Czech Technical University in Prague
Faculty of Electrical Engineering

Diploma thesis

Software for Robot Platform

Prague, 2011 Martin Dubec

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Martin D u b e c
Study programme:	Cybernetics and Robotics
Specialisation:	Robotics
Title of Diploma Thesis:	Software for Robot Platform

Guidelines:

1. Study the robot Mitsubishi RV6SDL.
2. Study the planned function of robotic platform in the project MASH.
3. Design software for remote robot control, automatic distribution and resetting of manipulated objects.
4. Implement the designed software and perform experiments with it.
5. Document the implemented software.

Bibliography/Sources: Will be provided by the supervisor.

Diploma Thesis Supervisor: Ing. Vladimír Smutný

Valid until: the end of the summer semester of academic year 2011/2012

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.
-4-

[illegible]

prof. Ing. Boris Šimák, CSc.
Dean

Prague, January 27, 2011

Prehlásenie

Prehlasujem, že som svoju diplomovú prácu vypracoval samostatne a používal som iba podklady (literaturu, projekty, SW atď.) uvedené v priloženom zozname.

Nemám závažný dôvod proti použitiu tohoto školského diela v zmysle ß 60 Zákona č.121/2000 Sb. , o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon).

V Prahe dňa ____________ ________________

podpis

Acknowledgement

I would like to convey my graditude to my supervisor Ing. Vladimír Smutný, who was always willing to consult any problem that occured and who created perfect conditions for elaborating this work.

Abstract

The aim of this master thesis is to create a workplace for experiments with robotic arm. The workplace should be controlled remotely, support changing environment to reliable simulate of the real world and return perceptions from environment. It is designed to be a part of the european project MASH as application server to perform goal-planing tasks.

The thesis is focused on description of the software and hardware components. There are solved solutions to integration to MASH platform, initialization equipments (robot, gripper, TV set), preparing/cleaning playground and processing goal-planing experiments (capturing images from cameras, computing reward, moving robot).

Anotácia

Cieľom tejto práce je vytvorenie pracoviska pre experimenty s robotickou buňkou. Pracovisko by malo byt ovládane na diaľku, má podporovať zmenu prostredia pre hodnôvernú simuláciu reálneho sveta a výstupom má byť obrazové vnímanie prostredia. Platforma je navrhnutá tak, aby bola súčasťou európskeho projektu MASH ako aplikačný server pre uskutočnenie plánovacích úloh.

Práca je zameraná na popis softvérových a hardvérových komponentov. V práci sú riešené riešenia pre integráciu do platformy MASH, inicializácia zariadení (robot, chapadlo, TV), príprava / upratovanie ihriska a spracovanie plánovacích úloh (zachytenie obrazov z kamier, výpočet odmeny ťahu, pohyb robota).

Contents

List of Figures

List of Tables

List of Abbreviations

2D	two dimensional
MAS	MASH application server
MASH	Massive sets of heuristics
MS	Microsoft
RAS	Robot application server

Chapter 1

Introduction

1.1 Motivation

Entrance of robots to the industry scene is a major milestone of automation and industrial. Most advanced component of automated systems in today's serial production are just robots. Robot is denoted as a stationary or mobile machine that performs the tasks according to pre-defined plan.

The robot platform is a part of the coordinated effort in the MASH project. It acts as physical embodiments of the general machine, which demonstrates the capabilities of computer vision, pattern recognition, machine learning, and goal planning algorithms implemented during the project.

1.2 The goal of MASH project

The project MASH, european project focused on machine learning, where the performance of ML techniques are assed on various task (image classification, goal-planning). This project is funded by the Information and Communication Technologies division of the European Commission, Cognitive Systems and Robotics unit, under the 7th Research Framework Program.

The main goal of the MASH project is to develop tools for the design of very large and complex learning systems. Large set of feature extractors is needed to create such system. The performance of this strategy will be assessed on multiple goal-planning tasks, either in a simulation environment, or with a real robotic arm. There is presented the robotic cell and the description of the of tasks for the real environment.

The robot platform operates over the simple world composed from objects lying on the playground. The robot can be operated and controlled remotely.

Main task of robot platform is to perform action (robot movement) on selected environment which is represented by objects placed on the playground with specified background, with chosen goal (ex. reach the read object on the playground) and return reward of this action and images from cameras providing perception.

1.3 Thesis structure

- **Chap. 1 - Introduction** - introduction to the issue, description of the MASH project
- **Chap. 2 - Robot platform** - definition of the terms used in the robot platform, design of the robotic tasks, goals and environments
- **Chap. 3 - Setup of the robotic cell** - description of the components and devices
- **Chap. 4 - Software architecture and structure** - designed software solution
- **Chap. 5 - Support modules** - description of support modules used in platform (camera calibration, object store area, object manipulation, etc.)
- **Chap. 6 - Conclusion** - summary of objectives of the work, proposal of the possible extensions
- **App. A - Toolboxes** - description of the implemented toolboxes
- **App. B - MASH Application Server Protocol** - description of protocol
- **App. C - Device setup parameters** - settings of the devices (camera, robot, gripper, television)
- **App. D - Object store area** - plan of the object store area

Chapter 2

Robot platform

2.1 Integration to MASH project

The robot is a real robot arm, which interacts through software daemon with the rest of the MASH platform with an extension of the application server protocol, described in App. B.

The robot platform integrates a physically real robot to perform actions and the set of cameras for a perception. Each task is defined by an environment and a goal. The environment comprises a geometrical configuration of playground, placements of objects, background on TV, etc. They are described in Sec. 2.4.3. The goals for this tasks are defined in Sec. 2.4.4, and composed of a reaching certain object. From that goal derives a reward, negative when the robot moves out the environment, positive when the goal is reached and positive otherwise. This document gives a description of the tasks sufficient for understanding their purpose and the challenges they induce.

The software part of robotic platform is a software daemon which implements an extension of the MASH Application Server protocol for interactive applications. This extension is described in App. B. This software part is called Robot Application server (RAS).

This daemon perform the actions with the robot on a playground and the physical interactions it has with it. The robot platform is composed of three cameras to grab images for a perception.

2.1.1 MASH Platform

The MASH platform as depicted on Fig. 2.1 is composed of the following Servers:

- Web Server: Standard HTTP server hosting the website used to to submit new heuristics or schedule experiments.
- Experiment Scheduler. This daemon dispatches the experiments scheduled by the project participants to the Experiment servers. There is a single instance of it.
- Experiment Server. It runs the heavy computations for the machine-learning experiments. The settings of the experiments is sent by the Experiment scheduler, and the data or goal-planning tasks are provided by the application servers.
- Application Servers. It provides access to either the data sets or the planning problems. Their functionalities depend on the task it implements:
    - Image Server: provides access to image databases.
    - 3D Simulation Server: allows the control of an avatar in a 3D virtual world.
    - Robotic Arm Server: allows the control of a robotic arm. This server is implemented by CTU.

2.2 Task Design and Robot Motion Strategy

The children toy blocks with the size (height, side, diameter) 33 mm and basic colors are used. There are placed on glass covering standard TV screen, which offers the possibility to change the background image. The blocks are placed in the single layer in the first series of tasks. The blocks are grasped by the robot manipulator with the two fingers gripper. The robot fingers are always placed left and right from the grasped block, so there are corresponding restrictions on the blocks placement, which is important especially in random and user built environments.

The robotic cell supports three main activities: blocks placement, blocks removal (reset), and standard motion controlled by experiment server.

During standard motion, the robot fingers move ca 65 mm above the blocks in the plane parallel to the plane of the supporting glass. This allows free motion without the collision with the blocks. All cameras are focused to the upper side of the blocks.

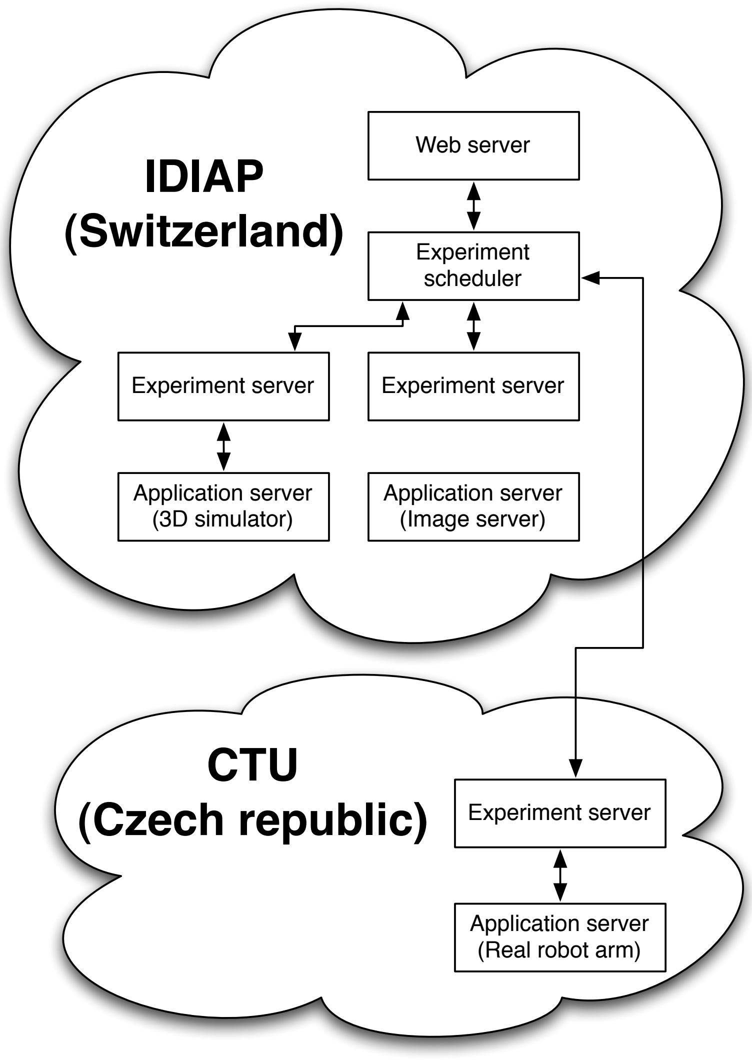

Figure 2.1: Components of the MASH platform.

The region, where gripper moves during standard motion, is located within the playground region, which is rectangle within intersection of top and oblique cameras field of view, TV screen, and robot working space. This setup allows to guarantee both visibility of blocks in all cameras and placement of blocks by the robot.

The center of the gripper moves in the regular grid given by lattice coordinates expressed as whole numbers. The centers of blocks are located in the same grid.

The goal is reached, when the gripper is just above the target cube.

2.3 Lifecycle of the experiment

The common experiment has following lifecycle:

- **Task definition** The Experiment server selects the goal and competent environment from the list.
- **Task setup** Initialization of devices (open communication with robot, gripper, TV, turn on the lights). The robot deploy objects from object store into the playground according the positions defined in the environment. Then the robot moved to the start position as defined in the goal.
- **Experiment loop** The Experiment server is sending action commands to perform robot movement e.g. UP, DOWN, RIGHT, LEFT. After each command the images captured by the cameras are sent to the Experiment server. The Application server also sends information about the current situation in the field as a reward.
- **Task finished** When the experiment is finished, the robot cleans the objects back to the object store to original positions are on the same positions as were dragged. Then are closed all open communications and turn off lights. The robot platform is waiting for a new experiment.

2.4 Robot platform Application server interface

Experiment and Application server communicate in MAS protocol described in App. B. MAS protocol uses following terms:

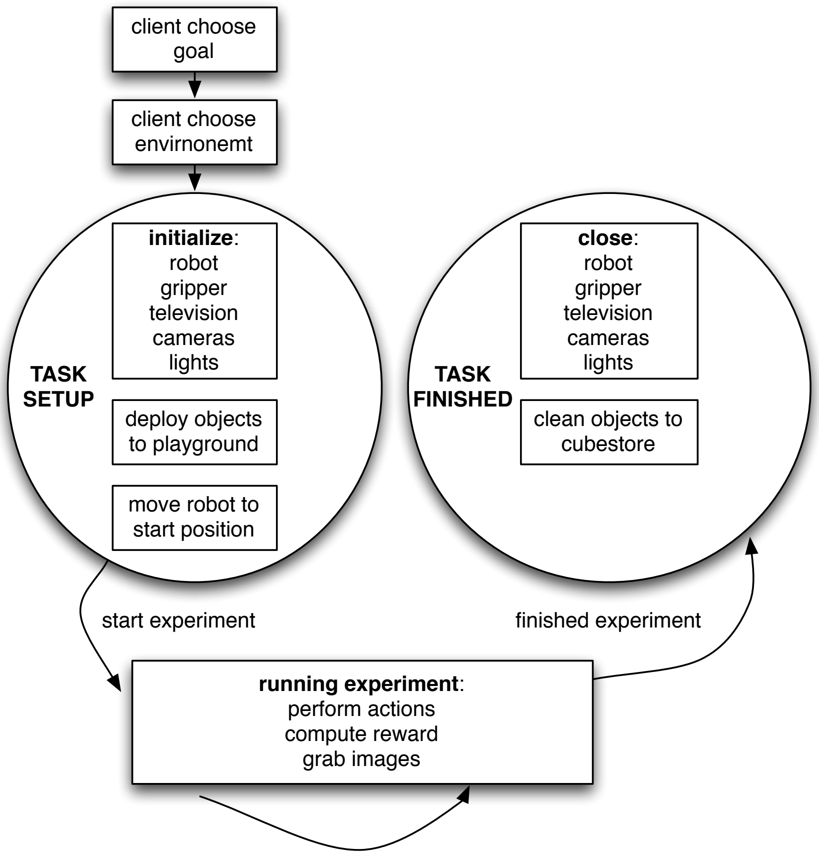
client choose goal
client choose envirnonemt
TASK SETUP
initialize: robot gripper television cameras lights
deploy objects to playground
move robot to start position
TASK FINISHED
close: robot gripper television cameras lights
clean objects to cubestore
start experiment
finished experiment
running experiment: perform actions compute reward grab images

Figure 2.2: Scenario of the experiment.

- Task
- Environment
- Goal
- Action
- View

The semantics of those terms in the context of the robotic platform is defined in following section.

2.4.1 Task

Task is specified by *goal* and the appropriate *environment*. The environment comprises a geometrical configuration, placements of blocks, etc. all defined with some random option to ensure an infinite variety of the setup. From that goal derives a reward, negative when the robot gripper collides with the environment and positive otherwise. When the goal is reached, the task is over, and another round can be started.

2.4.2 World

World is a two dimensional space where robot can move. It is a subset of the MASH playground. The world is defined as a rectangle. Dimensions and position of the *world* depends on the environment. *World* uses its own coordinates called lattice coordinates measured in lattice units (currently one lattice unit is 20 mm).

2.4.3 Environments

Each task comprises a 2D environment, which can be of one the following types, as pictured on Fig. 2.4.

The currently implemented environments in robot application server:

- `STATIC_12x12` - world size is $[12 \times 12]$, environment contains 3 blocks,
- `STATIC_8x6` - dimensions of the world are $[8 \times 6]$, environment contains 3 blocks,
- `STATIC_4x4` - dimensions of the world are $[4 \times 4]$, environment contains 2 blocks,

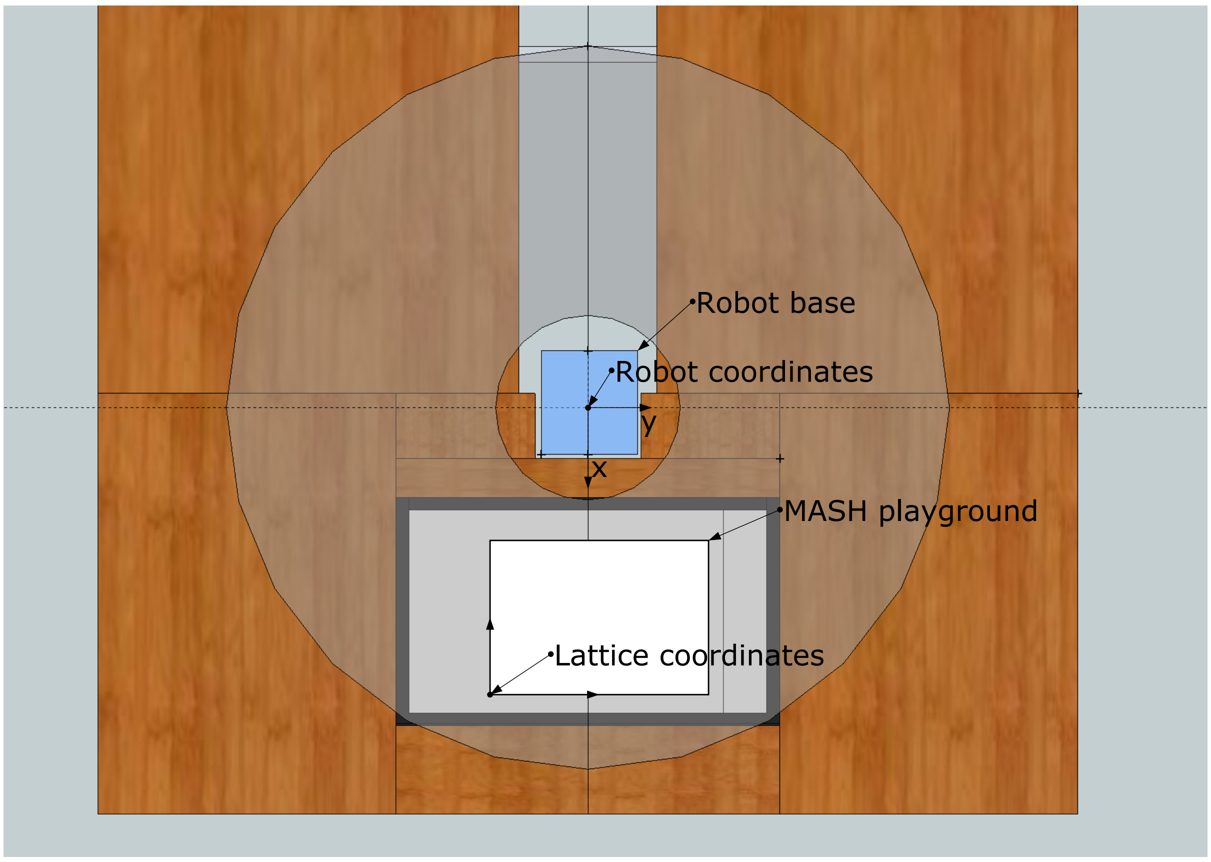

Figure 2.3: Coordinates of the robot (measured in millimeters) and lattice coordinates (measured in lattice units).

- `RANDOM_12x12` - world size is $[12 \times 12]$, block positions and types are random.

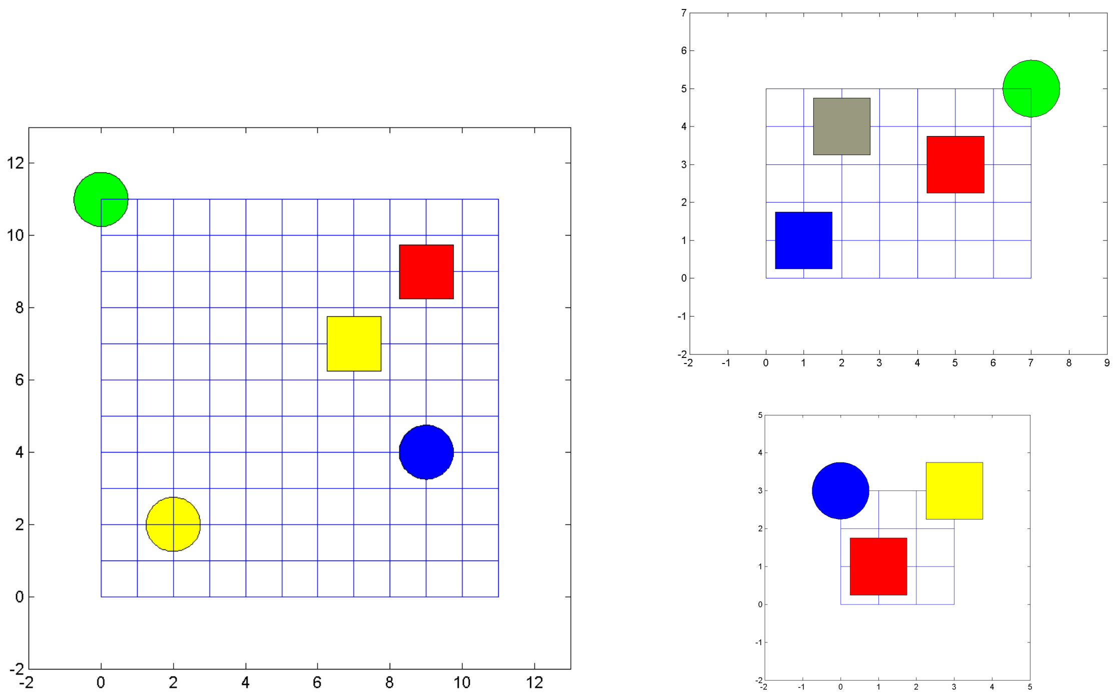

Figure 2.4: The environments `STATIC_12x12` (left), `STATIC_8x6` (upper right), `STATIC_4x4` (lower right).

In all these configurations, the size of the environment is from $[4 \times 4]$ to $[12 \times 12]$ centimeters to ensure variability between different experiments.

Environment contains informations about available views, actions, goals. *Environment* specifies properties which are described in the MAS protocol:

- **unique name** - unique identifier of environment. The environment is chosen by its name during initialization.
- **supported goals** - list of goals, which are supported by appropriate environment. Currently both goals are supported by all environments.
- **supported actions** - list of actions, which are supported by the environment. Currently all actions are supported by all environments.
- **supported teaching** - true if teaching is supported in the environment. Currently teaching is not supported.
- **supported views** - list of cameras to obtain images of the playground. Currently all environments support all views.

Properties of the environment specific to robot applications server:

- **tv image** - image which will be displayed on the TV screen.
- **border** - specifies dimensions and position of the *world* in the lattice coordinates. Border is an array of points in lattice coordinates in order lower-left corner, lower-right, upper-right, upper-left,
- **blocks** - initial positions and types of blocks specified by block type and its position $[x, y, \phi]$. Blocks could currently be only aligned only to the lattice axes ($\phi = 0$).

The list of currently available TV images: `black, white, mash, water, forest.` The list of available blocks:

- Cylinders: red, green, yellow, blue.
- Boxes (right rectangular, base size 33x100 mm): red, green.
- Cubes: red, green, yellow, blue (two pieces each), unpainted wood (four pieces).

2.4.4 Goals

A goal object is present in the task, and it is reached when the robot tool moves to the position just above them. The reward is then +20.

Goal defines properties:

- start position - start position of the robot gripper in lattice coordinates after initialization of the *task*,
- end position - position of the target block in lattice coordinates,
- target block - type of the target block.

We have defined two different goals, all involving reaching one object. Currently implemented goals in robot application server:

- `ReachRedCube` - goal is to find a red block in the world. Position of the target block is always in upper-right corner of the world. The reward is +20.
- `ReachRedCube_Random` - same as goal `ReachRedCube` but only position of the block is random. The reward is +20.

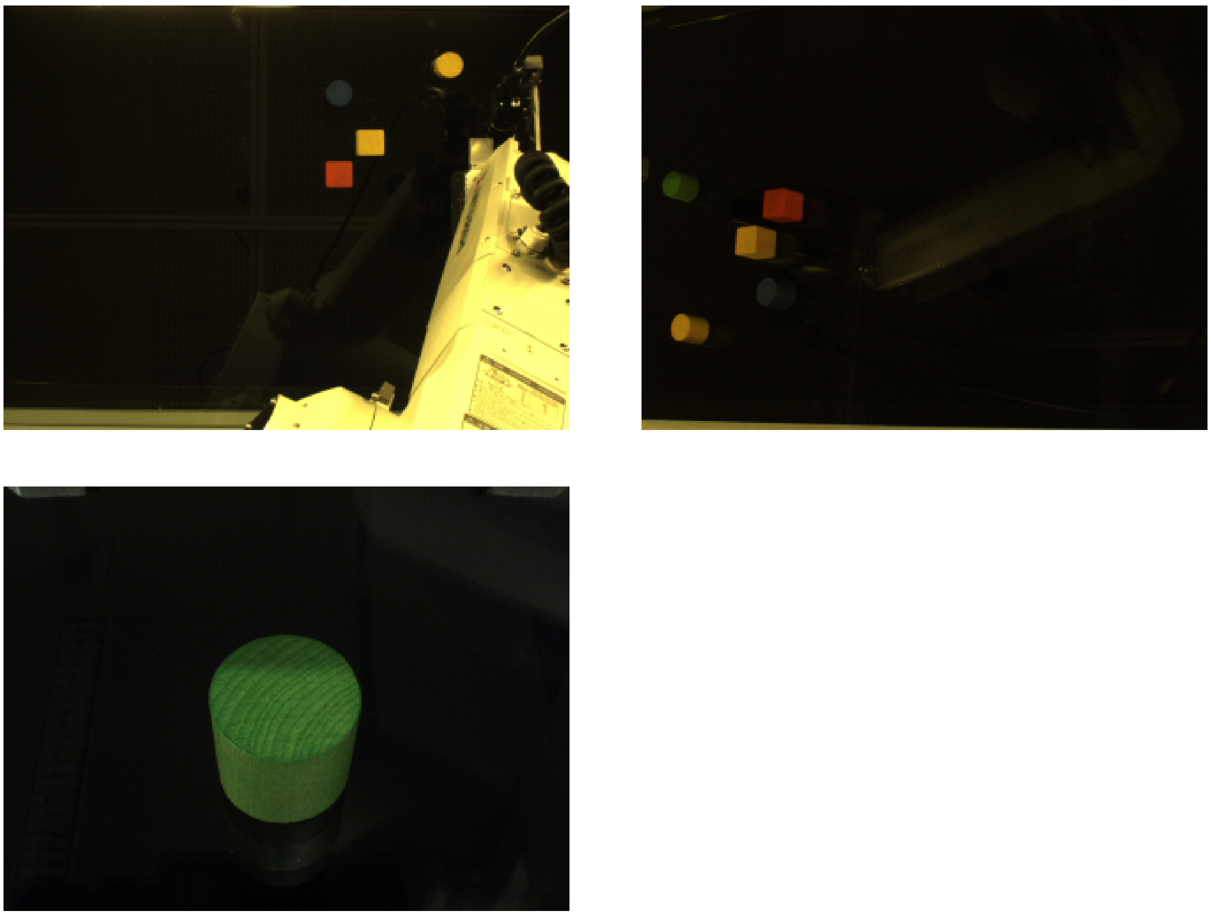

Figure 2.5: The images of the environment from the top camera (left), from the oblique camera (right), and from the gripper camera (bottom). The images are examples of the images captured during learning/planning phase. The robot arm is at position (0,11), that is just above green cylinder. The robot arm typically obstructs part of the top camera image, sometimes is obstructing oblique view as well.

2.4.5 Actions

All tasks have four actions to move forward, backward, left and right. Action moves the robot tool 20 mm in the chosen direction to the neighboring cell in the lattice coordinates. Robot is able to move in four directions :

- `UP`
- `DOWN`
- `LEFT`
- `RIGHT`

A negative reward of -10 is produced if the robot tool collides with a wall.
Reward is computed after action:

- -10 robot hit the wall,
- 0 robot performed allowed move,
- +20 robot reached the target.

2.4.6 View

Currently implemented views:

- `CAMERA_TOP` - top camera, resolution: 1280x960, output format: MIF
- `CAMERA_OBLIQUE` - oblique camera, resolution: 1280x960, output format: MIF
- `CAMERA_GRIPPER` - gripper camera, resolution: 1280x960, output format: MIF

These goals, while simple, already require the learning system to implement some crude form of object detection and localization (objects, robot).

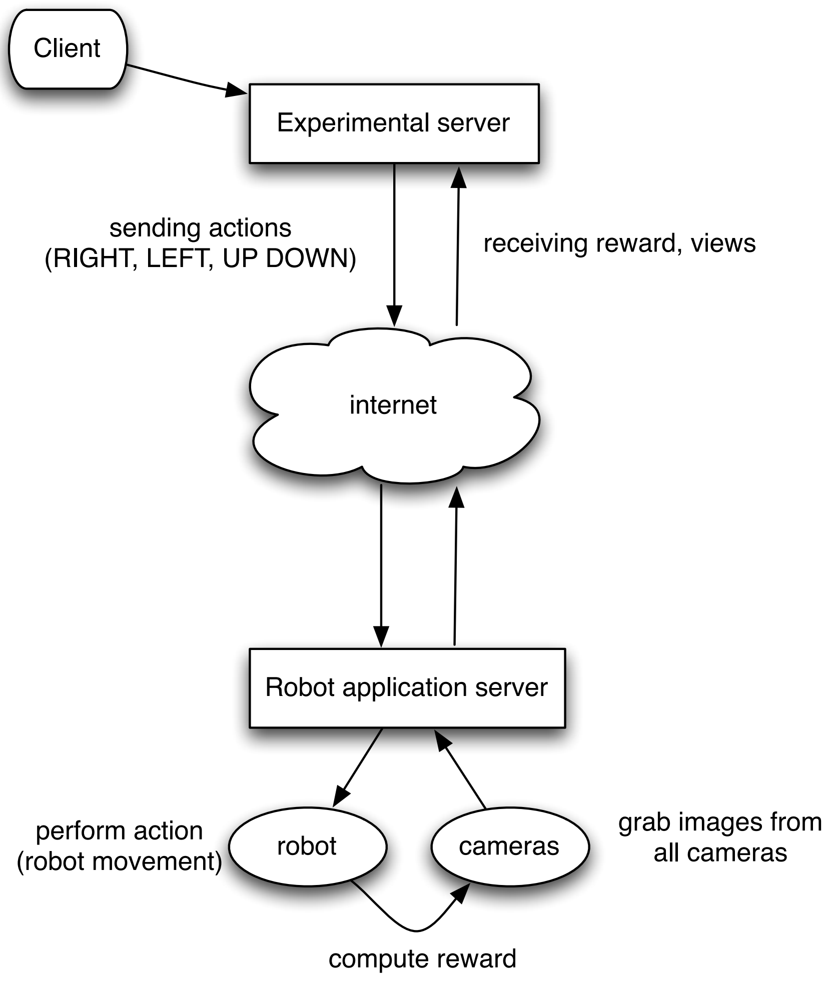

Figure 2.6: Simplified view to the process of the experiment cycle.

Chapter 3

Setup of the robotic cell

Robotic cell is composed of several components. Major role in the platform has the robot with the gripper, which is used to perform the experiments and to prepare the playground. There are also used three cameras, lights, TV set.

3.1 Setup description

3.1.1 Robot

The robot Mitsubishi Melfa RV-6SDL (Fig. 3.2) is 6 degree of freedom angular manipulator. It is equipped with a magnet to manipulate ferromagnetic blocks and a gripper (Sec. 3.1.3). The maximum weight of the blocks is about 1 kg. The robot manipulates the blocks lying on the horizontal table around the robot.

3.1.2 Control unit

Robot is controlled by the control unit CR2D-711 is with teaching pendant R32TB. The control unit is programmed in language Melfa Basic V. Mitsubishi Melfa Toolbox [4] is used to to control the robot.

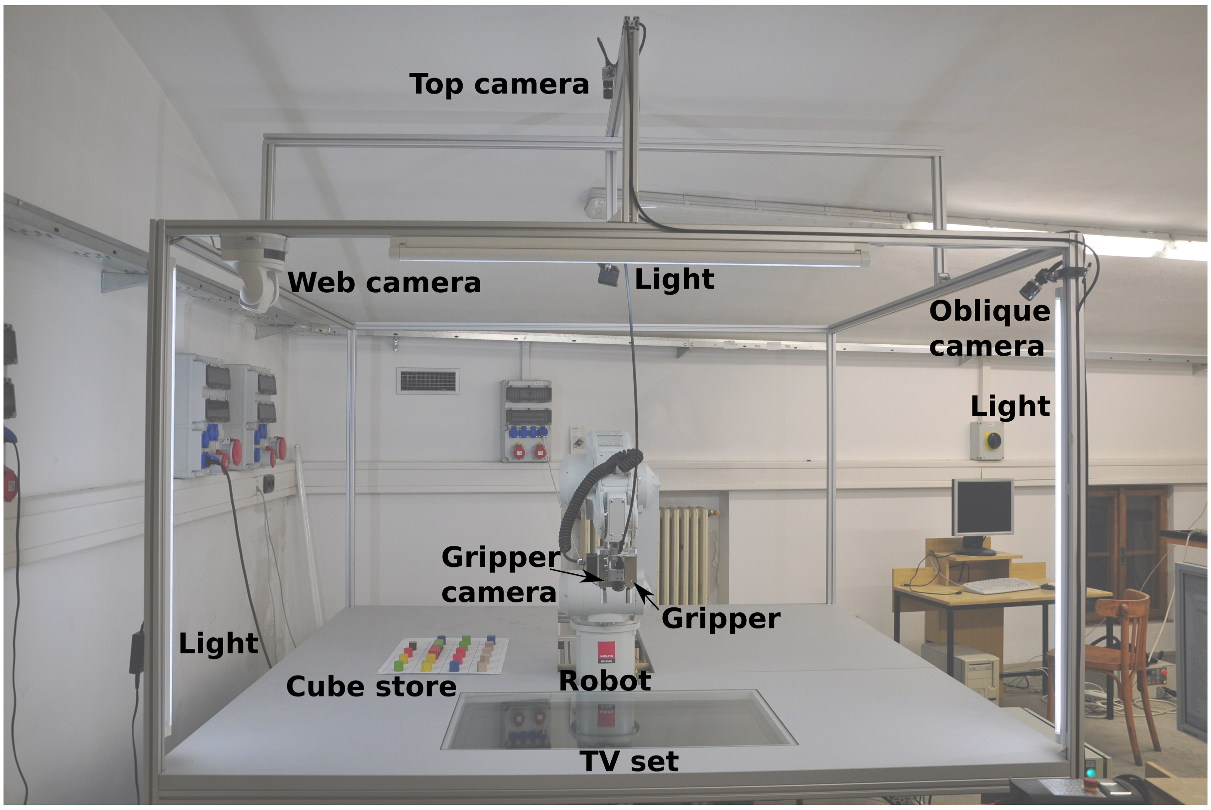

Figure 3.1: Photo of the robot platform. The first the camera above the playground. The second camera is on the right pillar capturing the oblique view. The third camera is mounted on the gripper capturing view below the gripper. The pan and tilt web camera is located on the upper beam of the cell construction. Lights are mounted on the left, right and top pillar. The TV screen is located bellow the playground behind the a protecting glass.

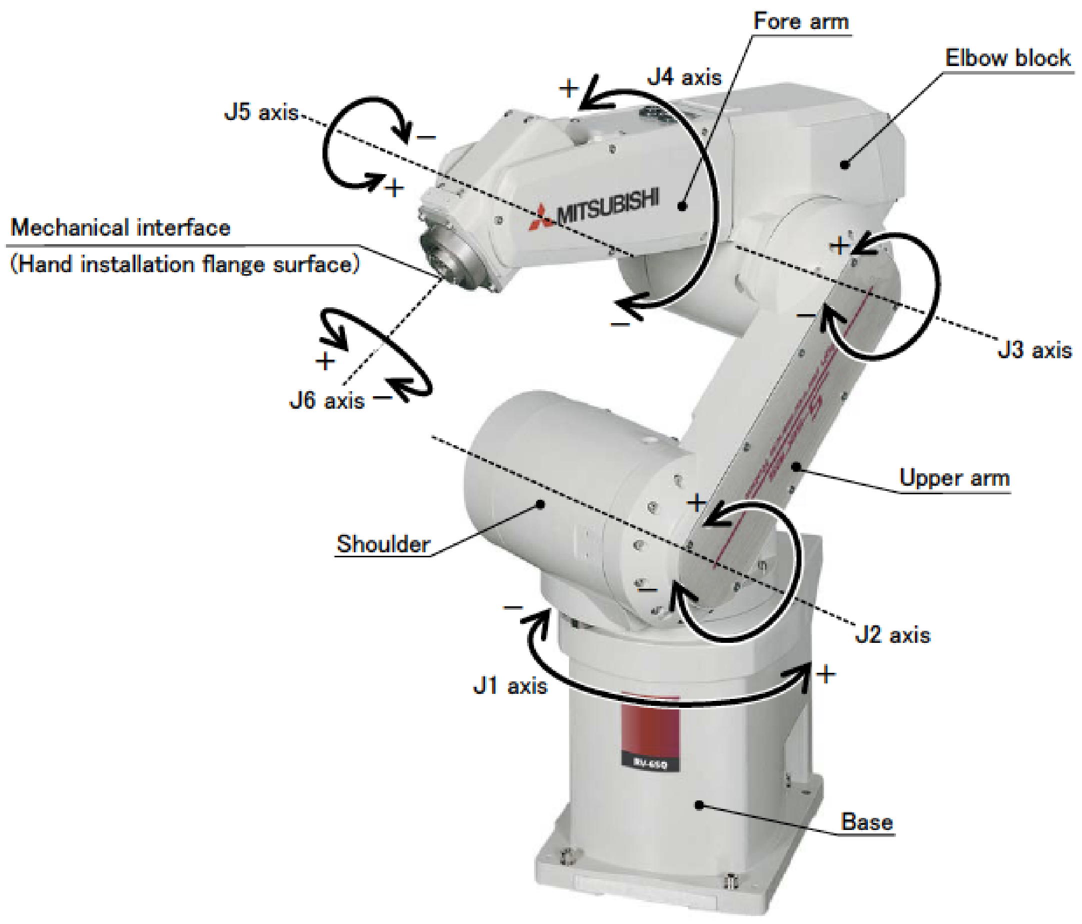

Figure 3.2: The robot Mitsubishi Melfa RV-6SDL

3.1.3 Gripper

The gripper is universal, flexible, servo-electric 2-finger parallel gripper with gripping force control and stroke 70 mm. The gripper is operating through Gripper Toolbox described in App. A.2.

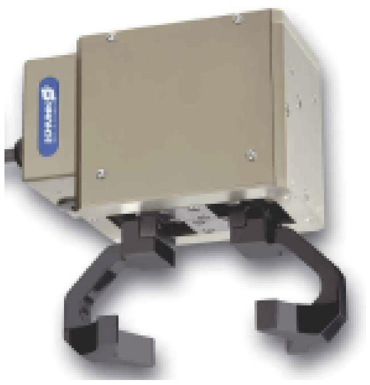

Figure 3.3: Gripper SCHUNK PG-70

3.1.4 Cameras

Robotic platform contains three color CCD cameras PointGrey FLEA2 FL2G-13S2C with maximum resolution 1288x964. The camera is controlled from the computer by using Camera Toolbox described in App. A.4.

Figure 3.4: Camera PointGrey FLEA2

Cameras layout is seen in Fig. 3.1.

- First camera is placed just above playground to capture approximately top view of the playground.
- Second camera is capturing oblique view of the playground.

- Third camera is mounted on the gripper to view below them.

Default cameras settings are described in App. C.4.

3.1.5 Web camera

The web pan and tilt camera AXIS PTZ 211 allows to check the situation of the robotic cell independently to Experiment and Application server infrastructure.

The access to the camera is allowed to people experimenting with it.

3.1.6 MASH playground

The region of the table in front of the robot is a playground, where MASH algorithms operate. The playground is a physical space satisfying following criteria:

- **Background** The TV screen located under the playground displays different images which effectively make a controlled background of image processing algorithms. The displayed images are controlled by the Experiment server. The controlled background also simplifies cleaning procedure when experiment is reset.
- **Robot manipulated** The robot has to be able to reach and manipulate the objects located within the playground.
- **Camera observed** The images of playground scene are captured by cameras located in the robotic cell.
- **Easy to describe** To simplify the description of playground area, the playground is a rectangle whose axis are aligned with the robot base axes.

The back side of the robot table serves as a store of the blocks.

The rectangular area of the table in front of robot (see Fig. 3.5) is dedicated to be a MASH playground. This area is divided logically into the lattice of points, where the blocks could be placed. This logical division is used to allow simplified description of the task and the planing algorithm. The blocks are supposed to be in a single layer. Both restrictions are on the SW layer and could be released in future set of tasks if that proves necessary or useful.

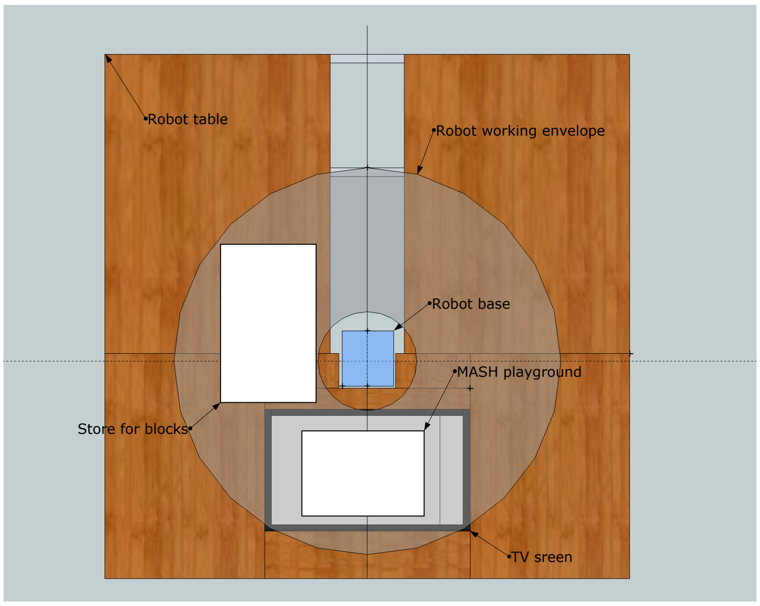

Figure 3.5: Top view on the robotic cell layout. Robot is in the center operating mainly in the MASH playground and store area. The TV screen is located bellow playground to allow to change background image.

3.1.7 Store

The area behind the robot is intended as a store of blocks. The printed plan allows to manually place the blocks into the store and the robot could blindly grasp for them. The robot will take blocks from the store during initialization and return them after the experiment during routine use.

3.1.8 TV screen

TV screen is placed underneath the playground to allow to change background appearance during advanced experiments. Chosen was LCD SHARP 40LE705, screen has a HD resolution (1920x1080 pixels) and 40":

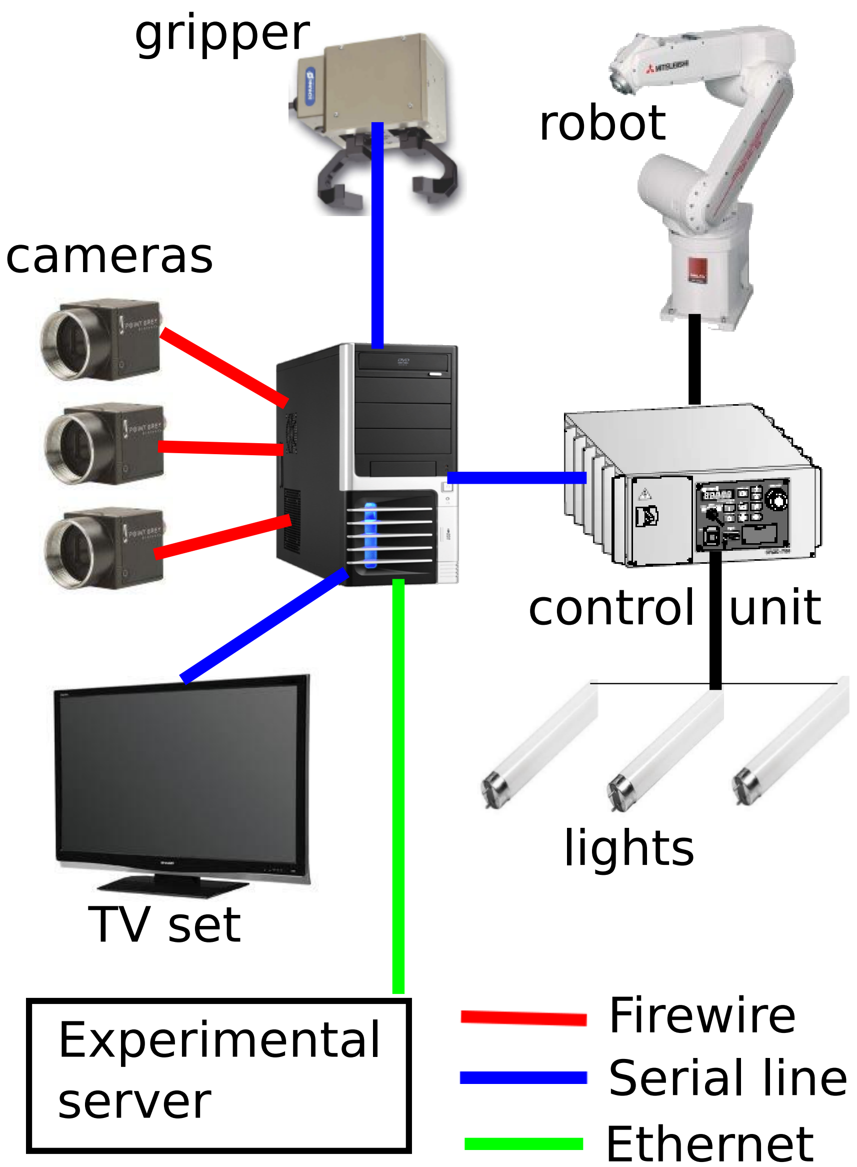

Figure 3.6: Signaling between robot platform components.

Chapter 4

Software architecture and structure

4.1 Programming languages

The software part of the robot platform is programmed in the various programming languages. MAS is programmed in `Java` using development tool `Eclipse`. The core environment is running in the `Matlab` and calling `MEX` functions in `C` language. All source codes of the robot platform are saved in the GIT repository of the project MASH at Switzerland.

4.2 Software structure

Robot application server is complex software solutions running as daemon. It has many software components, mainly part, called Mash Application Server (MAS) is handling incoming connections from the experimental server or the external users and allowing only one connection to the core environment. The software solution is divided into the several parts.

Main two parts are:

- Robot application server - manage incoming connections, allowing only one connection to the core environment. It is parsing and checking input commands and sending them to process to the core environment (`Matlab`) through MatlabControl library. It is written in JAVA SE 5 language. There is implemented TCP socket server to operate with connected experiment server or external user.

- Core environment - processing most of the commands, controlling all devices (robot, gripper, TV set, cameras). There is implemented whole logic of the experiment, manipulation with objects by the robot (placing and cleaning), object recognition, etc.

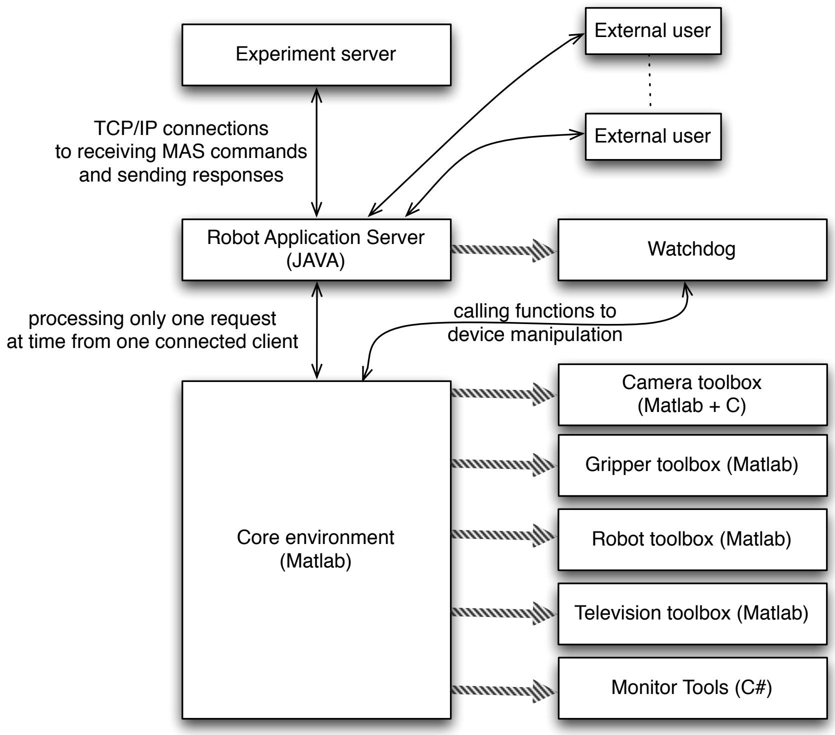

Figure 4.1: Software components of the application server.

The experiment server sends a command to the robot application server, which reply with a response. The application server never sends data spontaneously to the experimental server. Protocols is text-based, but binary data are enclosed in the case, where the images are as a response. All commands and responses are terminated by a UNIX end-of-line character.

Experimental server or external user is connected by TCP channel to the RAS. The communication between experimental server and application server is based on the commands described in the App. B. The command is send by experimental server and processed by RAS, where is command parsed, checked the syntax of the command. Commands, for a checking the status, version and type of application server, are processed immediately in the Robot application server. Other commands are forwarded to the core environment to the processing. Response from the core environment is sending back through the RAS to the experiment server or external user.

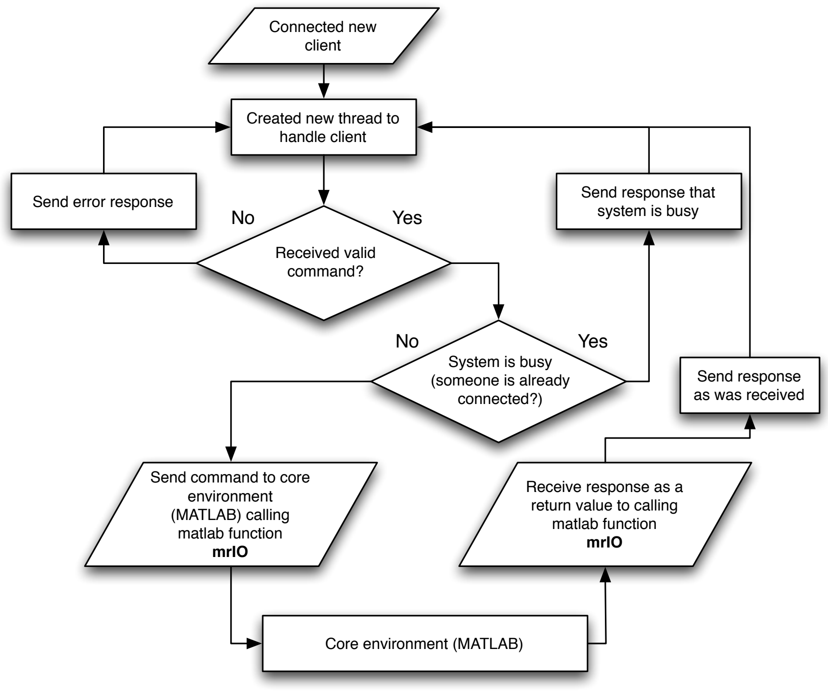
Connected new client
Created new thread to handle client
Send error response
No
Yes
Received valid command?
Send response that system is busy
No
Yes
System is busy (someone is already connected?)
Send response as was received
Send command to core environment (MATLAB) calling matlab function **mrIO**
Receive response as a return value to calling matlab function **mrIO**
Core environment (MATLAB)

Figure 4.2: Life cycle of the command processing in the Robot application server.

Example of the communication between experiment and application server (SEND): At the beginning of the experiment, the server is waiting for the selecting the goal and of select duties and the adjacent environment. Experimental server doesn't know the goals supported by our server, so it must ask by the command `LIST_GOALS` for them. Next, it need to know supported environments by selected the goal, so experimental server send the command `LIST_ENVIRONMENTS`. Next phase is initialization Robot application server by the command `INITIALIZE_TASK`. The devices are not still initialized. Supported views and actions are returned as a response. Following the part where can be setup custom properties of the experiments. Currently our server is not using this feature. All devices (robot, gripper, TV set, cameras, lights) are initialized at the end of the task setup. The objects are released to the playground and the robot is moved to the start position, which is defined in the goal object. Next part is self experiment, where are receiving action commands `ACTION` and as responses are sending commands with reward, state and event description. Images from specific camera is send if the command `GET_VIEW` is received. If the goal is reached after the action during the experiment, application server send response that experiment is finished. Then all devices close the communications and turn OFF.

Received command	Response
`STATUS`	`READY`

`INFO`	`TYPE ApplicationServer`
	`SUBTYPE Interactive`
	`PROTOCOL 1.2`

`LIST_GOALS`	`GOAL ReachRedCube`
	`END_LIST_GOALS`

`LIST_ENVIRONMENTS ReachRedCube`	`ENVIRONMENT STATIC_12x12`
	`END_LIST_ENVIRONMENTS`

`INITIALIZE_TASK ReachRedCube STATIC_12x12`	`AVAILABLE_ACTIONS UP DOWN RIGHT LEFT`
	`AVAILABLE_VIEWS CAMERA_TOP:640x480`

`BEGIN_TASK_SETUP`	`OK`
`END_TASK_SETUP`	`OK`

`ACTION RIGHT`	`REWARD 0`
	`EVENT Robot move`
	`STATE_UPDATED`

`GET_VIEW CAMERA_TOP`	`VIEW CAMERA_TOP image/mif 921608` image representing by byte array
...	...

Table 4.1: Example of the communication between experiment and application server.

4.2.1 Core environment

The core environment consists of two main parts:

- `MashFramework` - implement the whole logic of the experiment cycle and also commands processing
- `Modules` - support modules used to camera calibration, cube manipulation, object recognition, controlling the devices (robot, gripper, TV set) etc.

When the core environment is started (`Matlab`), the paths to the toolboxes and modules are initialized by the function `initPath`. Input function, which is processing all received commands, is `mrIO`. Responses are returned also through this function. Each command is processed by method with name `mrCMD_commandname`, where `commandname` is the name of the received command. This method is called from the input function `mrIO`. These functions are stored in the directory `Commands`.

The current state (selected goal and environment) is saved in `Matlab` memory in the static object `mrMain`.

The goals, environments, views and actions are coded modular, so adding new ones is easily (add class file to appropriate directory). There is no needed registration or manually hardcoding the list of goals, environments, views or actions. Everything is doing automatically that the appropriate directory is scanned for the class files.

There are existing the base `Matlab` classes for the representing goal, environment, view and action. The final classes inherit from the base classes. Directory structure, where are saved inherited final classes:

- `Goal` - goal classes inherited from the base class `mrGoal`
- `View` - view classes inherited from the base class `mrView`
- `Environment` - environment classes inherited from the base class `mrEnvironment`
- `Action` - saved action classes inherited from the base class `mrAction`

Abstract methods and properties, which are needed to be implemented in the inherited class, for each base class:

- `mrGoal` - representing the goal (Sec. 2.4.4)
  - property `name` - unique name of the goal

  - property `startPosition` - lattice coordinates of the start position
  - property `endPosition` - lattice coordinates of the goal position
  - method `getReward` - return the reward and state base on the performed action
  - method `initializeGoal` - method called during the initialization the task, body of the method can be empty

- `mrView` - representing the view (Sec. 2.4.6)
  - property `name` - unique name of the view
  - method `getImageData` - return the binary image data and the size of the byte array
  - method `getImageSize` - return the image resolution
  - method `getImageType` - return the image type (JPG, PNG, MIF, etc.)

- `mrAction` - representing the action (Sec. 2.4.5)
  - property `name` - unique name of the action
  - method `positionAfterMove` - return the position in the lattice coordinates which will be reach after the robot movement
  - method `performAction` - perform the action (robot movement)

- `mrEnvironment` - representing the environment (Sec. 2.4.3)
  - property `name` - unique name of the environment
  - property `goals` - list of supported goals
  - property `actions` - list of supported actions
  - property `views` - list of supported views
  - property `imageTV` - path to the image file to display on TV set
  - property `teachingSupported` - if the teaching (after each action is returned the action that best towards to the goal) is supported
  - property `borders` - corners of the playground in lattice coordinates
  - property `cubes` - list of cubes to release on the playground before starting the experiment

  - property `upwardWorld` - height of the robot over the playground in the millimeters during the experiment
  - method `resetEnvironment` - method called during the reset the environment
  - method `initializeEnvironment` - method called during the initialization phase

The modules of the core environment are using toolboxes described in App. A. In the next part are described modules for the robot, gripper, TV set operation, cube manipulation, object recognition.

4.2.1.1 Robot manipulation

- Opening the communication with the robot, turn ON the servo and set the default speed.

  ```
  r = robotOpen('RV6SDL');
  ```

  - input: name of the robot
  - output: handle to the object of the robot

- Closing the communication with the robot, turn OFF the servo.

  ```
  robotClose(r);
  ```

  - input:
    - `r` - handle to the object of the robot

- Setting the robot speed.

  ```
  setSpeed(r,speed);
  ```

  - input:
    - `r` - handle to the object of the robot
    - `speed` - value of the speed
  - example (set the speed to 20):

  ```
  setSpeed(r,20);
  ```

- Moving the robot in the cartesian coordinates. The method is waiting until the robot reach the position.

  ```
  moveToPosXYZABC(r,position);
  ```

  - input:
    - `r` - handle to the object of the robot
    - `position` - column vector representing the position $\begin{bmatrix} x & y & z & a & b & c \end{bmatrix}^T$, where x, y, z are cartesian coordinates in mm and a, b, c are angles yaw, pitch, roll in degrees.
  - example (move robot to the position $\begin{bmatrix} 600 & 0 & 400 & 180 & 0 & 180 \end{bmatrix}^T$):

  ```
  moveToPosXYZABC(r,[600;0;400;180;0;180]);
  ```

- Moving the robot in the lattice coordinates. Transformation between lattice and robot coordinates is defined the configuration file `conf`. The method is waiting until the robot reach the position.

  ```
  moveToPos(r,latticePos, up, fi);
  ```

  - input:
    - `r` - handle to the object of the robot
    - `latticePos` - column vector representing the position in the lattice coordinates $\begin{bmatrix} x & y \end{bmatrix}^T$.
    - `up` - optional parameter defining the height (axis Z), default value is 200mm.
    - `fi` - optional parameter representing the rotation of the last joint J6 in degrees, default value is 180.
  - example (move robot to the position $\begin{bmatrix} 2 & 3 \end{bmatrix}^T$) with default height and rotation:

  ```
  moveToPos(r,[2;3]);
  ```

- Moving the robot in the joint coordinates. The method is waiting until the robot reach the position.

  ```
  moveToJoint(r,jointPos);
  ```

  - input:

- `r` - handle to the object of the robot
- `jointPos` - column vector of rotation angles for each joint in degrees $\begin{bmatrix}\alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6\end{bmatrix}^T$.

- example (move robot to the position $\begin{bmatrix}-90 & 0 & 90 & 0 & 90 & 0\end{bmatrix}^T$):

```
moveToJoint(r,[-90;0;90;0;90;0]);
```

- Getting the current robot position in cartesian coordinates.

```
[~, position, ~, ~] = mmGetPos(r, 'P');
```

- input:
  - `r` - handle to the object of the robot
- output:
  - `position` - column vector of the current robot position in the cartesian coordinates $\begin{bmatrix}x & y & z & a & b & c\end{bmatrix}^T$

4.2.1.2 Gripper controlling

- Power ON the gripper, open the communication with the gripper, reset the gripper errors, turn ON the light on the gripper.

```
g = gripperOpen(r,'SCHUNK');
```

- input:
  - `r` - handle to the object of the robot (needed for a power on the gripper)
  - name of the gripper

- output:
  - `g` - handle to the object of the gripper

- Close the communication with the gripper, turn OFF the lights and power OFF the gripper.

```
gripperClose(r,g);
```

- input:

    - `r` - handle to the object of the robot
    - `g` - handle to the object of the gripper

- Move the gripper to the position

```
gripperMovePos(g,position);
```

  - input:
    - `g` - handle to the object of the gripper
    - `position` - position of the gripper in millimeters
  - example of usage (move the gripper to the position 50mm and wait until the gripper reach the position):

```
gripperMovePos(g,50);
gpWaitForPos(g,50);
```

- Gripping by the force:

```
gripperMoveGrip(g,force);
```

  - input:
    - `g` - handle to the object of the gripper
    - `force` - force representing as a current [A].
  - example of usage (grip by force and wait until the gripper stop $\Rightarrow$ object is gripped):

```
gripperMovePos(g,-1);
gpWaitForStop (g);
```

- Stop moving the gripper:

```
gpStop(g);
```

  - input:
    - `g` - handle to the object of the gripper

4.2.1.3 Operating with TV set

Set of commands used to control the TV set.

- Open the communication with the TV set and turn ON the screen.

```
t = televisionOpen(r,‘SHARP’);
```

  - input:
    - name of the television

- output:
  - `t` - handle to the object of the TV set

- Close the communication with the TV set and power OFF it.

```
televisionClose(t);
```

  - input:
    - `t` - handle to the object of the TV set

- Select image as a background on the TV set.

```
setTvPicture(path)
```

  - input:
    - `path` - full path to the image file
  - example of usage (set the black image as a background):

```
setTvPicture(’C:\Img\black.png’);
```

4.2.1.4 Camera manipulation

Camera context is open only once during the initialization or at each change of camera mode (frame-rate, resolution). Next, the images are captured from the camera. All open camera contexts are saved in the `Matlab` memory (static object `camSingleton`). To camera manipulation is used Camera Toolbox described in App A.4.

- Grab the image from the camera.

```
im = ctGrabImage(cam_num, mode);
```

- input:
  - `cam_num` - camera number to grab the image
    1 - oblique camera
    2 - top camera
    3 - gripper camera
  - `mode` - handle to the object of the TV set
    1 - resolution 640x480, frame-rate 7.5fps
    2 - resolution 800x600, frame-rate 7.5fps
    3 - resolution 1024x768, frame-rate 7.5fps
    4 - resolution 1280x960, frame-rate 7.5fps
  - example of usage (grab the image from the oblique camera with resolution 640x480):

```
im = ctGrabImage(1, 4);
```

• Close all camera contexts.

```
camSingleton.instance.closeAllContexts
```

4.2.1.5 Object manipulation

The 19 objects are located on the right side of the robot at object store area described in Sec. 5.8. There are described methods used to manipulate with objects (gripping, releasing, cleaning, deploying). There is described set of methods used to object manipulation.

• Get the position of the object from the object store area in the robot coordinates.

```
pos = cubeStore(id);
```

- input:
  - `id` - id number of the object (see App. D)
- output:
  - `pos` - position of the center of the object in robot coordinates

• Clean all objects back to the object store area.

```
cubeClean(r, g, objArray);
```

- input:
  - `r` - handle to the object of the robot
  - `g` - handle to the object of the gripper
  - `objArray` - matrix of type `[id,x,y;...]`, where `id` is number of the object, `x,y` is the position of the object, where actually is placed, in the robot coordinates

- Concept of the method to clean objects based on the image

```
cubeCleanFromImage(r, g)
```

  - input:
    - `r` - handle to the object of the robot
    - `g` - handle to the object of the gripper

- Release the objects to the playground from the object store area.

```
cubeDeploy(r, g, objArray);
```

  - input:
    - `r` - handle to the object of the robot
    - `g` - handle to the object of the gripper
    - `objArray` - matrix of type `[id,x,y;...]`, where `id` is number of the object, `x,y` is the position of the object, where it will be placed, in the robot coordinates

- Grip the object from the playground or store area.

```
cubeGrip(r, g, pos);
```

  - input:
    - `r` - handle to the object of the robot
    - `g` - handle to the object of the gripper
    - `pos` - column vector of the position `[x;y]` of the object to grip in the robot coordinates

- Place the object to the playground or store area.

```
cubeRelease(r, g, pos);
```

  - input:
    - `r` - handle to the object of the robot
    - `g` - handle to the object of the gripper
    - `pos` - column vector of the position `[x;y]` of the object to place in the robot coordinates

4.2.1.6 Object recognition

The methods from `Image Acquisition Toolbox` are used to object recognition (see Sec. 5.6)

- Convert the image to the binary image by thresholding.

```
bw = im2bw(im, level);
```

  - input:
    - `im` - image
    - `level` - threshold level (0-1)
  - output:
    - `bw` - binary image

- Computes the complement image of the binary image.

```
bw = imcomplement(im);
```

  - input:
    - `im` - binary image
  - output:
    - `bw` - complement image

- Remove all small objects (fewer than X pixels).

```
bw = bwareaopen(im, X);
```

  - input:
    - `im` - binary image
    - `X` - objects with area lower then X pixels will be removed
  - output:
    - `bw` - filtered binary image

- Trace the exterior boundary of objects.

```
[~,L] = bwboundaries(im, 'noholes');
```

  - input:
    - `im` - binary image after removing small objects
  - output:
    - `L` - traced exterior boundaries

- Measure the properties of image regions.

```
objProps = regionprops(im, 'all');
```

  - input:
    - `im` - traced exterior boundaries
  - output:
    - `objProps` - array of the properties recognized objects

4.3 Server configuration

The application server configuration is divided into the two parts. One configuration is for Robot application server (`Java`) and next one is for Core environment (`Matlab`).

4.3.1 Robot application server configuration

Robot application server is user configurable. There are options to setup parameters such as listening socket port, enabling watchdog, etc. The configuration section is located in the file `AppServer.java`.

List of setting options and their default values:

- `PORT` - Specifies the port on which the socket server listen. Default value is `10993`.
- `DEBUG` - Enables debug mode, where are additional informations written to the log. Default value is `false`.
- `MATLAB_RECONNECT` - It defines whether `Matlab` is automatically restarted after the crash. Default value is `true`.
- `WATCHDOG` - Enables watchdog (see Sec. 5.2). Default value is `true`.
- `WATCHDOG_TIMEOUT` - Specifies watchdog interval to fire watchdog procedure. Default value is `60` seconds.
- `BUGZILLA_SERVICE_PATH` - URL path to bugzilla webservice for a tracking errors in the system.

4.3.2 Core environment settings

The are option to setup many parameters such as paths to the toolboxes, names of the used devices, environment setup, robot default position, calibration of the world, etc. The configuration settings are saved in the `M-file conf.m`.

Definition of paths to the toolboxes:

- `pathMonitorTools` - Specifies the path to application `Monitor Tools` (Sec. 5.1)
- `pathTvToolbox` - Specifies the path to TV toolbox toolbox (App. A.3)
- `pathGripperToolbox` - Specifies the path to Gripper toolbox (App. A.2)
- `pathRobotToolbox` - Specifies the path to Robot toolbox
- `pathCameraToolbox` - Specifies the path to Camera toolbox

Settings the device names:

- `robotName` - Name of the robot. Default value is `RV6SDL`.
- `tvName` - Name of the TV. Default value is `SHARP`.
- `gripperName` - Name of the gripper. Default value is `SCHUNK`.

Parameters of the setup of the robot motion during placing, gripping, cleaning the objects, :

- `upwardService` - Specifies the service height [mm] of the robot when robot is moving from object store to playground and back. Default value is `200` mm.
- `downwardWorld` - It is defining the height [mm] of the robot during the laying of objects at playground or object store. Default value is `55` mm.
- `rotZ` - It specifies rotation of the last robot joint J6 in degrees during the experiment. Default value is `180` degree.

Enabling devices in the application server:

- `robotSupported` - Enables the robot in the application server. If it is 0, robot is not moving and also all operations with objects are not realizing. Default value is `1`.
- `tvSupported` - Enables the TV in the application server. If it is 0, image on TV screen is is not displayed. Default value is `1`.
- `gripperSupported` - Enables the robot in the application server. If it is 0, gripper is not used in the application server and operations with objects are disabled. Default value is `1`.
- `toolNumber` - Specifies the number of tools mounted on the robot.

  1 - gripper

  2 - ferromagnet

  Default value is `1`.
- `cubeManipulation` - Enables the operations with the objects (deploying the objects to the playground, cleaning the objects back to the object store). Default value is `1`.

Parameters of the world calibration (transformation between lattice and robot coordinates)

- `angle` - Defines the angle in degrees between coordinate systems. Default value is `90` degree.

- `step` - Specifies the scale [mm], ie. size of the one lattice. Default value is `20` mm.
- `trans` - Represents the beginning of lattice coordinate system in the robot coordinates. Default value is `[711;-600]`.

4.4 Guide to initialize robot platform

This guide describes process of initialization robot platform. There are several steps, which are needed to do.

1. **D**istribution board
   - The distribution board is powered on by the main switch on the left side.
   - Check, that the green power indicator on the left side is light on.
2. **R**obot
   - Turn on the robot control unit by switching the power switch to the upper position.
   - Wait, until the display is not showing status `READY` on the control unit.
   - Check, that the control switch is switched to `AUTOMATIC` mode.
   - Verify, that the teach pendandt is turned off. The light indicator `TP ENABLED` must be not light up in the front of the teach pendandt.
3. **T**V set
   - TV set turn on the power by switching the switch to ON on the left side of the TV screen.
4. **P**layground
   - The remaining blocks on the playground is necessarily to place in the object store area.
5. **C**omputer
   - Turn on the computer near the robot, optionally switch on LCD monitor.

- Log in by username MASH.

6. **S**tarting Robot Application server

   - Application server is started by the icon with name `ROBOT APPLICATION SERVER` which is placed on the desktop.
   - Console application window is automatically opened.
   - `MATLAB` is started by application server.

Chapter 5

Robot platform support modules

5.1 The application to display image in scene

Robotic cell contain TV set (see Sec. 3.1.8) used as a monitor background, which is placed in under the playground. It is mainly intended for simulation more real environment and make the recognition task more difficult.

TV set is connected to computer by serial line to control it and by HDMI to display a image.

The application is called `Monitor Tools` and it is written in `C# .NET` and it was a developed in MS Visual Studio 2010. It is using the windows extended desktop (TV set is representing the second monitor) because the computer has only one graphics card. The application is started maximized on the second monitor. The control PictureBox (control from Windows Forms) is used by application to display image.

Usage: `MonitorTools.exe path_to_image`

Usage from `Matlab` by using function `setTvPicture(path_to_image)`

5.2 Watchdog

Watchdog is a software timer that periodically triggers some actions. The role of watchdog is to check status of robotic cell components (robot, gripper, television, lights, cameras) and test them. If the watchdog test failed, error message is automatically reporting to bug-tracking system. The advantage is quick detection errors of some components caused by dropped communication or hardware failure and fixing them in time.

Watchdog testing components :

- robot - it opens/closes robot communication and robot moves from position $\begin{bmatrix}600\\0\\300\end{bmatrix}$ to $\begin{bmatrix}500\\0\\300\end{bmatrix}$ and back.
- gripper - it opens/closes gripper communication and move gripper from 60 mm to 50 mm and back.
- TV set - it opens/closes communication with TV set and turn it on/off.
- lights - it turns on/off lights around robotic cell.
- cameras - take image from all three cameras (top, oblique, gripper).

Watchdog is running every 2 hours if no experiments is performing. During the experiment is watchdog disableds.

It is written in `Java` and `Matlab` and is starting in separate thread together with Robot application server.

`Java` part consist of 2 classes (`Watchdog`, `AppServerObserver`). Class `Watchdog` is starting new thread and inform that a timeout has occurred. AppServerObserver is evaluating `Matlab` function `watchdog.m` through `MatlabControl JMI Wrapper`, which performs actions to test the components.

5.3 Camera calibration

The informations about the scene are obtained from camera. Top camera is used for the cleaning objects on the playground, because gives the view of whole playground.

Objects, which are recognized in the image, are gripped by the robot. We need to know the relation between the coordinates in the image and the coordinate system of the robot.

Objects used in the experiments has standard height 33mm, ie. the upper bases of objects lies in one plane. This allow us to solve this problem as 2D $\rightarrow$ 2D transformation. This transformation is represented by *h*omography, for that $\vec{u} = H\vec{x}$, where $\vec{u}$ is representing in our case homogeneous coordinates of the center of the object recognized in the image and $\vec{x}$ is representing homogenous coordinates of the center of the object in the robot coordinates system.

5.3.1 Computing of the homography

We have mapping $\vec{x} \to \vec{u}$, where $\vec{x}$ is value of positioning by the robot and $\vec{u}$ is recognized center of the object in the image. Homography is computed by DLT algorithm described in [14].

At least four correspondences (Fig. 5.1) are needed for a computing the homography by this algorithm. Correspondences are obtained by placing calibration object to four different positions in the view of top camera. More precise calibration is obtained if the positions are in the corners of the playground. Calibration object is placed by the robot to the selected position and the next step is to recognized the center of the object in the image from the top camera.

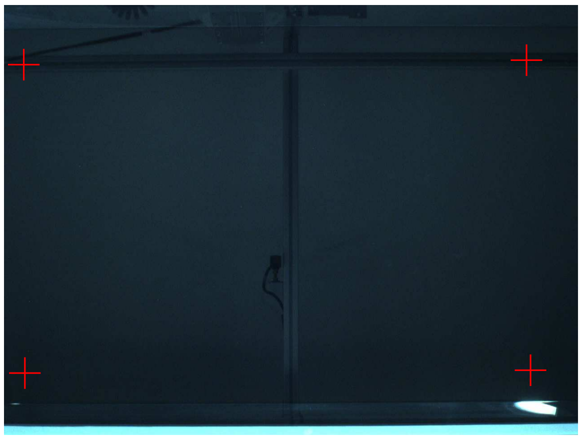
Figure 5.1: The positions of calibration object in the image.

We can see on Fig. 5.1 the quite dark image, because the TV set is turned off. Glass above the TV reflect the construction, where top camera is attached.

Recognition of the center of the object in the image:

- Capture the image from the top camera and crop them (selected rows from 60 to 870).
- Convert the image to the binary image by thresholding with level 0.4.
- Remove all small objects (fewer than 200 pixels).

- Trace the exterior boundary of objects.
- Measure properties of image regions.

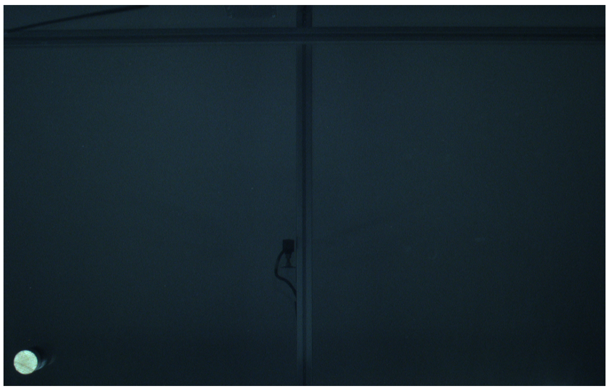
Figure 5.2: Captured image with the calibration object from the top camera.

Figure 5.3: Binary image with removed all small objects.

We have obtained correspondence $\vec{x} \rightarrow \vec{u}$, similarly repeat for the next positions to obtain four correspondences.

Homography is computed by the DLT algorithm, where the input are these correspondences.

$$H = \begin{pmatrix} 0.0066 & 0.5876 & 221.9813 \\ 0.5812 & -0.0050 & -375.5685 \\ 0.0000 & 0.0000 & 1.000 \end{pmatrix}$$

Usage: $\vec{x} = H^{-1}\vec{u}$, where $\vec{u}$ is recognized center of the object in the image.

Position	Robot coordinates		Image coordinates	
	x	y	x	y
1	700	-350	46.8	821.5
2	700	300	1175.8	815.1
3	300	-300	1166.8	123.0
4	300	-350	44.3	132.9

Table 5.1: Selected positions of the calibration object during calibration and recognized coordinates of the object centers.

5.4 World calibration

World, where is experiment processing, has own coordinate systems, called lattice coordinates. All robot movements during experiment and also coordinates of lying objects on playground are in lattice coordinates.

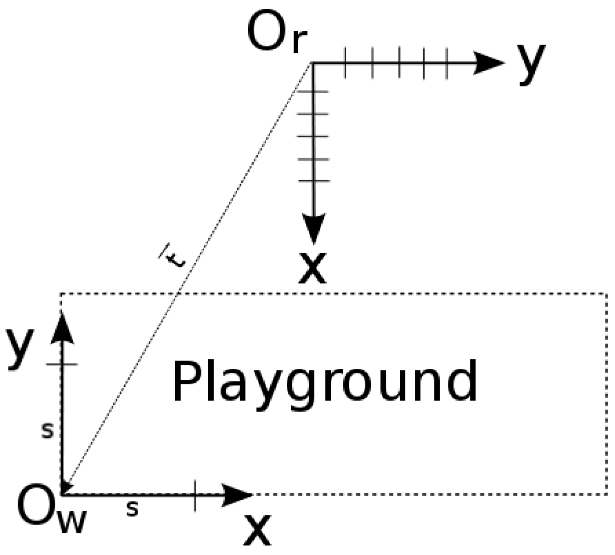

Figure 5.4: The relation between lattice and robot coordinates.

The relation $\vec{l} \rightarrow \vec{x}$ is represented by transformation matrix composed from rotation (M_r), scale (M_s) and translation matrix (M_t), where $\vec{l}$ are lattice homogenous coordinates and $\vec{x}$ are robot homogenous coordinates, that $\vec{x} = T\vec{l}$.

$$\begin{aligned}
M_r &= \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 & \\ -sin(\alpha) & \cos(\alpha) & 0 & \\ 0 & 0 & 0 & 1 \end{bmatrix} && (5.1)\\
M_s &= \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} && (5.2)\\
M_t &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} && (5.3)\\
T &= (M_s \cdot M_r \cdot M_t)^T && (5.4)
\end{aligned}$$

where α is angle between coordinate systems, $\vec{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$ is representing the beginning of lattice coordinate system in the robot coordinates, s is a scale coefficient.

Parameters α, $\vec{t}$ and s are saved in `Matlab` configuration file `conf.m`.

Current settings:

$$\begin{aligned}
\alpha &= 90 && (5.5)\\
s &= 20 && (5.6)\\
\vec{t} &= \begin{bmatrix} 711 \\ -600 \end{bmatrix} && (5.7)
\end{aligned}$$

Transformation matrix between lattice coordinates and the robot coordinates is

$$T = \begin{bmatrix} 0 & -20 & 710 \\ 20 & 0 & -265 \\ 0 & 0 & 0 \end{bmatrix}$$

Usage: $\vec{x} = T\vec{l}$, where $\vec{l}$ are homogenous lattice coordinates and $\vec{x}$ are homogenous robot coordinates.

5.5 Camera settings

Robot cell contains three color CCD cameras Pointgrey Flea2 to interact with environment. Each camera has own settings dependent on external illumination to return images with same color intensity, white balance. This settings can be made automatically by using of reference images or manually where the values are set to a certain type of environment. We are not using auto mode in camera, under our conditions did not work correctly (color intensity in image was too high). Settings is described for each camera in the App. C.4.

5.6 Object recognition

At the beginning of the experiments objects are placed on the playground and after experiments are put placed to the store. Because it is not possible to always rely on the fact that object will be positioned at the end of the experiment in the same position as at the beginning, so we need to find these objects using the camera.

It is necessarily to find coordinates of the centers of objects placed on the playground, their orientation and color. We chose the method for this purpose, which recognize objects and their parameters from the image from the top camera.

Procedure for object recognition:

- Prepare the environment for the recognition, ei. move robot out of the view of the top camera, turn on the lights.
- Set white background on the TV set and grab the image from the top camera.
- Set black background and also grab the image.
- Make difference between these two images (white image - black image). This image will be used further.

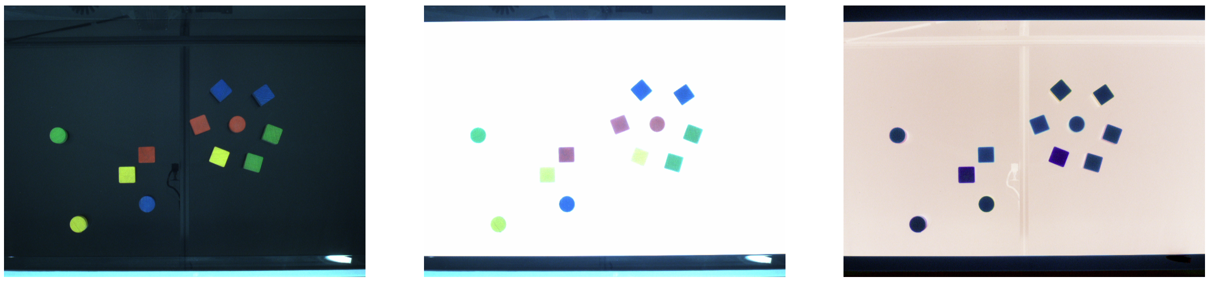

Figure 5.5: Image with black background (left), with white background (center) and the difference between them (right).

- Crop the image (selected are rows from index 60 to 870).
- Convert the image to the binary image by thresholding and computes the complement of the binary image.
- Remove all small objects (fewer than 200 pixels).

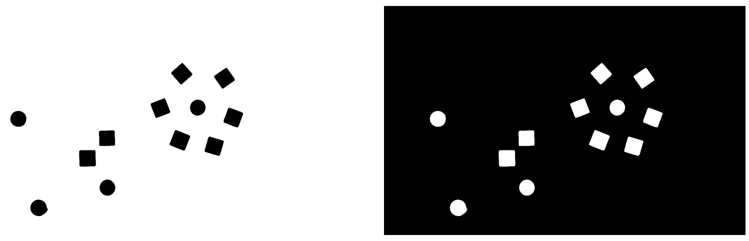

Figure 5.6: The binary image (left) and the complement to the binary image (right).

- Trace the exterior boundary of objects.
- Measure properties of image regions:
  - Area - the actual number of pixels in the region.
  - Perimeter - the distance around the boundary of the region.
  - Circularity - computed as $\frac{4\pi Area}{Perimeter^2}$.
  - Orientation - the angle between the x-axis and the major axis of the region.
  - Centroid - the center of mass of the region.
- Classify the object type by the thresholding measured properties (area, circularity).
- Classify the color of the object by the thresholding mean value for each color intensity.

Object type	Area		Circularity	
	From	To	From	To
cube	2800	3200	0.65	0.85
cylinder	2100	2600	0.85	1
bridge	9200	9600	0.4	0.65

Table 5.2: Thresholds for the object type classification.

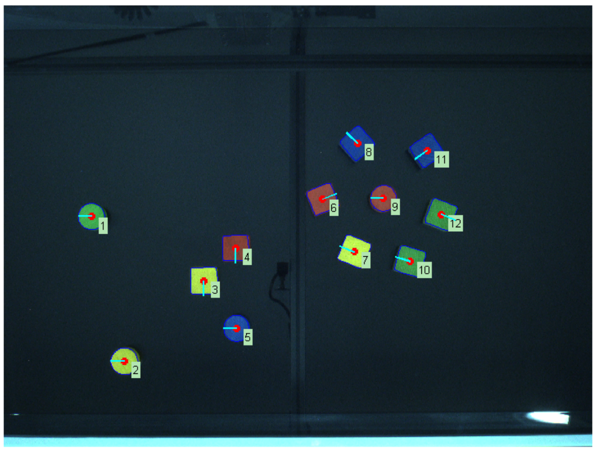

Figure 5.7: Recognized objects with orientation and center.

Color	R		G		B	
	From	To	From	To	From	To
blue	20	55	40	75	85	135
natural	145	190	125	170	100	140
yellow	175	230	150	200	40	80
green	55	120	110	160	40	80
red	155	220	45	75	35	60

Table 5.3: Thresholds for the object color classification.

5.7 Object manipulation

We need to prepare playground for the experiments. Playground is composed from various types of objects which are placed on them by robot. Robot have to grip the object in the object store are and place it to playground on the specified position. When the experiment is finished, all objects on the playground are cleaned back to the object store area.

5.7.1 Object placing to the playground

There is described algorithm to placing the objects from the object store area to the playground.

```
input: An array of the ids and positions in the robot coordinates of the
       objects, which will be placed on the playground $ObjArray$ of size $3 \times n$
place all objects to the playground;
for $i \leftarrow 1$ to $n$ do
    get the position of the object in the robot coordinates, where will be object
    placed on the playground:
    playgroundObjectPosition $\leftarrow ObjArray(i, 2:3)^T$;
    get the position of the place in the object store area in the robot coordinates,
    from where will be object gripped:
    cubeStoreObjectPosition $\leftarrow$ CubeStore($ObjArray(i, 1)$);
    grip the object from the object store area:
    CubeGrip(cubeStoreObjectPosition);
    place the object to the playground:
    CubeRelease(playgroundObjectPosition);
end
```

Algorithm 1: Placing the objects to the playground

5.7.2 Object cleaning from the playground

There is described algorithm to cleaning the objects from the playground to the object store area.

```
input: An array of the ids and positions in the lattice coordinates of the
        objects on the playground, which will be cleaned $ObjArray$ of size $3 \times n$
clean all objects placed on the playground;
for $i \leftarrow 1$ to $n$ do
    get the position of the object placed on the playground in the robot
    coordinates from the input:
    playgroundObjectPosition $\leftarrow ObjArray(i, 2:3)^T$;
    get the position of the place in the object store area in the robot coordinates,
    where will be object cleaned:
    cubeStoreObjectPosition $\leftarrow$ CubeStore($ObjArray(i, 1)$);
    grip the object from the playground:
    CubeGrip(playgroundObjectPosition);
    place the object to the object store area:
    CubeRelease(cubeStoreObjectPosition);
end
```

Algorithm 2: Cleaning the objects from the playground

5.8 Object store area

Objects, which are used for creating environments, for experiments and for calibration, are stored in special area on the side in the ready of the robot.

At the beginning of the experiments objects are placed on the playground and after experiments are placed back to the store.

The plan contains printed bases of objects with exact size and approximate color. It is generated as post-script file in a `Matlab` together with MAT-file, which contains coordinates of the centers of the objects in store (post-script) coordinations. This generator is using one source data file (`Excel`), where are described the properties of all objects (height, width, color, eventually diameter). Post-script file is printed to a paper, which is attached to the table on the side of the robot. The service operator is placing the cubes by color, shape and symbol.

For better manipulation, recognition was chosen set of blocks with standard module 33 mm. Subset of 18 objects various colors and shapes was selected. The plan has own coordinate system with center in upper-left corner. Unit is millimeter.

Store contains 18 objects (3 types of shapes, 5 types of colors) and 1 special calibration cylinder. Each object has unique ID represented as number $(1 - 19)$ list in App.D.

Color	Amount
green	4
red	4
blue	4
natural	4
yellow	4

Table 5.4: Amount of the color types.

Shape	Amount
bridge	2
cylinder	7
cube	9

Table 5.5: Amount of the object types.

Chapter 6

Conclusion

The goal of the thesis is to study the robot Mitsubishi RV6SDL, the planned function of robot platform in the project MASH. Design software for remote robot control, automatic distribution and resetting of manipulated objects. Next task is to implement designed software and perform experiments with it.

Integration of robotic cell to the MASH platform and design of the robotic task (environments, goals, actions) is described in the first part. Currently are is implemented one series of robotic tasks, where the goal is reach the red cube at four different environments. Devices (robot, gripper, camera, TV set), which are used in the robot platform, are described in the next part.

Chapter about software architecture and its structure describes in detail software solution, where the application has two main parts - Robot application server written in `JAVA` and Core environment written in `Matlab`.

Last part describes modules: camera calibration, world calibration, application to display image in scene, object manipulation and recognition, which are used in the robot platform.

Robot platform is able to to perform action (robot movement) on selected environment (currently implemented four environment) which is represented by objects placed on the playground with specified background, with chosen goal (reach the read object on the playground) and return reward of this action and images from cameras providing perception.

The displacement of blocks takes currently about 5 seconds per block, the similar is the time for block removal. Both times seems to be both sufficiently fast as well there is a little space for improvement.

The standard motion step, that is accepting the command, move, capturing of three full resolution images and sending them to the experiment server takes currently about 4 seconds, most of the time consumed by image handling and transfer. There seems to be quite a space for improvement here.

The robot platform is connected to the MASH platform. It is very often used by our contributor at INRIA France for the goal-planning experiments.

The platform is modular, so it is possible to simply adding new goals, environments, actions. It has several features which can be implemented in future. One of the planned extensions is to have the possibility to define the background image by the user. Extension of the MAS protocol is needed in this case. Next one is the possibility to define the size of the world and the position of individual blocks will greatly extent the flexibility of the robotic platform. One of the extensions worth to consider is an user defined reward. This significantly increases flexibility of the system to perform different tasks without explicitly recoding the properties of the robot platform.

Bibliography

[1] P. Abbet and F. Fleuret. First series of simulator tasks. European Project MASH - Massive sets of Heuristics for Machine Learning deliverable, June 2010.

[2] P. Abbet and F. Fleuret. Protocol specifications. European Project MASH - Massive sets of Heuristics for Machine Learning deliverable, March 2010.

[3] The MathWorks, Inc. *MATLAB Documentation*, 2011. `http://www.mathworks.com/help/techdoc/` [Accessed 12th May 2011].

[4] M. Meloun. Mitsubishi melfa robot control toolbox pro matlab, 2011. `http://cw.felk.cvut.cz/doku.php/help/common/robot_mitsubishimelfa_toolbox/` [Accessed 12th May 2011].

[5] MITSUBISHI ELECTRIC CORPORATION. *CR1D/CR2D/CR3D Controller - INSTRUCTION MANUAL - Controller setup, basic operation, and maintenance*, 2009.

[6] MITSUBISHI ELECTRIC CORPORATION. *CRnQ/CRnD Controller - INSTRUCTION MANUAL - Detailed explanations of functions and operations*, 2009.

[7] MITSUBISHI ELECTRIC CORPORATION. *RV-6SD Series - INSTRUCTION MANUAL - ROBOT ARM SETUP, MAINTENANCE*, 2009.

[8] MITSUBISHI ELECTRIC CORPORATION. *RV-6SD/6SDL Series - Standard Specifications Manual*, 2009.

[9] SCHUNK GmbH. *Servo Electric 2-Finger-Parallel-Gripper Type PG 70, Assembly and Operating Manual*, 2009. `http://cmp.felk.cvut.cz/cmp/hardware/chapadloSchunk/PG70_en_2010-02.pdf` [Accessed 12th May 2011].

[10] SCHUNK GmbH. *SCHUNK Motion*, 2010. `http://cmp.felk.cvut.cz/cmp/hardware/chapadloSchunk/MotionControl_Eng.pdf` [Accessed 12th May 2011].

[11] SHARP ELECTRONICS. *LCD COLOUR TELEVISION - OPERATION MANUAL*, 2009.

[12] V. Smutný and M. Dubec. First series of robotic tasks. European Project MASH - Massive sets of Heuristics for Machine Learning deliverable, December 2010.

[13] V. Smutný and M. Dubec. Operational robot. European Project MASH - Massive sets of Heuristics for Machine Learning deliverable, September 2010.

[14] Hartley, R. and Zisserman, A. *Multiple View Geometry in computer vision.* Cambridge University Press, 2nd edition, 2003.

[15] t. Sonka, M. and Boyle, R. *Image Processing, Analysis, and Machine Vision.* Thomson Learning, 3rd edition, 2007.

Appendix A

Appendix: Toolboxes

To control hardware devices is necessary to have a software. Many devices are delivered only with a description of the communication interface. Therefore is necessary to program the specific driver for the communication. The next section will be described such drivers for robot, gripper, television and camera. All of these devices except cameras communicates with the computer via a serial line.

A.1 Robot toolbox

The Mitsubishi robot RV6SDL (Sec. 3.1.1) is controlled by the `Mitstubishi Melfa Toolbox`. Detail description of this toolbox is in manual [4].

A.2 Gripper toolbox

A.2.1 Introduction

This toolbox provides access to gripper control in Matlab using a standard interface for grippers Schunk. Allows to move the gripper to certain position or gripping by force. Toolbox actually supports electrical gripper SCHUNK PG70. Communication with the gripper controller is realized through RS232 interface. Gripper control toolbox requires MATLAB 7.6 (R2008) and above, with the support serial port object (x64 to be higher). Make sure the power is on the gripper.

A.2.2 Initialization and close

```
gripper = gpOpen('SCHUNK')
```

Function load gripper specification and open communication port. The port must be free for Matlab. Object gripper is an instance of class derived from *hgsetget* class. This function returns the control object used by other toolbox functions.

```
gpClose(gripper)
```

Close the communication with the gripper.

A.2.3 Typical workflow with toolbox

First we need to open communication with the gripper by using `gpOpen`, thus creating a handle to the gripper. Closing of communication shall be made by calling function `gpClose`. Function `gpOpen` required free serial port, after calling is serial port acquired. On other side, function `gpClose` is expecting occupied serial port, and after end of communication with gripper is released.

Writing to the serial link is performed by `gpPrintf`, reading `gpScanf`. Respectively, can be used the `gpSendCmd` for write command and to read the answers. `gpExec` function sends the command specified in [10] and reads the answer. Identifier of the gripper is automatically entered from gripper description. All functions check whether the communication was successful.

Finally, the functions `gpClose` close communication with the gripper.

Returns variable r, which returns most of the functions, means that there is a problem communicating with the gripper (force command, etc), where r is zero, so communication was successful, ie. a command was sent from a PC, the gripper took command and is syntactically correct. For some functions with Safe option, which immediately after sending the command checks whether the gripper was an error.

A.2.4 Features overview

All functions are prefixing gp. For all the functions is the argument a handle to the gripper and the return variable r (control communications described above). Summary of commonly used functions and the arguments / return values:

gpAcquireCom	Acquires this com port for a gripper.
gpClose	Closes communication with gripper.
gpComprops_Serial	Creates serial object properties structure. The properties' names must match to the serial object properties' names.
gpFlush	Flushes gripper comport.
gpGetState	Returns gripper position, current, velocity and state.
gpGrippers	Lists available grippers implemented in the toolbox.
gpIsFreeCom	Whether the communication is not acquired.
gpMoveGrip	Moves the gripper to force.
gpMovePos	Moves the gripper to specified position.
gpOpen	Opens the communication with gripper.
gpParseIO	Parses the IO state bitfield into structure.
gpParseState	Parses the gripper state bitfield into structure.
gpPurge	Clears errors.
gpSendCmd	Sends a command to gripper and reads the reply.
gpSendCmdSafe	Sends a command to gripper, reads the reply and checks error.
gpStop	Stops the gripper.
gpWaitForPos	Waits until the gripper reach position.
gpWaitForStop	Waits until the gripper is in stop state.

Table A.1: List of basic commands to manipulate with gripper.

A.2.5 Basic movement

The control unit returns the position of gripper in millimeters. Function `gpMovePos` perform movement for certain position, otherwise function `gpMoveGrip` perform movement on the force. All movement functions are asynchronous. Waiting for completion of movement can be used gpWaitForStop. Detailed description of physical features is the manual for a gripper [9].

A.2.6 Example

```
%open the gripper communication with default properties
g = gpOpen('SCHUNK');

%move gripper to position 50mm
gpMovePos(g,50);

%wait unit the gripper reach the position 48mm
gpWaitForPos(g,48);

%gripping by force -1A
gpMoveGrip(g,-1);

%wait to stop the gripper
gpWaitForStop(g);

%get the current position
pos = gpGetState(g);

%close the communication with gripper
gpClose(g);
```

A.3 TV set toolbox

Television toolbox is used to control television in `Matlab`. Provides remote control of TV through serial line. Allows power on/off, switch the input source. Toolbox is supporting only SHARP televisions. It requires `Matlab` 7.6 (R2008) and above, with the support serial port object (x64 to be higher).

A.3.1 Initialization and close

```
tv = tvOpen('SHARP')
```

Function load television specification and open communication port. The serial port must be free for `Matlab`. Object television is an instance of class derived from *hgsetget* class. This function returns the control object used by other toolbox functions.

```
tvClose(tv)
```

Close the communication with the television.

A.3.2 Typical workflow with toolbox

First we need to open communication with the television by using `tvOpen`, thus creating a handle to the television. Closing of communication shall be made by calling function `tvClose`. Function `tvOpen` required free serial port, after calling is serial port acquired. On other side, function `tvClose` is expecting occupied serial port, and after end of communication with the television is released.

Writing to the serial link is performed by `tvPrintf`, reading `tvScanf`. Respectively, can be used the `tvSendCmd` for write command and to read the answers. `tvExec` function sends the command specified in [11] and reads the answer. Identifier of the television is automatically entered from television description. All functions check whether the communication was successful.

Finally, the functions `tvClose` close communication with the television.

Returns variable r, which returns most of the functions, means that there is a problem communicating with the television (force command, etc), where r is zero, so communication was successful, ie. a command was sent from a PC, the television took command and is syntactically correct. For some functions with Safe option, which immediately after sending the command checks whether the television was an error.

A.3.3 Example

```
%open the TV set communication with default properties
t = tvOpen('SCHUNK');

%power on tv
tvPowerOn(t);

%set the HDMI1 as source of signal
tvInputHDMI(t,1);

%mute the volume
tvVolume(t,0);

%power off tv
tvPowerOff(t);

%close the communication with TV set
tvClose(g);
```

A.4 Camera toolbox

The cameras are connected to the computer by bus Firewire 800. They are controlled from `Matlab` by using MEX libraries written in C. The MEX libraries are using original SDK FlyCapture2 from Point Great Research to direct access to cameras. They were developed in the MS Visual Studio 2010.

`Matlab` function `ctGrabImage` is creating (if not existing yet) camera context, next grab the image and demosaicing them.

List of MEX libraries and their description:

- ctInitCam - create the camera context, connect to this context, set the video mode and frame-rate and start capturing
- ctGrabImg - grab the image from the buffer, convert them to RGB format

- ctStopCam - stop capturing and destroy the context

A.4.1 Example

```
%open the camera context for the TOP camera
%with resolution 640x480
context = ctInitCam();

%grab the images
img1 = ctGrabImg(context);
img2 = ctGrabImg(context);
img3 = ctGrabImg(context);

%stop the camera context
ctStopCam(context)
```

Appendix B

Appendix: MASH Application Server Protocol

B.1 Description of the MAS protocol

This section details the version 1.5 of the protocol used by the Robot Application Server. [2]

B.1.1 Command: `STATUS`

Response:

```
READY
```

OR

```
BUSY
```

Description:

Indicates whether the Server can accept commands from the Client on this connection, or if the Server is already busy.

For instance, the Experiment Servers can only run one experiment at a time, but the Image Server can serve images to an unlimited number of clients.

B.1.2 Command: `INFO`

Response:

```
TYPE ApplicationServer
SUBTYPE Interactive
PROTOCOL 1.5
```

Description:
Used to check that the Client talks to the correct Server, using the correct protocol.

B.1.3 Command: `DONE`

Response:

```
GOODBYE
```

Description:
The connection is closed after that.

B.1.4 Command: `LIST_GOALS`

Response:

```
GOAL <name 1>
GOAL <name 2>
...
GOAL <name N>
END_LIST_GOALS
```

Description:
Ask for a list of the goals provided by the Server.

B.1.5 Command: `LIST_ENVIRONMENTS`

Format:

```
LIST_ENVIRONMENTS <goal name>
```

Response:

```
ENVIRONMENT <name 1>
ENVIRONMENT <name 2>
...
ENVIRONMENT <name N>
END_LIST_ENVIRONMENTS
```

Otherwise:

```
UNKNOWN_GOAL <goal>
```

OR

```
INVALID_ARGUMENTS <arguments>
```

Description:

Ask for a list of the environments provided by the Server where the given goal can be used.

B.1.6 Command: INITIALIZE_TASK

Format:

```
INITIALIZE_TASK <task name> <environment name>
```

Responses:

When successful:

```
AVAILABLE_ACTIONS <action 1> <action 2> ... <action N>
AVAILABLE_VIEWS <view 1> <view 2> ... <view N>
```

Otherwise:

```
UNKNOWN_GOAL <goal>
```

OR

```
UNKNOWN_ENVIRONMENT <environment>
```

OR

```
INVALID_ARGUMENTS <arguments>
```

OR

```
ERROR <description>
```

Description:

Notify the Server about the task that the Client wants to work on. The Response contains:

- all the actions that can be performed on the task
- a description of each available view, in the form `<name>:<width>x<height>` (for example: `main:320x240`)

B.1.7 Command: `BEGIN_TASK_SETUP`

Responses:

```
OK
```

OR

```
NO_TASK_SELECTED
```

OR

```
ERROR <description>
```

Description:

Tell the Server that all the following Commands (up until `END_TASK_SETUP`) are task-specific settings. See the documentation of the chosen task for more details.

Note that a task must have been successfully initialized before receiving that command.

B.1.8 Command: `END_TASK_SETUP`

Responses:

```
OK
```

OR

```
ERROR <description>
```

Description:

Terminate the TASK_SETUP section.

B.1.9 Command: `TEACHING`

Format:

```
TEACHING <ON or OFF>
```

Response:

```
OK
```

OR

```
NOT_SUPPORTED
```

OR

```
NO_TASK_SELECTED
```

OR

```
INVALID_ARGUMENTS <arguments>
```

OR

```
ERROR <description>
```

Description:

When enabled (`ON`), the Application Server might provide “good” actions as part of the Responses to the `ACTION` Commands. They may be used to train a predictor. The default is `OFF`.

B.1.10 Command: RESET_TASK

Response:

When successful:

```
STATE_UPDATED
```

Otherwise:

```
NO_TASK_SELECTED
```

OR

```
ERROR <description>
```

Description:

Reset the state of the task. The initial state is task-dependent: it can be something random or always the same thing.

B.1.11 Command: GET_VIEW

Format:

```
GET_VIEW <view name>
```

Responses:

When successful:

```
VIEW <view name> <MIME type> <image size in bytes>
<binary data>
```

Otherwise:

```
NO_TASK_SELECTED
```

OR

```
INVALID_ARGUMENTS <arguments>
```

OR

```
UNKNOWN_VIEW <view>
```

OR

```
ERROR <description>
```

Description:

Retrieves one of the views. The Server indicates the format of the image by sending its MIME type as part of the Response. The following MIME types are supported:

- `image/ppm`: Portable Pixel Map
- `image/pgm`: Portable Gray Map
- `image/jpeg`: JPEG images
- `image/png`: PNG images
- `image/mif`: MASH Image Format (see Appendix B.2 on the following page)

B.1.12 Command: `ACTION`

Format:

```
ACTION <action>

```

Responses:

When successful:

```
REWARD <reward>
(optional) EVENT <event>
(optional) TEACHER_ACTION <action>
STATE_UPDATED | FINISHED
```

Otherwise:

```
NO_TASK_SELECTED
```

OR

```
UNKNOWN_ACTION <action>
```

OR

```
INVALID_ARGUMENTS <arguments>
```

OR

```
ERROR <description>
```

Description:

Perform the given action on the task.

`EVENT` is an optional part of the Response, that contains a human-readable description of what happened due to the action.

`TEACHER_ACTION` is an optional part of the Response, that provides a “good” action as calculated by the Application Server. It may be used to train a predictor.

`FINISHED` indicates that the task was successfully solved.

B.2 MASH Image Format

The MASH Image Format has a very simple structure. The primary goal is to simplify the implementation of the Application Servers, which do not have to rely on an external library to transmit images to their clients. [2]

The header for version 1 is 8 byte long, composed of the following unsigned chars:

Offset	**Value**	**Meaning**
0	77	ASCII code of ’M’
1	73	ASCII code of ’I’
2	70	ASCII code of ’F’
3	1	Version number
4	width % 256	Width least significant byte
5	width / 256	Width most significant byte
6	height % 256	Height least significant byte
7	height / 256	Height most significant byte

then from 8 to 8 + 3 * width * height, pixels are listed line after line. Each pixel is defined with 3 bytes to be interpreted as the RED, GREEN and BLUE components encoded as unsigned char (0..255).

Hence, the total byte size of an image in that format is

$$8 + 3 * \text{width} * \text{height}$$

Appendix C

Appendix: Device setup parameters

C.1 Robot

C.1.1 Robot control unit setup

The robot is unique identify by robot and slot number:

- `Robot number:` 1
- `Slot number:` 1

Robot control unit setup parameters:

- `MEPAR` - work above plane with the z coordinate. Value: 50 mm.
- `MEJAR` - sets the overrun limit value for each joint axis. Value:

Joint	- [deg]	+ [deg]
J1	-170	170
J2	-93	135
J3	-129	166
J4	-160	160
J5	-100	100
J6	-200	200

- `MEXTL` - sets the default value for the tool data. Value:
- `COMSPEC` specified the communication method of the robot controller and RT-ToolBox2 (conventional communication method). Value: 0

	(MEXTL) [mm, deg]
X	0
Y	0
Z	159.2
A	0
B	0
C	0

C.1.2 RS-232 parameters

RS-232 cable has to be crossed according documentation [7]. RS-232 is currently set to:

- COM9,
- 38400 baud,
- 8 bits,
- parity even,
- two stopbits,
- non-procedural,
- terminator CR

C.2 Gripper

The gripper is unique identify by gripper ID:

- `Gripper ID:` 12

C.2.1 RS-232 parameters

RS-232 is set to:

- COM8,
- 38400 baud,

- 8 bits,
- parity even,
- two stopbits,
- non-procedural,
- terminator CR

C.3 TV set

C.3.1 RS-232 parameters

RS-232 is set to:

- COM7,
- 9600 baud,
- 8 bits,
- non-parity,
- two stopbits,
- non-procedural,
- terminator CR

C.4 Camera

Each camera has own default settings described in table:

	Top	Oblique	Gripper
Brightness [%]	0	0	0
Exposure [EV]	1.757	0.539	1.398
Sharpness	1024	1024	1024
Hue [deg]	0	0	0
Saturation [%]	100	100	100
Gamma	1	1	1
Pan	2	2	2
Tilt	0	0	0
Shutter [ms]	16.020	149.869	172.678
Gain [dB]	-4.012	6.053	9.678
Frame Rate [fps]	7.5	7.5	7.5
W.B. (Red)	597	480	519
W.B. (Blue)	693	839	752

Table C.1: Default camera settings

Appendix D

Appendix: Object store area

D.1 Description of the objects

The next table is describing all objects placed in the object store area and their properties used as input to the plan generator (Sec. 5.8).

						Coordinates	
ID	Shape	Width	Height	Diameter	Color	x	y
1	cube	33	33	-	natural	55	50
2	cube	33	33	-	natural	165	50
3	cube	33	33	-	natural	275	50
4	cube	33	33	-	natural	385	50
5	cube	33	33	-	blue	495	50
6	cube	33	33	-	blue	55	150
7	cube	33	33	-	red	165	150
8	cube	33	33	-	red	275	150
9	cube	33	33	-	green	385	150
10	cube	33	33	-	green	495	150
11	cube	33	33	-	yellow	55	250
12	cube	33	33	-	yellow	165	250
13	bridge	100	33	-	red	275	250
14	bridge	100	33	-	green	495	250
15	cylinder	-	-	33	blue	55	350
16	cylinder	-	-	33	green	165	350
17	cylinder	-	-	33	yellow	275	350
18	cylinder	-	-	33	red	495	350

Table D.1: List of all objects placed in the object store area

D.2 Plan of the object store area

We can see on Fig. D.1 generated plan, which is attached to the table.

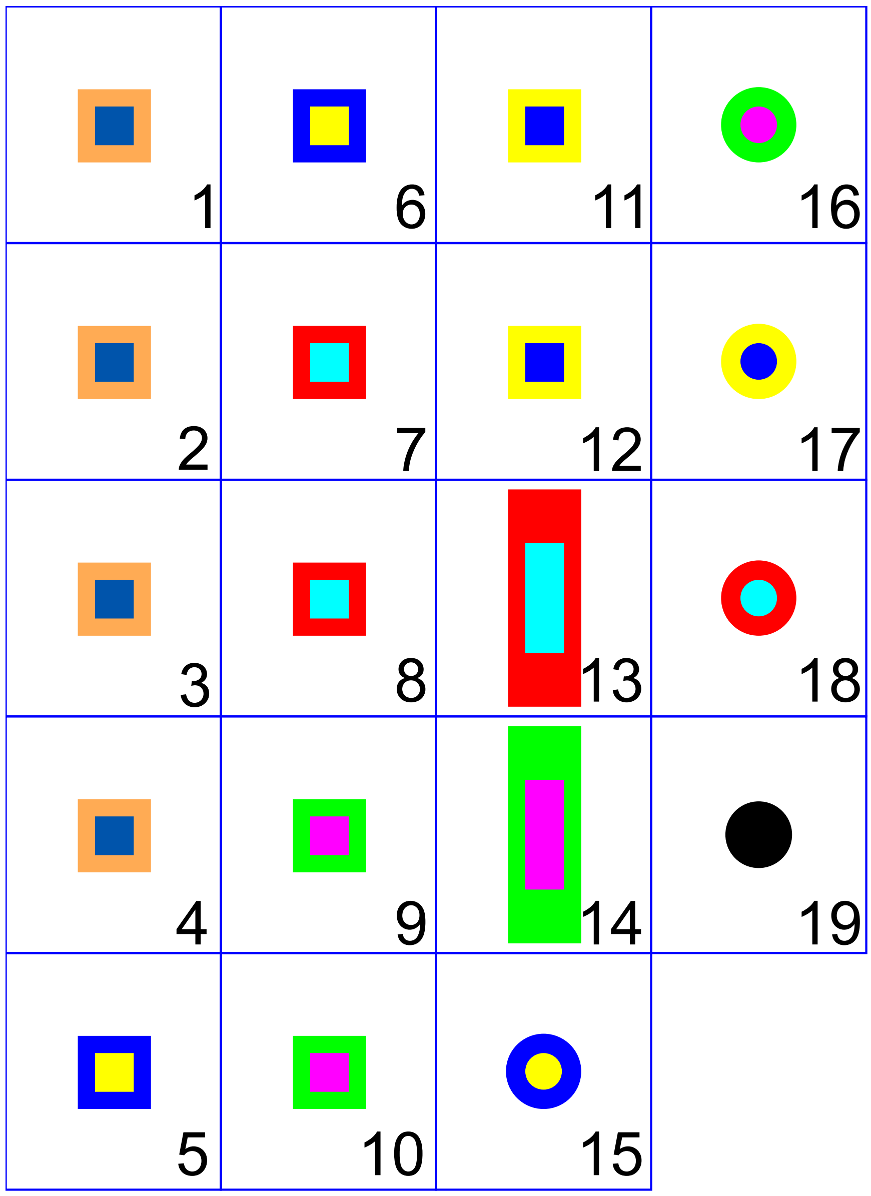

Figure D.1: The plan of the object store area.

Appendix E

Appendix: Content of the included CD

CD is included to work. It contains the source codes and electronic version of thesis.

- Directory `src` - complete source codes of the application server
- Directory `thesis` - final electronic version of the thesis
- Directory `thesis-src` - LaTeX source codes of the electronic version of the thesis (contains also figures)