Charles University in Prague
Faculty of Mathematics and Physics

DIPLOMA THESIS

Radim Krupička

Unsupervised learning based on images

Department of Software Engineering

Supervisor: Prof. Ing. Václav Hlaváč, CSc.
Study programme: Computer science

Acknowledgments

I would like to thank my supervisor Prof. Ing. Václav Hlaváč, CSc. for his leading of my diploma thesis, for inspiring discussions and ideas. Many thanks belong to Oleksandr Shekhovtsov for his help with the algorithms and his advices. Finally I am very grateful to Ing. Pavel Krsek, Ph.D., Ing. Vojtěch Franc, Ph.D. and Ing. Tomáš Werner, Ph.D. for their consultations.

Prague, April 18, 2007 Radim Krupička

Contents

Abstract

Title: Unsupervised learning based on images
Author: Radim Krupička
Department: Department of Software Engineering
Supervisor: Prof. Ing. Václav Hlaváč, CSc.
Supervisor's e-mail address: hlavac@fel.cvut.cz

Abstract: The aim of the project was to create an experimental environment for solving the children puzzle "shape sorter". This environment serve as a testbed for various learning tasks of cognitive nature with image feedback. This puzzle is a demonstrator of the EU project COSPAL, IST-004176. The industrial robot CRS A-465 with six degrees of freedom and its control system is used. The scene is observed by one static camera. The system processes the images from the camera and the further robot movement is based on the reinforcement learning.

Keywords: cognitive systems, COSPAL, image processing, reinforcement learning

Název práce: Učení bez učitele na základě obrazů
Autor: Radim Krupička
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: Prof. Ing. Václav Hlaváč, CSc.
E-mail vedoucího: hlavac@fel.cvut.cz

Abstrakt: Cílem diplomové práce bylo vytvoření experimentálního prostředí pro řešení dětského hlavolamu – skládačka, v němž je možné zkoušet různé metody pro učení systému s obrazovou zpětnou vazbou s kognitivními rysy. Hlavolam je demonstrátorem EU projektu COSPAL, IST-004176. Pro vkládání je použitý průmyslový robot se šesti stupni volnosti. Scéna je sledována jednou statickou kamerou. Systém zpracovává obraz z kamery a na základě zpětnovazebného učení rozhoduje o dalším pohybu robota.

Klíčová slova: kognitivní systémy, COSPAL, zpracování obrazu, zpětnovazebné učení

Chapter 1

Motivation

The aim of the diploma project is to contribute to studies in cognitive systems. The activities are part of the EU project COSPAL, IST-004176. COSPAL is the acronym for "COgnitive Systems using Perception-Action Learning". Methods developed within the project are tested on the demonstrator solving the children puzzle "shape sorter", see Figure 1.1. With the puzzle, a child is supposed to insert pieces (prisms with bases of different shapes) into holes with matching shape in a box. The diploma thesis follows principles in the rather new research field of cognitive systems.

The aim of the project is to create and use an experimental environment for solving the puzzle. This environment serves as a testbed for various learning tasks of cognitive nature which use feedback from images and robot actions. These modules are provided by the researchers of the COSPAL project. The limit case, which is not practically achievable, would be the case in which the system knows nothing about the task and environment a priori and builds basic concepts, relations from random movements and corresponding observations. It is supposed that a priori information will be used which will simplify the task.

The industrial robot CRS A-465 with six degrees of freedom and its control system is used (Figure 1.2). The robot is available in the Center for Machine Perceptions of the Faculty of Electrical Engineering of the Czech Technical University in Prague, where the advisor of this diploma thesis works. The base-level robot control and image acquisition software is the outcome of the Pavel Janda's diploma project which run in parallel.

The COSPAL project

The purpose of this project is to investigate design principles and architectures for technical systems, which are capable of learning basic cognitive skills

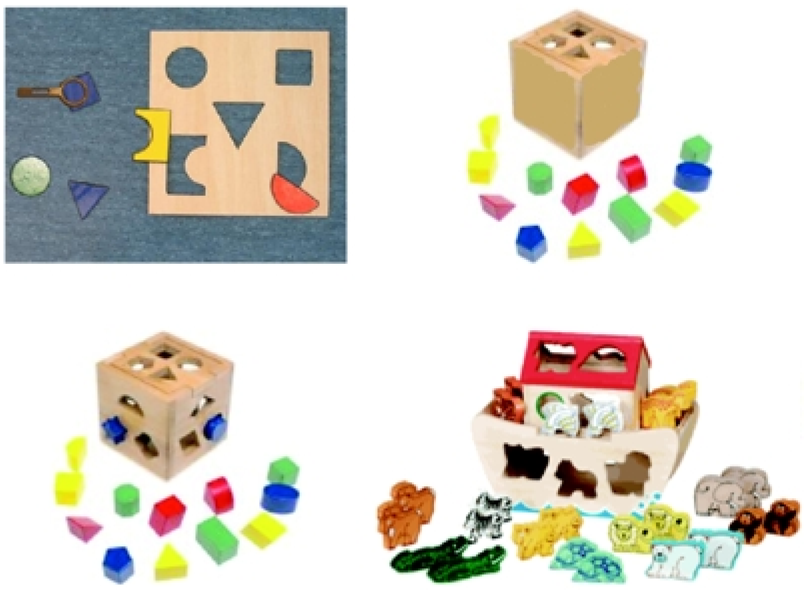
Figure 1.1: Different types of puzzles

in a similar way as humans do. It might appear surprising that, for instance, learning basic motor skills as done by children during the first three years of their life is hard to simulate in a technical system.

To understand why the "simpler" capabilities of humans are more difficult to simulate than "higher level" capabilities like, for instance, playing chess, one has to consider the different approaches to human learning and to implementing artificial systems, which appear to be intelligent.

Nowadays artificial systems typically consist of a predefined set of rules, which control the system actions depending on the inputs. Unforeseen inputs and constellations cannot be processed by such a system. Playing chess is a very well suited problem for this kind of system design, because the chess game follows exact rules and has a finite (although enormous) number of distinct constellations. Simulating certain cognitive capabilities of a one year old child, however, cannot be implemented by a rule based system for two reasons: First, there are no exact rules in basic behavioral schemes, they remain too fuzzy and qualitative to be used in an artificial system. Second, the number of possible inputs and the number of possible constellations are infinite in a way that for any number of rules there would exist cases that would not be covered by the rules [8].

The COSPAL architecture combines techniques from the field of artificial intelligence (AI) and learning of artificial neural networks (ANN). Feedback loops through the continuous and the symbolic parts of the system is established. This arrangement allows the perception-action feedback at several abstraction levels in the system. After an initial bootstrapping phase, incremental learning techniques are used to train the system simultaneously at different levels, allowing adaptation and exploration.

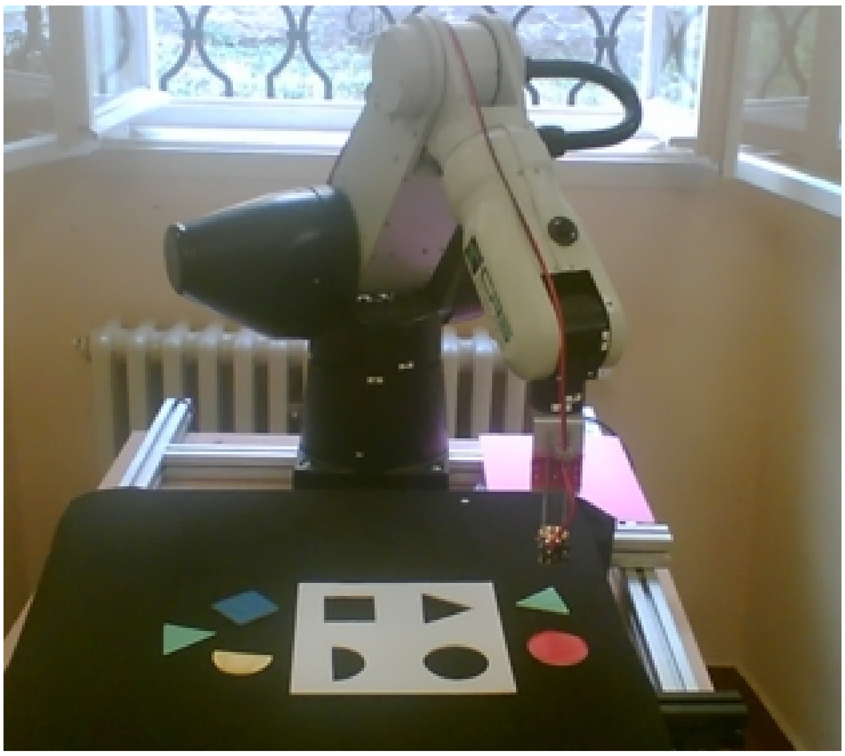

Figure 1.2: The industrial robot CRS A-465 with six degrees of freedom with installed electromagnet and with the puzzle.

Chapter 2

Problem formulation

The assignment of the diploma project is to design and implement an experimental environment for solving the “shape sorter puzzle”, see Figure 2.1. This puzzle serves as a demonstrator of the project COSPAL in which abilities of the artificial cognitive system should be demonstrated. The diploma project should prepare basic computer vision and machine learning tools allowing experimenting with the puzzle.

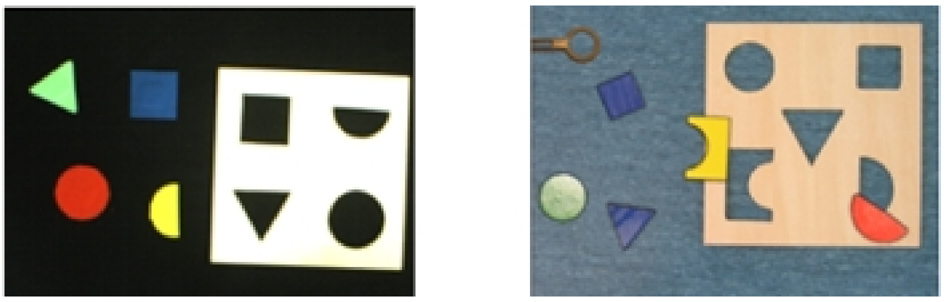

Figure 2.1: Shape sorter puzzle used in the COSPAL demonstrator.

The “shape sorter puzzle” consists of a box with holes of different shapes and several pieces of different shape. Each piece matches to only one hole. The task is to insert all pieces into appropriate holes.

In the COSPAL demonstrator, the pieces have to be manipulated by the arm of the industrial robot. One or more stationary cameras observe the scene and are used as a feedback of the system. If the information about the task and the puzzle were known to the system then the solution of the puzzle would be trivial, of course. In the project, the cognitive abilities are simulated by having a minimal knowledge about the puzzle and solving it by learning from experience.

This diploma project follows up the concurrently running diploma project of a fellow student Pavel Janda [5]. He is supposed to prepare more low-level modules for the robot control and the image acquisition.

The above mentioned general assignment can be decomposed into several subtask:

Create the experimental environment in MATLAB allowing to solve the puzzle and experiment with various facets of the solution. The expected features of the environment are:

- Integration of the robot hardware and video input prepared by Pavel Janda into a coherent system which should be able to move the robot arm and to acquire the image from cameras.
- Image processing and object detection part together with the representation of the internal states of the system based on the situation in the scene.
- Flexibility of the system which should manifest in ability to solve sufficiently robustly similar tasks.

Verification of the environment on the pilot problem which uses reinforcement learning as a learning algorithm.

Creation of the documentation of the written software.

Chapter 3

Introduction

The diploma project is a part of the COSPAL project. The architecture solving the cognitive tasks was designed for the COSPAL project. The COSPAL architecture, Figure 3.1, consists of three modules, forming a hierarchy in different levels of abstraction: the perception-action (PA) module, the abstraction-management (AM) module, and the symbolic processing (SP) module. The four universities participate in the COSPAL project as depicts Figure 3.1. The CVUT (Czech Technical University) is focused on the development of the AM module. The AM module is described in [11]. The AM module was divided into several subtasks, which were worked by CVUT staff on. One of these subtasks is this diploma project.

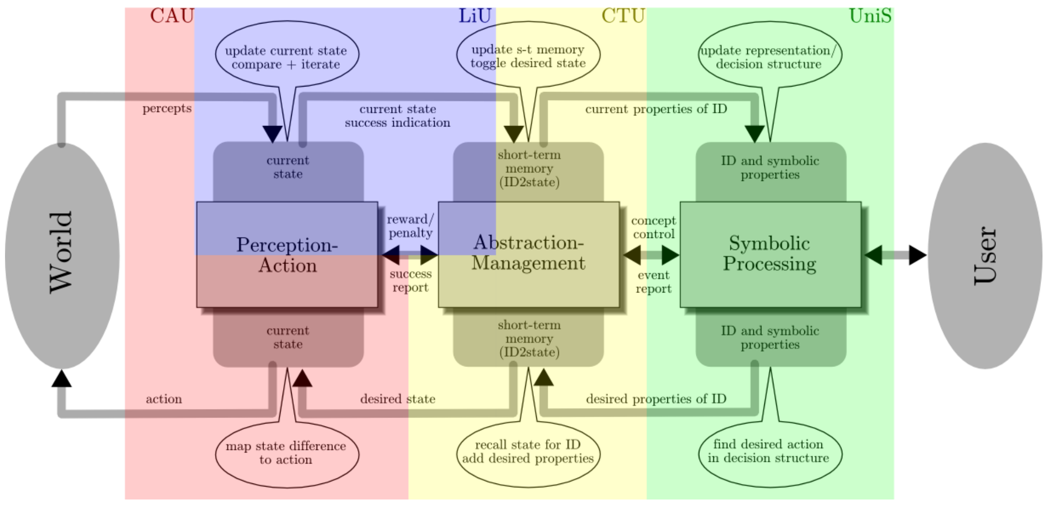

Figure 3.1: Current specification of the COSPAL architecture.

The diploma thesis includes these chapters:

State of the art chapter contains the basic knowledge, which is important to understanding the diploma thesis. This chapter defines the cognitive architectures and explains the segmentation and the classification methods. The last section of this chapter is focused on the reinforcement learning.

My solution chapter deals with the theoretic solution of the problemme. The image processing and artificial intelligent algorithms are presented here.

Implementation chapter describes how and where the algorithms were implemented.

Chapter 4

State of the art

4.1 Cognitive architectures

4.1.1 Task outline

Cognitive architectures are based on the belief that understanding (human) cognitive processes can be helped by implementing them on a computational level. Cognitive architectures can be characterized by certain properties or goals that are follows [12]:

1. Implementation of not just various different aspects of a cognitive behavior but of a cognition as a whole. This is in contrast to same cognitive models.

2. The architecture often tries to reproduce the behavior of the modelled system (human) in a way that timely behavior (reaction times) of the architecture and modelled cognitive systems can be compared in detail.

3. Robust behavior in the face of error, the unexpected, and the unknown.

4. Learning.

5. Parameter-free: The system should not depend on parameter tuning (in contrast to, e.g., artificial neural networks).

4.1.2 Objectives

The diploma project is focused on vision based artificial cognitive systems (ACS). State-of-the-art of these tend to be either based on classical artificial intelligence (AI) paradigm or follow the ideas of artificial neural networks (ANN) [2]. Combinations of both approaches are rare.

In AI systems, the designer supplies knowledge and scene representations to a largest possible extent. For designing AI based ACSs, a rule based system is set up, which typically uses some features extracted by low-level image processing in order to follow a path through a decision tree, ending up in a certain system response. These responses can be of different kind, e.g., the choice in a database, the movement of robot arm, or estimation of certain parameters. The discrete choices made everywhere in the system design, and in particular in the lower levels, lead necessarily to unstable and non-robust results.

ANNs derive knowledge about the external world through exploration. That means ANN based ACS are supposed to tackle the same goals by a totally different method, which in practice boils down to a non-linear local optimization process. The constraints on this optimization are typically called bias and its search space is reflected by the term variance. Instead of selecting features and setting up rules, these system require a good choice of constraints leaving sufficient degrees of freedom: the bias-variance dilemma is still an unsolved problem in this area [2].

Both paradigms, AI and ANN, have similar goals concerning the system performance, but have very much different drawbacks and advantages. It's possible to combine both methods in the hope that the drawbacks are compensated and the overall system performance is improved.

As a result of insufficient interweaving, todays ACS show the following drawbacks:

- The system is either stable and robust or competent, i.e., stable and robust system are of little practical value and systems fulfilling relevant tasks are too specialized and not fault tolerant.
- No association of appearance and function of objects is possible.
- The perception cannot sufficiently adapt to the current action state or current percepts, i.e., the performance of the perception design limits the performance of the whole system.

4.2 Segmentation

The main goal of image Segmentation is to divide an image into parts that have a strong correlation with objects or areas of the real world depicted in the image [9].

Most image segmentation approaches can be placed in one of three categories:

- Characteristic feature thresholding or clustering.
- Edge-based segmentation.
- Region-based segmentation.

The main segmentation problem is the image data ambiguity that is often accompanied by information noise. **Characteristic feature thresholding** or clustering represents the simplest image segmentation process, and it is computationally inexpensive and fast. Thresholding does not exploit spatial information, and thus ignores information that could be used to enhance the segmentation results. Gray-level thresholding is the simplest segmentation process. Many objects or image regions are characterized by constant reflectivity or light absorption of their surfaces; a brightness constant or threshold can be determined to segment objects and background. Thresholding can easily be done in real time. Complete segmentation can result from thresholding in simple scenes. Thresholding is the transformation of an input image f to an output (segmented) binary image g as follows:

$$g(i,j)\begin{cases} = 1 \text{ for } f(i,j) \geq T, \\ = 0 \text{ for } f(i,j) < T, \end{cases} \tag{4.1}$$

where T is the threshold, $g(i,j) = 1$ for image elements of objects, and $g(i,j) = 0$ for image elements of the background.

Edge-based segmentation relies on edges found in an image by an edge detecting operator. The edge detection does exploit spatial information by examining local edges found throughout the image data, it does not necessarily produce closed connected region boundaries. For simple noise-free data, detection of edges usually results in straightforward region boundary delineation. However, edge detection on noisy, complex image data often produces missing edges and extra edges that cause the detected boundaries to no necessarily from a set of close connected curves that surround connected regions. Three types of region borders may be formed due to 'pixelized' nature of the image: inner, outer, and extended. The inner border is always part of region, but the outer border never is. Therefore, using inner or outer border definition, two adjacent regions never have a common border. Extended borders are defined as single common borders between adjacent regions still being specified by standard pixel co-ordinates.

Region growing segmentation should satisfy the following condition of complete segmentation:

$$R = \bigcup_{i=1}^{S} R_i \quad , \quad R_i \cap R_j = \emptyset \quad , \quad i \neq j. \tag{4.2}$$

and the maximum region homogeneity conditions:

$$H(R_i) = \text{true} \qquad i = 1, 2, \ldots, S. \tag{4.3}$$

$$H(R_i \cup R_j) = \text{false} \qquad i \neq j, \qquad R_i \text{ adjacent to } R_j. \tag{4.4}$$

Three basic approaches to region growing exist:

Region merging starts with an oversegmented image in which regions satisfy equation (4.3). Regions are merged to satisfy condition (4.4) as long as equation (4.3) remains satisfied.

Region splitting is the opposite of region merging. Region splitting begins with an undersegmented image which does not satisfy condition (4.3). Therefore, the existion image regions are sequentially split to satisfy conditions (4.2),(4.3), and (4.4).

A combination of splitting and merging may result in a method with the advanteges of both other approaches. Split-and-merge processing options are available, the starting segmentation does not have to satisfy either condition (4.3) or (4.4).

Region growing approaches to segmentation are preferred in the project because region growing exploits spatial information and guarantees the formation of closed, connected regions.

4.3 Object recognition

The task of object recognition is to determine which, if any, of a given set of objects appear in a given image or image sequence. Object recognition solves a problem of matching models from a database with representations of those models extracted from the image luminance data [9].

An object is physical unit, usually represented in image analysis and computer vision by a region in a segmented image. The set of objects can be divided into disjoint subsets, that, from the classification point of view, have some common features. These subsets are called classes. The definition of how the objects are divided into classes is ambiguous and depends on the classification goal.

Object recognition is based on assigning classes to objects. The device that does these assignments is called the classifier. The number of classes is usually known beforehand, and typically can be derived from the problem specification. Nevertheless, there are approaches in which the number of classes may not be known. The classifier (similarly to a human) does not decide about he class from the object itself–rather, sensed object properties serve this purpose. For example, to distinguish steel from sandstone, we do not have to determine their molecular structures, although this would describe these materials well. Properties such as texture, specific weight, hardness, etc., are used instead.

The main pattern recognition steps are shown in Figure 4.1. The block "Construction of formal description" is based on the experience and intuition of the designer. A set of elementary properties is chosen which describe some characteristics of the object; these properties are measured in an appropriate way and form the description pattern of the object. These properties can be either quantitative or qualitative in character and their form can vary (numerical vectors, chains, etc.). The theory of recognition deals with the problem of designing the classifier for the specific (chosen) set of elementary object descriptions.

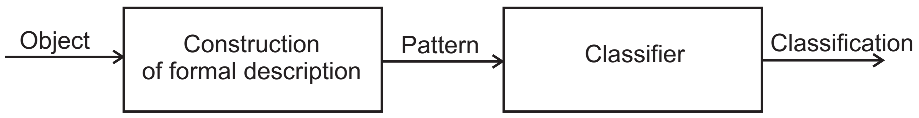

Figure 4.1: Main pattern recognition steps

Statistical object description uses elementary numerical descriptions called features, $x_1, x_2, \ldots, x_n$. The pattern (also referred to as pattern vector, or feature vector) $x = (x_1, x_2, \ldots, x_n)$ that describes an object is a vector of elementary descriptions, and the set of all possible patterns forms the patterns space X (feature space). If the elementary descriptions were appropriately chosen, similarity of objects in each class results in the proximity of their patterns in the pattern space. The classes form clusters in the feature space, which can be separated by a discrimination curve (or hyper-surface in a multi-dimensional feature space) - Figure 4.2.

Intuitively, we may expect that separable classes can be recognized without errors. The majority of object recognition problems do not have separable classes, in which case the locations of the discrimination hyper-surfaces in the feature space can never separate the classes correctly and some objects will always be misclassified [9].

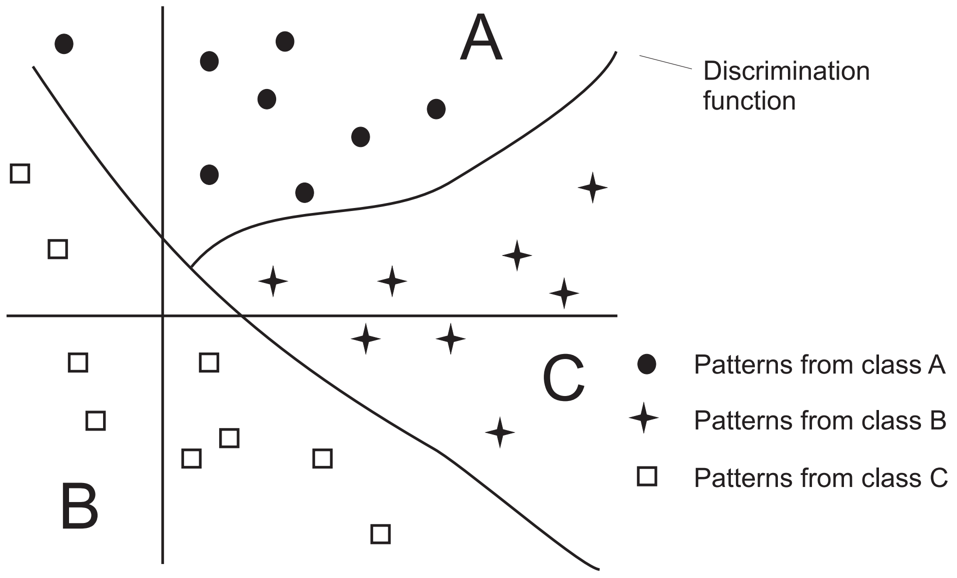

Figure 4.2: A general discrimination function.

4.4 Reinforcement learning

Reinforcement learning [10] is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, as in the most forms of machine learning, but instead must discover which actions yield the highest reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.

Reinforcement learning refers to a class of problems in machine learning which postulate the agent exploring an environment. The agent perceives its current state and takes actions. The environment, in return, provides a reward (which can be positive or negative). Reinforcement learning algorithms attempt to find a policy for maximizing cumulative reward for the agent over the course of the problem.

There are two main strategies for solving reinforcement-learning problems. The first one is to search in the space of behaviors in order to find the behavior that performs well in the environment. This approach has been usually taken by works in genetic algorithms and genetic programming. The second strategy is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world. In this diploma project, the second set of techniques is used because they take advantage of the special structure of reinforcement-learning problems that is no available in optimization problems in general [10].

4.4.1 Elements of Reinforcement Learning

Beyond the agent and the environment, one can identify four main subelements of a reinforcement learning system: a policy, a reward function, a value function, and, optionally, a model of the environment [7].

A policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.

A reward function defines the goal in a reinforcement learning problem. Roughly speaking, it maps each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state. A reinforcement learning agent's sole objective is to maximize the total reward it receives in the long run. In general, reward functions may be stochastic.

A value function specifies what is good in the long run. The value of a state is the total amount of reward. An agent can expect to accumulate over the future, starting from that state.

We seek actions that bring about states of highest value, not the highest reward, because these actions provide the greatest amount of reward over the long run. In decision-making and planning, the derived quantity called value is the one with which we are most concerned. Rewards are basically given directly by the environment, but values must be estimated and reestimated from the sequences of observations an agent makes over its entire lifetime.

Model of the environment is mimics the behavior of the environment. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced.

4.4.2 Reinforcement-learning model

In the standard reinforcement-learning model, an agent is connected to its environment via perception and action, as depicted in Figure 4.3. On each step of interaction the agent receives as input i, some indication of the current state s of the environment; the agent then chooses an action a to generate as output. The action changes the state of the environment. The value of this state transition is communicated to the agent through a scalar reinforcement signal r. The agent's behavior B should choose actions that tend to increase the long/run sum of values of the reinforcement signal. The reinforcement-based learning system can learn to do this over time by systematic trial and error, guided by a wide variety of algorithms. Formally, the model consists of:

- a discrete set of environment states S,
- a discrete set of agent actions A,
- a set of scalar reinforcement signals; typically 0,1, or the real numbers.

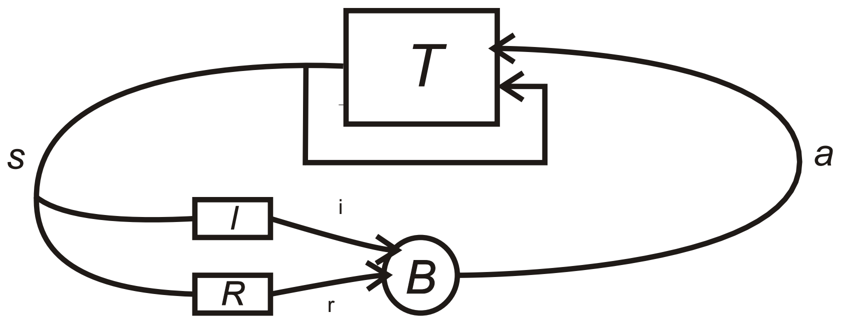

Figure 4.3: The standard reinforcement learning model.

The agent's job is to find a policy π mapping states to actions that maximizes some long-run measure of reinforcement. We expect, in general, that the environment will be non-deterministic; that is, that taking the same action in the same state on two different occasions may result in different next states and/or different reinforcement values.

Reinforcement learning differs from the more widely studied problem of supervised learning in several ways. The most important difference is that there is no presentation of input/output pairs. Instead, after choosing an action, the agent is told the immediate reward interests. It is necessary for the agent to gather useful experience about the possible system states, actions, transitions and rewards actively with the aim to act optimally. Another difference from supervised learning is that the on-line performance is important: the evaluation of the system is often concurrent with learning [7].

4.4.3 Basic principles

One major difference between reinforcement learning and supervised learning is that a reinforcement-learner must explicitly explore its environment.

The simplest possible reinforcement-learning problem is known as the k-armed bandit problem [1]. The agent is in a room with a collection of k gambling machines (each called a "one-armed bandit" in colloquial English). The agent is permitted a fixed number of pulls, h. Any arm may be pulled on each turn. The machines do not require a deposit to play; the only cost is in wasting a pull while playing a locally suboptimal machine. When arm i is pulled, machine i pays off 1 or 0, according to some underlying probability parameter p_i, where payoffs are independent events and the $p_i s$ are unknown. What should the agent's strategy be?

This problem illustrates the fundamental trade-off between exploitation and exploration. The agent might believe that a particular arm has a fairly high payoff probability; should it choose that arm all the time, or should it choose another one that it has less information about, but seems to be worse? Answers to these questions depend on how long the agent is expected to play the game; the the longer the game lasts, the worse the consequences of prematurely converging on a sub-optimal arm, and the more the agent should explore.

There is a wide variety of solutions to this problem. I will show two of them.

Dynamic-Programming Approach

If the agent is going to be acting for a total of h steps, it can use basic Bayesian reasoning to solve for an optimal strategy [1]. This requires an assumed prior joint distribution for the parameters $\{p_i\}$ that each p_i is independently uniformly distributed between 0 and 1. We compute a mapping from "belief states" (summaries of the agent's experiences during this run) to actions. A belief state can be represented as a tabulation of action choices and payoffs: $\{n_1, w_1, n_2, w_2, \dots, n_k, w_k\}$ denotes a state of the play in which each arm i has been pulled n_i times with w_i payoffs. We write $V^*(n_1, w_2, \dots, n_k, w_k)$ as the expected payoff remaining, given that a total of h pulls are available, and we use the remaining pulls optimally.

If $\sum_i n_i = h$ then there are no remaining pulls, and

$$V^*(n_1, w_2, \dots, n_k, w_k) = 0.$$

This is the basis of a recursive definition. If we know the V^* value for all belief states with t pulls remaining, we can compute the $V*$ value of any belief state with $t+1$ pulls remaining:

$$\begin{aligned} & V^*(n_1, w_2, \ldots, n_k, w_k) = \\ & = \max_i E \left[\begin{array}{l} \text{Future payoff if agent takes action } i \\ \text{then acts optimally for remaining pulls} \end{array} \right] \\ & = \max_i \left(\begin{array}{l} \rho_i V^*(n_1, w_i, \ldots, n_i+1, w_i+1, \ldots, n_k, w_k) + \\ (1-\rho_i)(n_1, w_i, \ldots, n_i+1, w_i, \ldots, n_k, w_k) \end{array} \right), \end{aligned} \tag{4.5}$$

where ρ_i is the posterior subjective probability of action i paying off given n_i, w_i and our prior probability. For the uniform priors, which result in a beta distribution, $\rho_i = (w_i + 1)/(n_i + 2)$.

The expense of filling in the table of V^* values in this way for all attainable belief states is linear in the number of belief states times actions, and thus exponential in the horizont.

Greedy strategies

The first strategy that comes to mind is to always choose the action with the highest estimated payoff. The flaw is that early unlucky sampling might indicate that the best action's reward is less than the reward obtained from a suboptimal action. The suboptimal action will always be picked, leaving the true optimal action starved of data and its superiority never discovered. An agent must explore it to ameliorate this outcome.

A useful heuristic is "optimism in the face of uncertainty" in which actions are selected greedily, but strongly optimistic prior beliefs are put on their payoffs so that a strong negative evidence is needed to eliminate the action from consideration. This still has a measurable danger of starving an optimal but unlucky action, but the risk of this can be made arbitrarily small.

Chapter 5

My solution

This chapter describes the theoretic solution of the project. The solution of the project is divided into four sections. The division and the dependencies are shown in Figure 5.1. The figure depicts the environment and the system, which changes the environment with the robot. The environment and the robot abilities are described in Section 5.1. The system acquires the images of the environment for processing. After the image acquisition, the objects are detected in the image. This is described in Section 5.2. The detected objects are included to the scene and scene is processed, Section 5.3. Finally, the learning algorithm decides the next robot movement, Section 5.4.

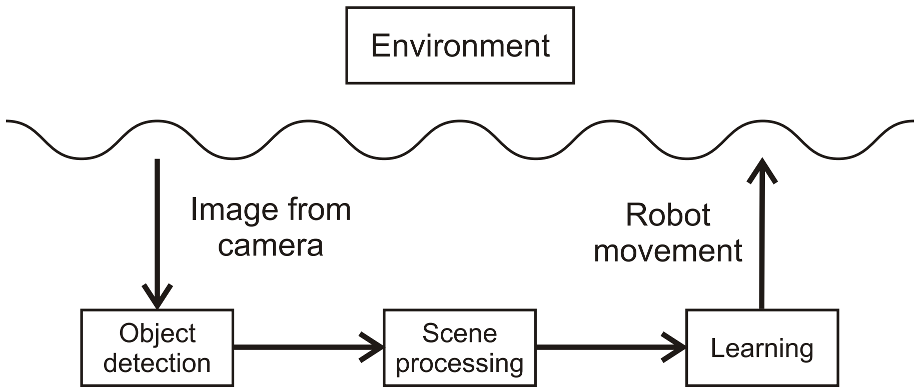

Figure 5.1: The solution scheme.

5.1 Robot's world and body

5.1.1 Physical setup

The shape sorter board, pieces and the robot gripper were made deliberately simple. Only flat 2–dimensional arrangement was chosen to keep the image analysis and magnetic gripping as easy as possible. The scene consists of the table, the board, and objects. An object can be:

- Piece. Pieces are flat magnetic objects of equal height but possibly different shapes and different colors.
- Hole. Holes in the flat board can have arbitrary sizes and shapes.
- Distractor is something (drawn image, non-magnetic object) unmovable and ungrippable on the table.

These examples of the setup are illustrated in Figure 5.2. The board lies on the table. The pieces can lie on the table or on the board. Also, they can be inserted into the holes in the board (Figure). There is a single, stationary, top-view camera not attached to the robot. To avoid perspective distortions, the higher the camera is positioned the better.

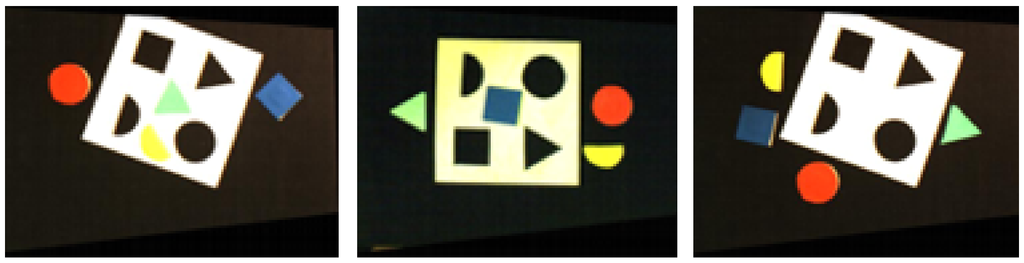

Figure 5.2: Examples of the puzzle arrangement.

5.1.2 Innate elementary capabilites

The vertical position of the gripper can take only two levels: the grip/release level and the move level. The robot has the following innate action capabilities:

- Grip. It is assumed that the gripper is in the move level above the object. Descend to the grip level and turn on the electromagnet in the gripper. The grip action succeeds if the object is a piece. Gripping an empty space is considered a failure.

- Move. It is assumed that the gripper is in the grip level and is holding an object. The electromagnet is turned on, the gripper is risen to the move level and moved horizontally in the move level above a specified object. Moving always succeeds, because there is no an grippable and unmovable object in the gripper trajectory.

- Release. It is assumed that the gripper is in the move level. The magnetic gripper is descended to the release level and the magnet is turned off.

After each action, the system produces the success/failure signal. Detection of success/failure is done on the images seen by the camera.

Besides these manipulation capabilities, the system has the following innate perceptual capabilities:

- Object detection and tracking. The system detects objects and tracks their identities in time. Ocassionally, the track can be interrupted (i.e., the identity lost), e.g., after an accidental moving of an object occluded by the robot arm.

- Feature extraction. For each object, a set of features is extracted from its iconic representation (i.e., the image of its blob). The system does not know the meaning of individual features, meanings will be assigned later by learning classifiers acting on this feature space. The features themselves are not subject to learning and stay the same during the system's lifetime. The feature set is a vector

  $$(x, y,\ \ t_1, \ldots, t_n,\ \ s_1, \ldots, s_m),$$

  where:

  - x, y is the position of the reference point of the blob.
  - $t_1, \ldots, t_n$ are texture descriptors. In the simplest case of textureless puzzle pieces, they can be just mean R, G, B values of the blob region.
  - $s_1, \ldots, s_m$ are shape descriptors. For example, they can be textbook shape descriptors, like the ratio of the outline length and square root of the area, a few centered moments, or a few affine invariants [9].

- Piece insertion detection. The system should detect the insertion of a piece to the appropriate hole. The detection could be accomplished by an external sensor, user input or camera. The camera detection is used in our system. For this purpose the system can recognize object's types (piece/hole) and find suitable hole for each piece. This information is explored only for the detection and the robot can't use it. When the robot tries to insert a piece to an appropriate hole, the system automatically aligns the piece with the hole and returns the 'success' signal.

5.1.3 Functionality and example session

The system is able to perform several subtasks, corresponding to forces (modes of behavior) driving the system do something:

1. Identify the model of the world. I.e., become able to predict outcomes of actions (or find out the set of allowed actions, or something like that).
2. Find a goal to be fulfilled (e.g., put all objects to holes). At the current state of development, we can temporarily make this goal innate.
3. Fulfill the given goal.

Note that subtask 3 cannot be performed without performing subtask 2 first, and both subtask cannot be done without performing subtask 1. However, attempting to perform solely subtask 1 for a longer time than is necessary for fulfilling the other two subtasks would be a mistake. Ideally, there should be a trade-off between them optimizing some criterion (fitness function).

Example: one piece, 2D distractor

The scene consists of two 2D red distractors of rectangular shape and a yellow object laying on one of distractors. There are two allowed positions: the object laying on the first distractior or on the second distractor. When laying on a distractor, the object makes it completely invisible.

The robot tries to grip a object, i.e., the gripper descends to the grip level above the object, turns the magnet on and goes back with the object. This produces the 'success' signal. The robot makes a hypothesis that all objects are gripable.

The robot tries to move the gripped object. This produces the 'success' signal. The robot makes a hypothesis that all objects are moveable. The gripper with the object moves to a different, randomly chosen, position – in our case there is only one such position, namely above the other distractor.

The robot tries to release the object, i.e., the gripper tries to descend to the release level. This produces the ‘success’ signal. Therefore the robot makes a hypothesis that any object can be placed on any other object. Note, if there was another piece under the gripper, the release attempt would produce ‘failure’ because descending to the release level would cause a collision.

In subsequent random actions, the system tries to falsify the current hypotheses. It tries to grip the distractor, which fails. The way out from the contradictory set of hypotheses (‘all objects are gripable’ and ‘all objects are not gripable’) is to discover a hidden property of objects that would resolve the contradiction. The robot has an innate capability to compute the feature vector $(x, y,\ t_1, \dots, t_n,\ s_1, \dots, s_m)$ for each object. It tries to resolve the contradiction by teaching a linear classifier [9] (devoted to gripability) acting on this feature space, using a labeled training set with two elements being the two actions performed so far. It is always possible to run a hyperplane in this $(m + n + 2)$-dimensional feature space separating these two elements. However, this hyperplane is highly ambiguous – therefore the robot has no way to find out whether the gripability/non-gripability is distinguished by position, texture/color, shape, or their linear combination.

5.2 Object detection

5.2.1 Segmentation

Stable image segmentation is very important for correct work of the algorithm. There were implemented three different image segmentation algorithms. Finally the Graph-Based image segmentation algorithm was used for its stability, short running time and easy configurability. Algorithm proof and more detailed description is in the report [3].

Graph-based image segmentation

Graph-based image segmentation techniques generally represent the problem in terms of a graph where each node corresponds to a pixel in the image, and the graph edges connect certain pairs of neighboring pixels. Let $G = (V, E)$ undirected graph with vertices $v_i \in V$, the set of elements to be segmented, and edges $(v_i, v_j) \in E$ with corresponding weight $w((v_i, v_j))$, which is a non-negative measure of the dissimilarity between neighboring elements v_i and v_j. In this case of image segmentation, the elements in V are pixels and the weight of an edge is some measure of the dissimilarity between the two pixels connected by that edge (e.g., color, difference in intensity, location or some other local attribute).

In the graph-based approach, a segmentation S is a partition of V into components such that each component (or region) $C \in S$ corresponds to a connected component in a graph $G' = (V, E')$, where $E' \subseteq E$. In other words, any segmentation is induced by a subset of the edges in E.

We define predicate D for evaluating whether or not there is evidence for a boundary between two components in a segmentation. This predicate is based on measuring the dissimilarity between elements along the boundary of the two components relative to a measure of the dissimilarity among neighboring elements within each of two components. The resulting predicate compares the inter-component differences to the within component differences and is thereby adaptive with respect to the local characteristics of the data.

The internal difference of a component $C \subseteq V$ is the largest weight in the minimum spanning tree of the component, $MST(C, E)$. That is,

$$Int(C) = \max_{e \in MST(C,E)} w(e). \tag{5.1}$$

The difference between two components $C_1, C_2 \subseteq V$ to be the minimum weight edge connecting the two components.

$$Dif(C_1, C_2) = \min_{v_1 \in C_1, v_3 \in C_2, (v_i, v_j) \in E} w((v_i, v_j)). \tag{5.2}$$

If there is no edge connecting C_1 and C_2 then $Dif(C_1, C_2) = \infty$.

The region comparison predicate evaluates if there is evidence for a boundary between a pair or components. If the difference between the components, $Dif(C_1, C_2)$, is large relative to the internal difference within at least one of the components, $Int(C_1)$ and $Int(C_2)$. A threshold function is used to control the degree to which the difference between componets must be larger than the minimal internal difference. We define pairwise comparison predicate as,

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2), \\ \text{false} & \text{otherwise .} \end{cases} \tag{5.3}$$

where the minimum internal difference, $MInt$, is defined as,

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)). \tag{5.4}$$

The threshold function τ controls the degree to which the difference between two components must be greater than their internal differences in order for there to obtain the evidence about boundary between then (D to be true).

For small components, Int(C) is not a good estimate of the local characteristics of the data. In the extreme case, when $|C| = 1$, $Int(C) = 0$. Therefore, a threshold function is used based on the size of the component,

$$\tau(C) = k/|C|, \tag{5.5}$$

where $|C|$ denotes the size of C, and k is some constant parameter. That is, for small components stronger evidence is required for a boundary. In practice, k sets a scale of observation. A larger k causes a preference to larger components. For our purpose k was set to 1500. However, that k is not a minimal component size. Smaller components are allowed when there is a sufficiently large difference between the neighboring components.

Segmentation algorithm

The input is a graph $G = (V, E)$, with n vertices and m edges. The output is a segmentation of V into components $S = (C_1, \ldots, C_r)$.

1. Sort E into $\pi = (o_1, \ldots, o_m)$, by non-decreasing edge weight.
2. Start with a segmentation S^0, where each vertex v_i is in its own component.
3. Repeat step 4 for $q = 1, \ldots, m$.
4. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components then merge the two components, otherwise do nothing. More formally, let C_i^{q-1} be the component of S^{q-1} containing v_i and C_j^{q-1} the component containing v_j. If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ then S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise $S^q = S^{q-1}$.
5. Return $S = S^m$.

For integer weights this algorithm can be performed in linear time using counting sort. In general, it can be performed in $O(m \log m)$ time using any one of several sorting methods.

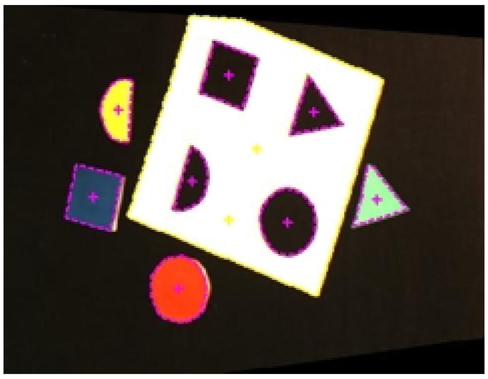

Figure 5.3: The segmentation result produced by this algorithm. Distinct classes are labeled by different colors. The crosses in the image give the representative points of segmented objects.

5.2.2 Feature extraction

The image segmentation find objects in the image. Each object is characterized by a lot of several features. For future processing there is necessary extract this features [9]:

- Area of the object.
- Filled area of the object - area of the object with holes.
- Center of mass position.
- Major and minor axis length.
- Eccentricity.
- Bounding box coordinates.
- Convex hull and convex area.
- Solidity - quotient of the area and the convex area.
- Boundary - vector of boundary points.
- Mean Color and average color.
- Perimeter.

5.2.3 Objects classification

The purpose of the classification in the project is recognition of holes, pieces and their pairs for piece inserting test. As features in classification the following ones were chosen: perimeter^2 divided by filled area, solidity, eccentricity and the mean color. For classification, an ellipsoid separation algorithm was used [4].

Consider a binary classification problem in $\mathbb{R}^d$. Let X and Y be two finite subsets of $\mathbb{R}^d$ giving examples two classes. Let class X has a distribution which can be approximated by multivariate a Gaussian, let class 2 has some unknown distribution. Under this non-symmetric setting we call $x \in X$ as positive examples and $y \in Y$ as negative ones. Our assumptions which motivated the selection of classification model is as follows:

- Under limited number of training examples it is desirable to keep the complexity of the classification model as low as possible.
- Scaling of the feature space may be arbitrary: color features with area features, etc.
- The linear model is not sufficient.
- Increasing the dimension of the feature space is undesirable.

Consider problem of learning discriminative ellipsoid with center fixed at 0:

$$\text{find } A\colon \mathbb{R}^d \mapsto \mathbb{R}^d \text{ such that: } \begin{cases} ||Ax|| < 1, & x \in X\,, \\ ||Ay|| > 1, & y \in Y\,. \end{cases} \tag{5.6}$$

Constraints $||Ay|| > 1$ are non-convex in A, however we can express $||Ax||^2$ as $\langle x, A^{\mathrm{T}}Ax\rangle$ and reformulate expression (5.6) as follows:

$$\begin{array}{l} \min 1 \\ \langle x, Qx\rangle < 1, \quad x \in X \\ \langle y, Qy\rangle > 1, \quad y \in Y \\ Q \succcurlyeq 0 \end{array} \tag{5.7}$$

which contains linear constraints and $Q \succcurlyeq 0$ is equivalent to $Q = A^{\mathrm{T}}A.\langle\rangle$ denotes the scalar product.

Examples

Figure 5.4 shows objects detected in the 2D puzzle simulator environment. Feature vectors describing the objects contain color and some affine-invariant shape features.

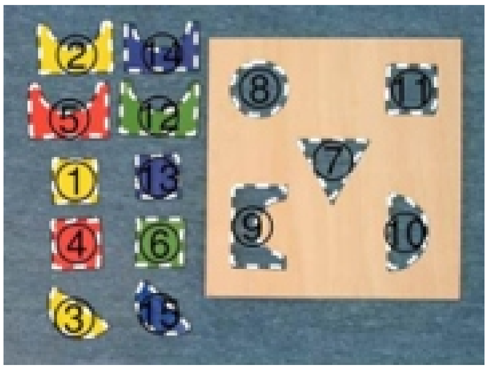

Figure 5.4: Objects detected 2D puzzle simulator.

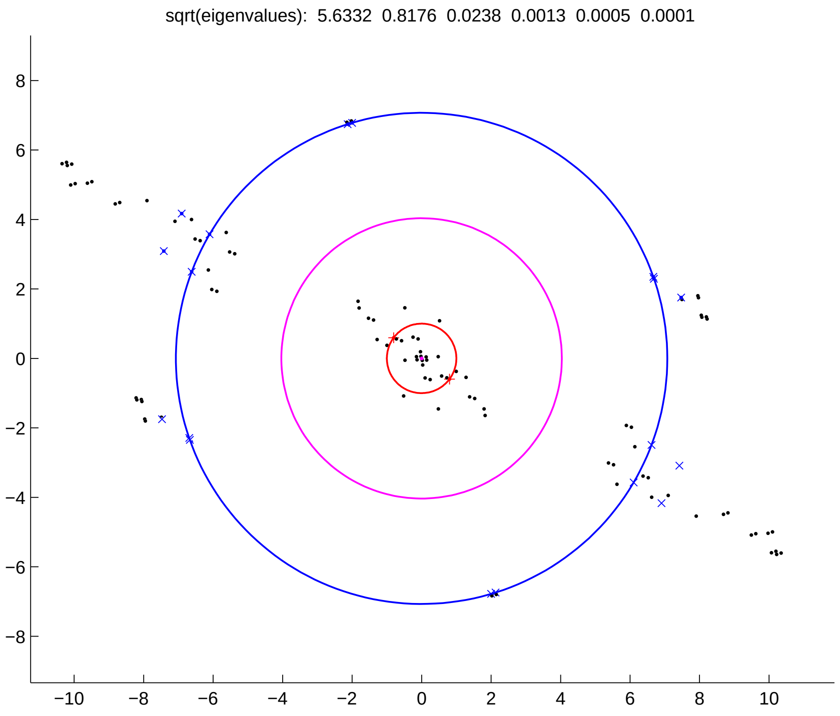

Figure 5.5: Learning “color” equivalence. Red – positive examples; blue – negative examples; black – rest of all possible pairs from Figure 5.4. Black dots inside magenta circle are indeed represent objects of equivalent color.

Figure 5.5 shows how class of equivalence can be learned based on positive training pair: $(7, 8)$ and negative training pairs $(\{7, 8\}, \{1, 5, 15, 13\})$

$(\{12, 7, 6\}, 4)$ w.r.t. Figure 5.4. The learned equivalence represents color equivalence. Note that only one positive training sample was present.

Fig. 5.6 shows learned equivalence class for training data: objects $1, 4, 11$ are said to be equal (numbers are w.r.t. Figure 5.4), negative examples were sequentially added until learned classification indeed corresponded to shape equivalence. Note that positive examples are given only for 3 square shapes.

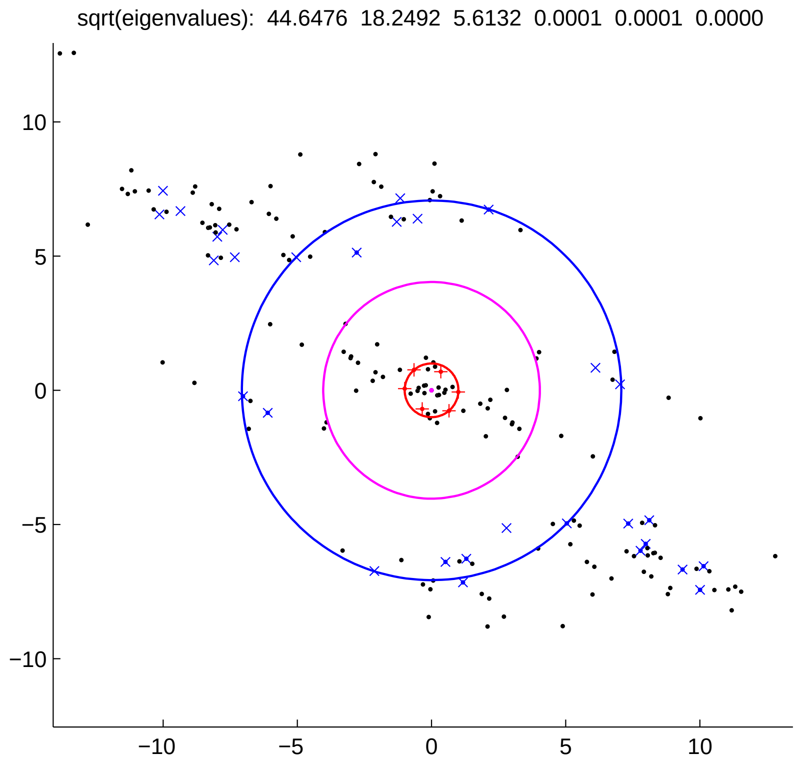

Figure 5.6: Learning "shape" equivalence: best separating ellipsoid is 3D, the projection on two main axis is shown.

5.3 Scene processing

5.3.1 Object tracing

The scene has to be processed by the system after each robot action. The robot needs recognize changes in the scene for correct work of the algorithm. The system keeps a database of the objects detected in the scene. When the image is segmented, the detected objects are compared with the stored objects. Each object in the database has its own unique index. If the object is not found then the object with a new unique index is added to the database.

Each object is characterized by a feature vector, $v = (t_1, \ldots, t_n)$, where $v_1, \ldots, v_n$ are chosen features. Similarity, S, between two objects described by vectors, $v_1 = (t_{11}, \ldots, t_{1n})$ and $v_2 = (t_{21}, \ldots, t_{2n})$, is

$$S = \|v_2 - v_1\| = \sum_{i=1}^{n} (t_{2i} - t_{1i})^2. \tag{5.8}$$

The chosen features for the comparison are the same as in classification: perimeter^2 divided by filled area, solidity, eccentricity and mean color. Separately from the feature vector the last object position in the scene is saved. Matching objects from the database with object from the scene is followed by this rules:

1. Remove superobjects (superobjects contain other objects).
2. Each object from the database can be linked with just one object in the scene.
3. An invisible object may appear only at the same position where it was visible last time.
4. The robot can change the position of just one object in the scene.
5. The value of the similarity between two objects is minimal.

5.3.2 Grip test

When the robot tries to grip an object, the system recognizes if the object was gripped or not. The system uses images from a camera for grip recognition. The system needs two images - first before gripping and second after it. Example of this images are shown in Figure 5.7. When the images are available, the gripped object is cut from the first image and this template is being found in the second one. If the object was found and its position was changed then grip is successful. For finding the template in the image, the sum of squared distances is used by template matching algorithm.

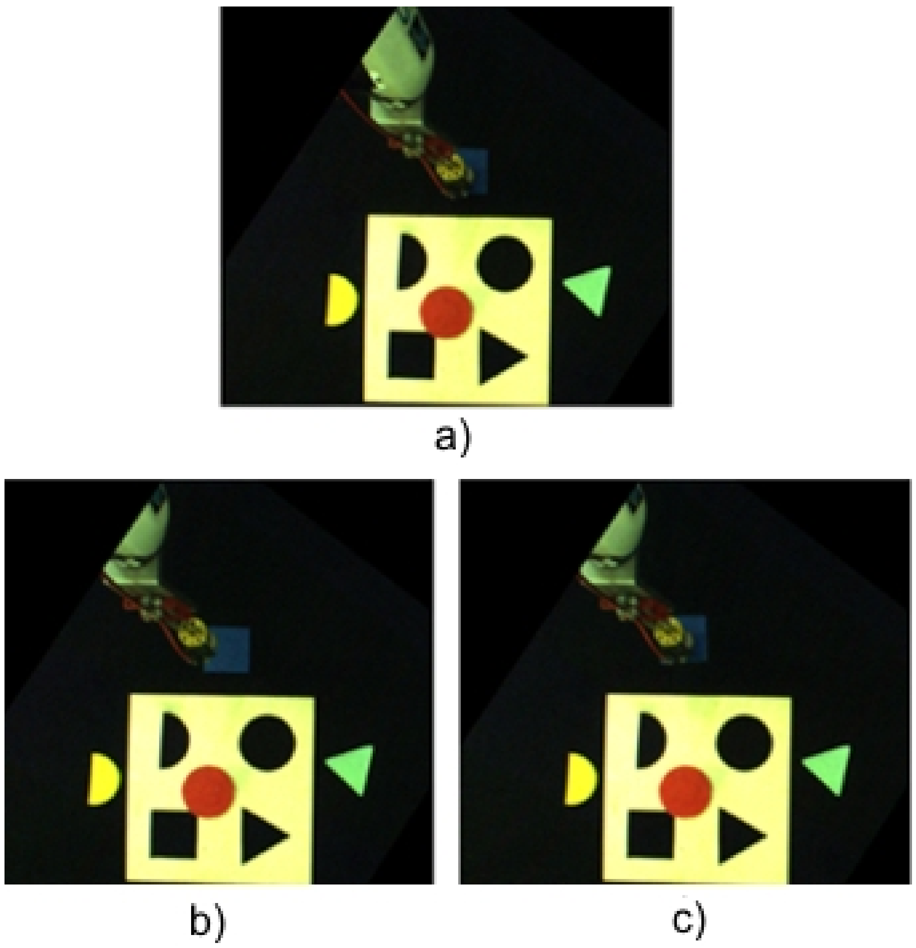

Figure 5.7: Images from camera a) before grip b) successful grip c) unsuccessful grip

Let the template T be of size $M \times N$, and the image $I(x,y)$ of size $W \times H$, where (x,y) denotes the generic image coordinates. We also indicate the image subwindow located at (x,y) as $I_s(x,y)$. Then, the generic distance function measuring the dissimilarity between the template and the image subwindow can be written as follows:

$$\delta_p(x,y) = \sum_{i=1}^{M}\sum_{j=1}^{N}(I(x+i,y+j)) - T(i,j))^2. \tag{5.9}$$

The image subwindow is the most similar with the template, where the dissimilarity is minimal.

5.3.3 Shape alignment

When the robot tries to insert a piece to a hole, the system has to align the piece with the hole automatically. Each detected object (hole/piece) has a boundary and a center of mass. This features are used for the object alignment.

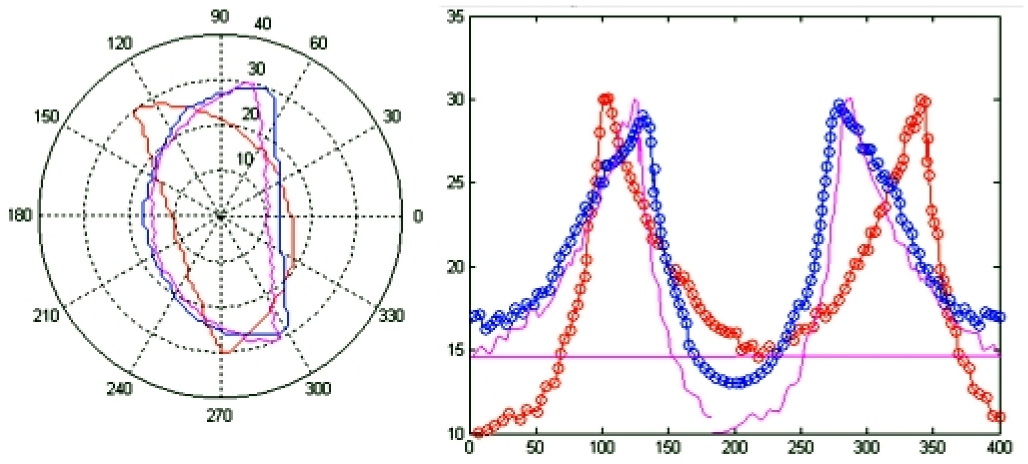

Figure 5.8: The shape alignment and the border function of two objects (red and blue line). The magenta line shows the result of the shape alignment.

For a computation of a rotation angle, we transform the border to polar coordinates, where the origin is in the center of mass. We define periodic function of the border, $f_1(\phi) = r$, for the first object and function, $f_2(\phi) = r$, for the second object, where ϕ is an angle and r is a radius used in the polar coordinates. We find the angle of rotation $\alpha \in (0, 2\pi)$, so that a value V of a function

$$V = \int_0^{2\pi} (f_1(\phi) - f_2(\phi + \alpha))^2 \mathrm{d}\phi \tag{5.10}$$

is minimal. Example of the alignment is shown in Figure 5.8.

5.4 Reinforcement learning

The example in Section 5.1.3 describes, how the robot should react to the problem. In our project, the robot decision is controlled by reinforcement learning [7]. For reinforcement learning we define:

- Set of the environment states:
  $S = (in\_gripper \times under\_gripper)$
  $=$ { Piece, Empty } $\times$ { Piece, Hole, Empty}
  $=$ {Piece-Piece, Piece-Hole, Piece-Empty, Empty-Piece, Empty-Hole, Empty-Empty}.
- Set of the actions: $A =$ {Grip, Release, move to piece, move to hole}.
- Reward function: $R\colon S \times A \to R$
- State transition function: $T\colon S \times A \to \Pi(S)$, where a member of $\Pi(S)$ is a probability distribution over the set S (i.e., it maps states to probabilities). We write $T(s, a, s')$ for the probability of making a transition from state s to state s' using action a.

  The state transition function specifies probabilistically the next state of the environment as a function of its current state and the agent's action. The reward function specifies expected instantaneous reward as a function of the current state and action.

We define the optimal value of a state – it is the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy.

$$V^*(s) = \max_\pi E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right). \tag{5.11}$$

This optimal value function is unique and can be defined as the solution to the simultaneous equations

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right), \forall s \in S, \tag{5.12}$$

which assert that the value of a state s is the expected instantaneous reward plus the expected discounted value of the next state, using the best available action. Given the optimal value function, we can specify the optimal policy as

$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right). \tag{5.13}$$

To find an optimal policy was used a simple iterative algorithm called 'policy iteration' that can be shown to converge to the correct V^* values. Using π as a complete decision policy.

```
choose an arbitrary policy $\pi'$
loop
    $\pi := \pi'$
    compute the value function of policy $\pi$:
        solve the linear equations
            $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$
    improve the policy at each state:
        $\pi'(s) = \arg\max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s')$
until $\pi = \pi'$
```

The value function of a policy is just the expected infinite discounted reward that will be gained, at each state, by executing that policy. It can be determined by solving a set of linear equations. Once we know the value of each state under the current policy, we consider whether the value could be improved by changing the first action taken. If it can we change the policy to take the new action whenever it is in that situation. This step is guaranteed to strictly improve the performance of the policy. When no improvements are possible then the policy is guaranteed to be optimal.

Chapter 6

Implementation

The project has been implemented in MATLAB environment. Team of four people worked on the implementation, because the project was very extensive. Vojtěch Franc modified the COSPAL 2D puzzle simulator, which was created by Eric Jonson from the cooperative Linköping University [6]. Pavel Janda created a hardware interface. Radim Krupička with Oleksandr Shekhovtsov used the hardware interface and modified the simulator for the real environment. This modifications are described in this work.

Whole implementation is divided into three modules, see Figure 6.1:

Hardware module includes functions of the image acquisition and the robot movement. Connection between the camera and the robot is implemented.

GM module (grounding and management) includes functions for receiving and processing the images from the camera. GM module keeps information about the scene and moves with the robot arm.

Learning module makes a decision on what the robot has to do.

Each module is described in the following sections. Sections include examples of the source code, which is modified for demonstration use. The complete source code and its documentation can be found on the enclosed CD.

6.1 Hardware implementation

The hardware implementation is a theme of Pavel Janda's diploma work [5]. The GM module processes an image acquired from the camera. Calibration

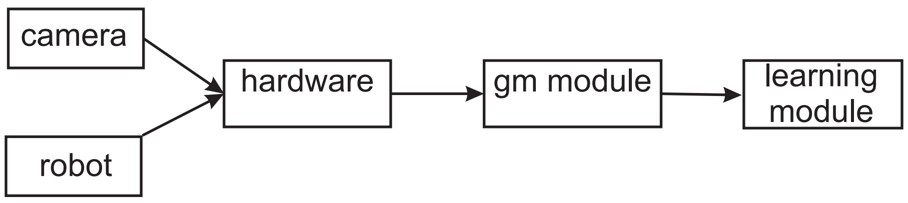

Figure 6.1: Implementation diagram.

enables to move with the robot arm in the image coordinates. There are implemented these functions: Move to the position in the image coordinates, Grip and release piece on the coordinates, Take a picture of the environment with the robot arm and without the robot arm.

6.2 GM module

The GM module is the main part of the implementation. The GM module is built on 2D COSPAL simulator, which was created by Eric Jonson [6] and later modified by Vojtěch Franc. This simulator was modified for practical use. The GM module includes functions for the robot movement and the image and scene processing.

The GM module controls robot actions and returns feedback. The actions are:

- *grip* - If the object is gripped, the function returns success.
- *move to position* - Move the robot arm on coordinates.
- *move to object* - Move the robot arm over the indexed object.
- *release* - If the object was inserted to appropriate hole, function returns 'success' and 'inserted'.

Function *gm_action* is defined in *gm_module/gm_action.m* file. More detailed description is in Section 6.2.1. *Move to object* and *grip* functions need an index of the object. The GM module contains a database of the detected objects for tracking. Each object has own unique index and own features, see Section 5.2.2. When the robot moves with a piece, the system has to take a new picture and recognize the same objects, Sections 5.3.1 and 6.2.3. The recognition needs the segmentation of the image, which is described in Sections 5.2.1 and 6.2.2.

Grip detection is described in Sections 5.3.2 and 6.2.5.

When the robot tries to insert a piece to an appropriate hole, the system automatically aligns the piece with the hole, Sections 5.3.3 and 6.2.6, and detects insertion. For insertion test, the system recognizes the pieces and its suitable holes, Sections 5.2.3 and 6.2.4. The GM module system and dependencies are shown in Figure 6.2

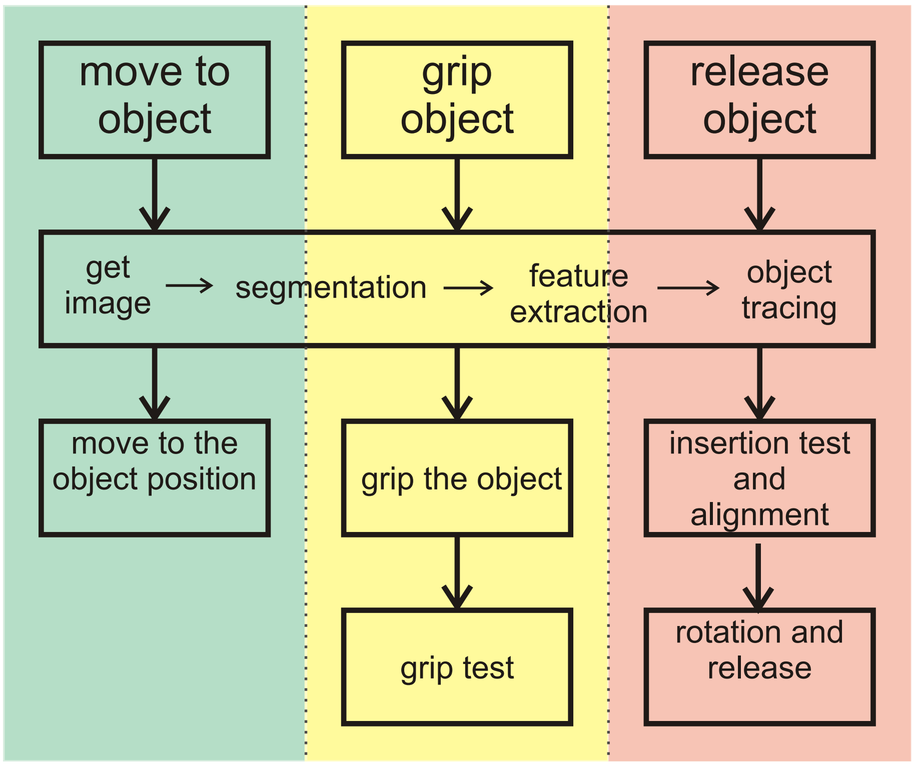

Figure 6.2: GM module workflow diagram.

6.2.1 Robot movement

The robot arm movement is implemented in the *gm_module/gm_action.m* file. There are four possible actions: grip, release, move to position, move to object.

Grip algorithm

The robot tries to grip an object.

1. If other object is in the gripper or no object is under the gripper, the grip fails,

2. else let the robot perform the action (call the appropriate hardware function).
3. If action 2 is successful then the grip is detected from the image. If the grip is successful then the object is loaded and the function returns 'success'.

Release algorithm

The robot releases a gripped object.

1. When the gripper is empty then the function returns 'fail'.
2. If an object is under the gripper then alignment angle is computed and the gripped object is aligned with the object under the gripper.
3. Let the robot perform the release action.
4. If the piece in the gripper corresponds to a hole under the gripper then the function returns successful insertion.

Move to position

Move to position calls hardware action, which moves the robot to input coordinates.

Move to object

The robot is moved to position, where the object with entered *ID* is placed.

6.2.2 Segmentation

The segmentation is written in the C++ language as a MEX module and is compiled for the MATLAB environment. The sources of the library are located in the directory *graph_segmentation* on enclosed CD. The segmentation function returns segmented objects and their features. The function has four input parameters:

- *img* - Input image for the segmentation.
- *sigma* - Smooth of the image. Values from 0 to 1, where 1 is maximum smooth.
- *k* - Constant for threshold function based on the size of the component.

- *min* - Minimum component size.

After a lot of experiments, the values were set to $sigma = 0.7$,$k = 1600$, $min = 1500$.

6.2.3 Object tracing

The function *gm_update* is called before every robot action. This function processes an image from a camera and traces objects. The function is implemented in the *gm_module/gm_update.m* file. The rules are defined in Section 5.3.1, which are respected by the following algorithm.

1. The objects (blobs) are acquired from the image by the image segmentation.

2. Convert the tracking problem to the max-sum problem [9]. Create a similarity matrix between objects from the database and detected blobs. A similarity between some blob and object is

   $$-norm(feature\_vector\_object - feature\_vector\_blob),$$

   where *norm* is the vector norm.

3. Set the similarity between blob and object to *-infinity*, when the invisible object does not appear at the same position where it was visible last time.

4. Find the combination of the blobs and the objects where the sum of their similarity is maximal. Exclude the combinations where two objects change their position. The Vojtěch Franc's function *maxsumg* was used for the computation.

6.2.4 Object detection

The object detection is used for holes and pieces recognition. Several examples of the images of the holes and the pieces were chosen and the classifier was learned and saved for future recognition. Learning equivalence class in linear space is based on “positive” and “negative” examples. Positive examples are pairs of points which have to be classified as equal, negative examples are pairs of points which have to be classified as not equal. The classifier is searched in the class of separating ellipsoids. The separating ellipsoid with center at 0 is found such that positive examples are inside unit sphere. Negative examples are outside the sphere with maximal radius, where spheres are considered in the metric defined by the ellipsoid. Learning is implemented in the function *gm_ module/classifier/learn_ eq_ classifier.m*.

When the classifier is learned, the system could classify the image of the object as the hole or the piece. The classifier class is found in the */gm_ module/classifier* directory. Classifier class was implemented by Alexander Shekhovstov. Yalmip and SeDumi extension are used for classification computation.

6.2.5 Grip test

The function *detect_ grip* is implemented in the *gm_ module/ private/ detect_ grip.m* file. This function takes four parameters: object ID, image of the scene with the gripper on the object before lift up - I_1, image after grip - I_2 and the image of the scene without the robot arm before grip. The function cuts off the object from I_1 and calls a function *detect_ object*, which finds object from I_1 in the I_2. If the object from I_1 changes the position, the grip was successful.

6.2.6 Alignment

The function *shape_ align_ bd* computes the rotation angle to align two shapes. This function is implemented in the *gm_ module/private/shape_ align_ bd.m* file. The function takes a border and a centroid of the objects and returns the rotation angle.

1. Transform border coordinates from Cartesian to polar coordinate system.

2. Create the border function. Make a cubic interpolation for undefined points.

3. Find the shift of the functions, where the sum (5.10) is minimal.

6.3 Reinforcement learning

The pilot problem is to insert all the pieces into the appropriate holes. This problem is solved by the reinforcement learning. The reinforcement learning algorithm was described in Section 5.4. The algorithm is implemented in the *rl* directory and the script *rl/rl.m* starts the demonstration.

At first, the start script *rl.m* initializes hardware and then loads the saved classificators. If the problem was solved before the previously learned model is loaded. The main part of the algorithm is the function *policy_iteration*, which computes optimal policy for the given model. This function calls the *policy_evaluation* function for computing values of states. The *policy_iteration* returns the action, which the robot has to execute now. A function *rl/make_action.m* makes this action. The *make_action* function calls *gm_action.m* (from GM module) which causes the robot to move. If the action was successful then the robot is rewarded. The reward is 5 for an insertion of a piece into its hole and 1 for a successful grip.

Chapter 7

Conclusions

The aim of the diploma project is to contribute to the studies in the cognitive systems. The task of solving the "shape sorter puzzle" was chosen as a demonstrator of the cognitive systems. The experimental environment was created and used for solving this puzzle. The diploma project is a part of the EU project COSPAL. The diploma project is designed so that it is compatible with the other COSPAL projects and follows the demands of the COSPAL architecture.

Vojtěch Franc modified the puzzle simulator from the cooperative Linköping University and created the core of the GM (grounding and management) module. Pavel Janda [5] prepared low-level modules for the robot control and the image acquisition. We included these modules to the GM module and tested the functionality of the robot movement and the camera. The GM module was enhanced with the object alignment, the object tracking, the object recognition (classification) and the grip detection. After the GM module was tested, I found out that the original image segmentation was unsuitable and implemented another segmentation algorithm.

The reinforcement-learning algorithm was used to determine the decision strategy. The algorithm has been successfully verified on the pilot task and is suitable for solution of the cognitive tasks. The implementation and the whole system was successfully defended on the COSPAL workshop in September 2006. At this moment, the robot is used for another tasks. The diploma project provided also another outcome. The developed modules are used in a tracking process. The reinforcement learning in combination with the vision and the robot manipulation has been used in student laboratory exercises at the Czech Technical University, Faculty of Electrical Engineering, Department of Cybernetics.

Bibliography

[1] D. A. Berry and B. Fristedt. Bandit problems: Sequential allocation of experiments. Chapman and Hall, London, UK, 1985.

[2] COSPAL. *Cognitive Systems using Perception-Action Learning*, number COSPAL-004176, June 2005.

[3] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, pages 167–181, September 2004.

[4] Fr. Glineur. Pattern separation via ellipsoids and conic programming. Technical report, Faculté Polytechnique de Mons, Belgium, 1998.

[5] Pavel Janda. Control of the industrial robot with cameras. Master's thesis, Charles University in Prague, 2007.

[6] Erik Jonsson. The COSPAL 2D puzzle simulator v1.1, October 2004.

[7] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W.Moore. Reinforecement learning. *Journal of Artificial Intelligence Research 4*, pages 237–285, June 1996.

[8] LiU. COSPAL deliverable D1: Project presentation. Technical Report IST-2003-004176, LiU, 2004.

[9] Milan Šonka, Václav Hlaváč, and Roger Boyle. *Image Processing, Analysis and Machine Vision.* Thomson Toronto Canada, third edition, 2007.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press Cambridge, Massachusetts London, England, 1997.

[11] Tomáš Werner, Václav Hlaváč, and Vojtěch Franc. COSPAL Deliverable 8: Network States to Symbols Software Prototype. Technical Report

CTU–CMP–2005–24, Czech Technical University in Prague, September 2005.

[12] www.wikipedia.org. Cognitive architectures. 2006.

List of Figures

Appendix A

The Contents of the Enclosed CD

A.1 Overview of the contents of the enclosed CD

The enclosed CD contains:

- System guide and code documentation.
- PDF file containing this diploma thesis.
- Project source codes and compiled binaries.

The contents of the individual directories:

gm_module – The main control module of the application is here.

graph_segmentation – This directory contains the segmentation algorithm written in “C” language.

hw_ctu – The contents of this folder present both the robot control and the image acquisition modules.

rl – The source codes of the reinforcement learning modules are stored here.

robot – This directory contains the source codes, documentation and compiled DLL libraries of both the TROL Robot Control Library and MATLAB Interface for CRS A465 Robot.

SEDUMI, stptool and yalmip – These folders incorporate external libraries required for the work.

A.2 The list of the created and modified files

The integral component of this diploma project are the files containing the source codes. The works on this project required the creation of the new files as well as the modification of the current ones. The enclosed CD contains some functional modules of the COSPAL project. A group of people has been engaged in the COSPAL project development, and thus the CD contains the works of many authors. All files referring to this diploma thesis contain the author's mark *RK*. The following list contains both newly created and modified items. Only the most important files are listed:

- */rl/* – The reinforcement learning algorithm.
  I created or modified all files in this directory.
- */graph_segmentation/* – The graph image segmentation algorithm.
  I created or modified all files in this directory.
- */gm_module/eq_shape.m*
- */gm_module/gm_action.m*
- */gm_module/gm_can_insert.m*
- */gm_module/gm_update.m*
- */gm_module/private/detect_grip.m*
- */gm_module/private/detect_object.m*
- */gm_module/private/get_blobs_gr.m*
- */gm_module/private/gm_auto_align.m*
- */gm_module/private/shape_align_bd.m*
- */gm_module/private/shape_align.m*
- */gm_module/private/template_matching.m*