automation*, pages 2677–2682.

N. S. Pollard. 1994. *Parallel Methods for Synthesizing Whole-Hand Grasps from Generalized Prototypes*. Phd thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

N. S. Pollard. 2004. Closure and quality equivalence for efficient synthesis of grasps from examples. *International Journal of Robotic Research*, 23(6):595–613.

D. Prattichizzo and A. Bicchi. 1998. Dynamic analysis of mobility and graspability of general manipulation systems. *IEEE Transactions on Robotics and Automation*, 14(2): 241–257.

A. Pronobis, B. Caputo, P. Jensfelt, and H. I. Christensen. 2006. A discriminative approach to robust visual place recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'06*.

L. R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proc. of the IEEE, vol 77, no. 2*, pages 257–286.

S. Renals, N. Morgan, H. Bourlard, M. Cohen, and H. Franco. 1994. Connectionist probability estimators in HMM speech recognition. *IEEE Transactions on Speech and Audio Processing*, 2(1):161–174.

M. Riley, A. Ude, and C. G. Atkeson. 2000. Methods for motion generation and interaction with a humanoid robot: Case studies of dancing and catching. In *AAAI and CMU Workshop on Interactive Robotics and Entertainment*, pages 35–42.

D. Roobaert. 2001. *Pedagogical Support Vector Learning: A Pure Learning Approach to Object Recognition*. PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Sweden.

M. Ruchanurucks, S. Nakaoka, S. Kudo, and K. Ikeuchi. 2006. Humanoid robot motion generation with sequential physical constraints. In *IEEE International Conference on Robotics and Automation*, pages 2649–2654.

S. Russel and P. Norvig. 2003. *Artificial intelligence: A modern approach*. Second edition, Prentice Hall.

M. Rychetsky, S. Ortmann, and M. Glesner. 1999. Support vector approaches for engine knock detection. In *International Joint Conference on Neural Networks, IJCNN '99*, volume 2, pages 969–974.

S. Schaal. 1999. Is imitation route learning route to humanoid robots? *Trends in Cognitive Sciences*, 3:232–242.

B. Schiele and J. L. Crowley. 2000. Recognition without correspondence using multidimensional receptive field histograms. *International Journal of Computer Vision*, 36(1): 31–50.

R. Schmidt and T. Lee. 1999. *Motor Control and learning: a behavioral emphasis*. Human Kinetics, 3rd edition.

A. Selinger and R. C. Nelson. 2001. Appearance-based object recognition using multiple views. Technical Report 749, Comp. Sci. Dept. University of Rochester, Rochester NY.

R. Smith, M. Self, and P. Cheeseman. 1988. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotics Research*, pages 467–474.

T. Starner and A. Pentland. 1995. Visual recognition of American sign language using hidden Markov models. In *Proc. International Workshop on Automatic Face- and Gesture-Recognition*, pages 189–194.

M. Steedman. 1997. Temporality. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 895–938. Elsevier.

R. S. Sutton and A. G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

M. Swain and D. Ballard. 1991. Color indexing. *International Journal of Computer Vision*, 7:11–32.

R. H. Taylor and D. Stoianovici. 2003. Medical robotics in computer-integrated surgery. *IEEE Transactions on Robotics and Automation*, 19:765–781.

J. Tegin, J. Wikander, S. Ekvall, D. Kragic, and B. Iliev. 2006. Experience based learning and control of robotic grasping. In *IEEE-RAS International Conference on Humanoid Robots, Workshop: Towards Cognitive Humanoid Robots*.

C. Theobalt, J. Bos, T. Chapman, A. Espinosa, M. Fraser, G. Hayes, E. Klein, T. Oka, and R. Reeve. 2002. Talking to Godot: Dialogue with a mobile robot. In *IEEE International Conference on Intelligent Robots and Systems, IROS'02*, pages 1338–1343.

W. Townsend. 2000. The BarrettHand grasper – programmably flexible part handling and assembly. *Industrial Robot: An International Journal*, 27(3):181–188.

A. Ude. 1999. Robust estimation of human body kinematics from video. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1489–1494.

M. Vincze, M. Ayromlou, and W. Kubinger. 1999. An integrating framework for robust real-time 3D object tracking. In *Proceedings of the First International Conference on Computer Vision Systems, ICVS'99*, pages 135–150.

P. Viola and M. Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518.

Virtual Technologies Inc. 1995. CyberGlove. In *User's Manual*.

P. Wunsch and G. Hirzinger. 1997. Real-time visual tracking of 3D objects with dynamic handling of occlusion. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'97*, volume 2, pages 2868–2873.