Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Master's Thesis

Ekaterina Nikandrova

LEARNING ROBOT ENVIRONMENT THROUGH SIMULATION

Examiners:	Professor Ville Kyrki
	Docent Sergey Ivanovsky
Supervisor:	Professor Ville Kyrki

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Ekaterina Nikandrova

Learning Robot Environment through Simulation

Master's Thesis

2011

77 pages, 27 figures, 3 tables, and 1 appendix.

Examiners: Professor Ville Kyrki
Docent Sergey Ivanovsky

Keywords: simulation, robotic manipulation and grasping, world model, sensors, optimization

Traditionally simulators have been used extensively in robotics to develop robotic systems without the need to build expensive hardware. However, simulators can be also be used as a "memory"for a robot. This allows the robot to try out actions in simulation before executing them for real. The key obstacle to this approach is an uncertainty of knowledge about the environment.

The goal of the Master's Thesis work was to develop a method, which allows updating the simulation model based on actual measurements to achieve a success of the planned task. OpenRAVE was chosen as an experimental simulation environment on planning,trial and update stages. Steepest Descent algorithm in conjunction with Golden Section search procedure form the principle part of optimization process. During experiments, the properties of the proposed method, such as sensitivity to different parameters, including gradient and error function, were examined. The limitations of the approach were established, based on analyzing the regions of convergence.

PREFACE

First of all, I wish to thank my supervisor, Professor Ville Kyrki, for proposing me an interesting research topic and also for his help and guidance during the whole preparation period.

I wish to express my gratitude to Janne Laaksonen for his precise advices in the question of robot grasping.

Finally, thanks a lot to my parents for their support and encouragement to my work.

Lappeenranta, May 20th, 2011

Ekaterina Nikandrova

CONTENTS

ABBREVIATIONS AND SYMBOLS

DOF	Degree Of Freedom
CCD	Charged Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
COM	Center Of Mass
CRM	Contact Relative Motion
FED	Feature Extraction by Demands
FFC	Face-to-Face Composition
GUI	Graphical User Interface
NIAH	Needle-In-A-Haystack
ODE	Open Dynamic Engine
PAL	Physics Abstraction Layer
SSH	Special Semantic Hierarchy
WRT	World Relative Trajectory
φ	angle of rotation around the z-axis
$\mathbf{x}$	vector of uncertain object location
d_k	direction of the step
λ_k	step size
$\nabla f(\mathbf{x})$	gradient of the function $f(\mathbf{x})$
$\mathbf{g_k}$	gradient in a given point
$E(x)$	error function
$grad_{one}$	one–sided gradient
$grad_{two}$	two–sided gradient
$coll$	collision matrix
w	weight function
ang	matrix of finger joints angles

1 INTRODUCTION

The section provides the motivation and the main objectives of the work, as well as the structure of the thesis.

1.1 Motivation

Traditionally simulators have been used extensively in robotics to develop robotic systems without the need to build expensive hardware. Thus the simulation has been used only to plan the robot environment.

However, simulators can be also be used as a “memory”for a robot, that is, the simulation is the robot’s internal mental view of the world. This allows the robot to try out actions in simulation before executing them for real. However, predicted and measured sensor readings of the robot can differ, which indicates that the knowledge about the robot environment is uncertain. The question is how to update the simulation model on the basis of actual knowledge about the environment to minimize this difference. As a result, being based on the updated internal view, the robot will be capable to change its action to achieve success of the plan.

1.2 Objectives and Restrictions

There are two most actual things in this research. Primarily, it is necessary to determine which quantities are used as the input in a simulator. It can be different parameters of the object in a world: shape, mass, location. Other important point is a type of environment model for robot used in simulation. Among such models precise values, statistical models and fuzzy logic models can be mentioned.

Perception of the world by robot can be indirect. The results for the sensors can be modulated taking into account the uncertainty. In simulations different types of sensors can be involved. In this work only few sensors will be involved. First of all, contact sensors were selected. In reality these sensors determine only whether there was a contact with some point of the object. In simulation they define at which point of object robot gets the contact. These sensors are commonly used for manipulation.

Results obtained from a simulation are predicted values. Sensors register actual values. To make the simulation and “real”cases as identical as possible, the difference between measured (real) and simulated (predicted) sensor measurements should be minimized. Several solutions can be proposed to solve this problem. They depend on the chosen environment model for the robot. One possible way is to minimize the sum of squared errors over a single action attempt. This means to run an optimization algorithm to minimize the error. If a statistical model is used it is possible to apply a statistical optimization algorithm. In the overview part of the thesis different alternatives will be described, but only one approach will be chosen for realization. The goal is to determine the problems that can be solved by using this method and to answer the question how far this approach extends? (How big mistakes can be corrected with it?)

As a result a simple demonstration of a chosen approach should be developed. For this purpose free OpenRAVE simulator will be used. Practical example of the task is to move the object from one location to another (in 3-D space) if the location of the object is uncertainly known. To correct the simulation estimations in accordance with the actual knowledge about the environment and implement the goal action the optimization approach will be applied.

For further improvements intelligent optimization can be carried out. There are several options. One of it is to calculate the gradient to determine in which direction to move to get faster the optimum. In this case it is assumed to use numerical estimation by giving different initial parameters in simulator. Another option is to use available information about the object. This means that optimization should be performed only for uncertainly known object parameters.

1.3 Structure of the Thesis

The thesis is structured in seven sections. Section 2 considers the role of simulation in robotics especially in manipulation area. Relevant examples of applying the simulation in different fields of robotics are examined and the most popular simulators are observed. Section 3 focuses on world models conceptions and the problem of the perception of the world using sensors. First, world models using by robots in reality are presented. Next, the models implemented in simulation are discribed. In the last part, various sensor types and their role in manipulation task are introduced. Section 4 considers the different approaches that allow to update the world model based on observations of the environmental current state. Section 5 includes the description of the implemented system:the problem to solve, the explanation of the choosen tools, the algorithm realization and its integration with the already developed robotic simulator. Section 6 is intended for presenting and discussing the results of the experiments conducted. The points for future work are also provided in this section. Finally, the conclusion of the Master's thesis work is made in Section 7.

2 SIMULATION IN ROBOTIC MANIPULATION

Simulation is the process of designing a model of an actual or theoretical physical system, executing the model, and analysing the execution output [1]. Simulation has become an important research tool since the beginning of the 20th century. Initially, it was first of all an academic research tool. Nowadays, simulation is a powerfool tool providing the possibilities for design, planning, analysis and making decisions in various areas of research and development. Robotics, as a modern technological branch, is not an exception. Actually, simulation plays a very important role in robotics and especially in robotic manipulation.

2.1 Usage before

The first commercial robotic simulation software was developed by Deneb Robotics (now Dassault/Delmia) and Tecnomatix Technologies (now UGS/Tecnomatix) more than twenty years ago. The key goal of the elaborated products was to help solve the growing complexity of designing and programming robotic systems [2]. From that time until now simulation remains an essential tool in robotic research fields such as mobile robotics, motion and grasp planning.

In this section the role of the simulation in general is discussed and an overview of the simulation in robotics is provided.

2.1.1 The role of simulation

Zlajpah [1] mentions several fundamental concepts of simulation. First of them is the “learning”principle.The simulation gives a possibility to learn about environing objects in a very effective way and allows to observe the effect of the interaction by altering the parameters. The visualization in the simulation is another paramount in simulation.

The possibility to simulate opens a wide range of options for insertion of creativity into problem solutions. Results can be obtained and evaluated before the system is built. Use of simulation tools allows to avoid injures and damages. Unnecessary changes in design after starting the production can be also avoided. In research, simulators makes possible to build experiment environments with desired characteristics. Such factors, as complexity, reality and specificity, can be gradually increased in simulation mode [1].

Simulation has been recognized as an important tool in robotics: from designing of new products and their performance evaluation to designing applications of them. The study of the structure, characteristics and the function of a robot system at different levels is possible with simulation. The more complex the system under the investigation, the more important role the simulation plays. Simulations make possible to use computationally expensive algorithms, like genetic algorithms, that would be exclusively time consuming process to run on real robot microcontrollers. Thus, the simulation can enhance the design, development and even the operation of a robotic system.

Depending on the concrete application different structural attributes and functional parameters have to be modeled. For this purpose a variety of simulation tools has been created. They are used in mechanical modeling of robot manipulators, control systems design, off-line programming systems and many other areas.

2.1.2 Approaches to simulation of robotic systems

The majority of existing simulation software focus on the motion of the robotic manipulator in different environments. The central aspect of all simulation systems is a motion simulation that requires the kinematic and dynamic models of robot manipulators [1]. For example, trajectory planning algorithms or the construction of a robotized cell rely on kinematic models. Conversely, the design of actuators requires dynamic models. Thus, the choice of the specific model depends on the objective of the simulation system.

To model and simulate the robot manipulator different approaches are possible. One of the differences can be in the way the user builds the model. For example, in block diagram-oriented simulation software the user combines different blocks to create the model. The alternative are the packages requiring the manual coding.

The simulation tools for robotic systems can be divided into 2 key groups [1]:

1. Tools based on general simulation systems
2. Special tools for robot systems

First group includes special modules which simplify the implementation of robot systems and their environments within general simulation systems. Such integrated toolboxes enable to use other tools of the general system to implement different tasks.

The second group covers more specific purposes like off-line programming or mechanical design. Special simulation tools can be also specialized for certain types or families of robots like mobile or underwater robots.

2.1.3 Simulation in different fields of robotics

Robotics is a growing research field. Robots are introduced in various areas and the requirements for the simulation tools depend on the particular application.

The greatest advantage of robots is their flexibility. Although a robot can be programmed directly through the controller, the alternative is off-line programming that avoids to occupy the production equipment during the programming. The off-line programming takes place on a separate computer and uses the models of the work cell including robots and other devices.

The main goal of humanoid robotics is an introduction of humanoid robots into human environments in such a way that they will be able to collaborate with humans or do certain jobs instead of a human. These applications are based on control strategies and algorithms, sensory information and appropriate models of robot and environments. Among the topics addressed by researchers in this field one can distinguish virtual worlds, dynamic walking and haptic interaction [3, 4].

Nowadays, robotics is playing a very important role in medicine. The modeling of deformable organs, planning and simulation of robotic procedures, safe and real-time integration with augmented reality are the major topics in the area of robot-assisted surgery [5].

Simulation in mobile robotics helps reducing time and hardware needed for developing real applications and prototypes. It also allows researchers to focus on the most interesting parts of systems [1].

Nanorobotics is a challenging, dynamically growing industry, that requires the creation of very complex nanomechatronic systems. The simulation helps in design and development of such systems. As nanorobotic systems should effectively respond in real-time to changes in microenvironments, the simulation tools should be able to provide not only visualization but also to take into consideration physical aspects of nanorobots and their environment [6].

Space robotics is a very specific robotics branch. The behavior of space robots is much more complex compared to ground arm-like robots. Gravitation negligibility and conservation of linear and angular momentum make the control and trajectory planning highly complicated tasks [7]. Simulation plays an important role in solving these tasks. The simulation tools for space applications are designed in the way to be able to overcome difficulties of astronaut and ground personnel training and to guarantee space operations support.

2.1.4 Simulation for solving manipulation problems

Manipulation and particularly grasping are well-known problems in intelligent robotics. Great number of researchers use simulation in the process of evaluation and demonstration of their algorithms for solving these problems.

There are a lot of problems that require accurate location of a robot relative to an object in the world, including grasping and insertion. Hsiao, Kaelbling and Lozano-Perez [8] proposed to apply a decision procedure, based on world relative trajectories (WRT's), to a robot hand with tactile sensors, to find the object location in the world and finally achieve target placement. WRT represented the robot arm trajectory parameterized with respect to a pose of an object in the environment. To select the action in conditions of world state uncertainty a decision-theoretic controller was proposed. It consisted of state estimation and action selection components. The simulation was used to compare the performance of three different strategies. Tests were conducted on ten objects with different amount of world state uncertainty.

Sensor based grasping of objects and grasp stability determination are essential skills for general purpose robots. The presence of imprecise information in unstructured environments is a great challenge in state-of-the-art work. Using sensor information is the way to reduce this uncertainty. One approach [9] is to use the information from tactile sensors while grasping the object to tailor knowledge about grasp stability before further manipulation. For the inference a machine learning approach was applied. Good generalization performance requires large amount of training data. Generation of large datasets on real hardware is time consuming and difficult task, because of the dynamics of grasping process. The problem of acquiring enough training data was solved with help of simulation. A simulation model was used to generate data for both training and learning systems and also for evaluation. As a result, one-shot grasp stability recognition from a single tactile measurements and a time-series analysis were implemented. In addition, a database with examples of tactile measurements for stable and unstable grasps on a set of objects was generated from the simulation.

Lack of accurate sensory information and knowledge about shape and pose of graspable object is a big challenge for grasp synthesis algorithms. Platt in [10] focused on the case when only some limited general information is available and new information can be produced just by force feedback. He introduced the concept of a contact relative motions (CRM's) – a parameterizable space of atomic units of control, which in this method, at the same time, displaced contacts on object surface and collected relevant force feedback data. This approach allowed to transform the grasp synthesis problem into control problem which main goal is to find a strategy for executing CRM's with a smallest number of steps. This control problem is partially observable, since force feedback does not always determine the system state. The problem was expressed as a k-order Markov Decision Process (MDP) and was solved using Reinforcement Learning technique. The algorithm was tested in simulation for a planar grasping problem. Learning this grasp solution in simulation allowed thereafter to check the strategy on a real Robonaut robot.

Another example of applying simulation tools for methods solving grasping and manipulation problems in presence of object shape and position uncertainty was presented in [11]. On purpose to determine stable contact regions of a graspable object a special grasp quality metric was proposed. The elaborated method incorporated shape uncertainty into grasp stability analysis. The probability of force-closure grasp was computed by quantifying shape uncertainty using statistical shape analysis. Simulation experiments showed successful results in distinction grasps equivalent without uncertainty.

2.2 Examples of simulators

The purpose of this section is to study different variants of simulators to chose the most suitable and effective tool for the implementation of the task posed in this paper.

The term robotics simulator can refer to several different robotics simulation applications. As was discussed previously in 2.1.2 all simulators can be divided into general and special purposes groups. Another criterion for classification is a division into open source and proprietary categories. Examples of open source and proprietary simulators are presented in Table 1.

Table 1. Robotics simulators [12].

OpenSource	**Proprietary**
Stage, Gazebo, Pyro, Simbad, OpenSim, UsarSim, Robocode, OpenRave, Breve, Blender, GraspIt, OpenGRASP, Eyesim.	Webots, Marilou, Cogmaiton, Ms Robotics Studio, Modelica, Robologix, LabView, Simulink, MATLAB, Robotics Toolkit, EasyRob.

2.2.1 General simulation systems

One of the most used platform for simulation of robot systems is MATLAB and Simulink, adding extra dynamic simulation functionality. The main reasons for its popularity are capabilities of matrix calculations and easy extensibility. Among special toolboxes developed for MATLAB one can mention planar manipulators toolbox [13]. This toolbox is created for the simulation of planar manipulators with revolute joints and is based on Lagrangian formulation. It can be used to study kinematics and dynamics, to design control algorithms or for trajectory planning. It enables a possibility of real-time simulation. Due to its concepts it can be chosen as a tool in education process. Another example of MATLAB-based technology is planar manipulators toolbox with SD/FAST [14]. The distinction from the previous one is that the dynamic model is calculated SD/FAST library. The "Robotics Toolbox" [15] provides a lot of functions such as kinematics, dynamics and trajectory generation. It is useful both for simulation and for analyzing results from experiments with real robots, so it can be a powerful educational tool. "SimMechanics toolbox" [16] extends Simulink with the facilities for modeling and simulating mechanical systems specifying bodies and their mass properties, possible motions, kinematic constraints. With SimMechanics it is possible to initiate and measure body motions. All four technologies allows to easily build the robotic system. One of the differences of the special toolboxes is that they include more specific predefined functions comparing with the general one.

Systems such as Dynamola, Modelica or 20-sim provide similar possibilities for robot system simulation. The robot system is built by connecting blocks representing different robot parts like link bodies or joints. Figure 1 shows the block scheme of a complete model of a KUKA robot in Modelica.

Robotica is a computer aided design package for robotic manipulators based on Mathematica [17]. Robotica is intended, first of all, for model generation and analysis of robotic systems and for simulation.

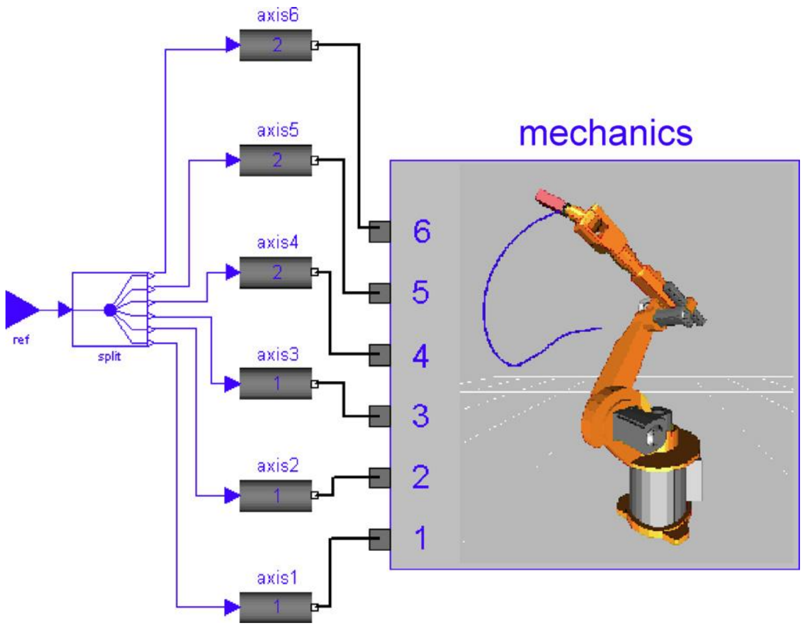


Figure 1. Simulation of a robot with Modelica [1].

2.2.2 3D robot simulators

3D modeling and rendering of a robot and its environment are some of the most popular applications for robotics simulators. Modern 3D simulators include a physics engine for more realistic motion generation of the robot. Robots can be equipped with a large number of sensors and actuators. Modern simulators provide also various scripting interfaces. The examples of wide-spread script languages are URBI, MATLAB, Python.

There exist numerous simulators which are targeted for mobile robot navigation, but this section will concentrate on the simulation software tools which can be used for manipulation and, therefore, can be applied within this research. Development and testing of new algorithms, modeling of the environments and robots, including different kinds of sensors and actuators are the key aspects the manipulation and grasp simulators have to cope with. For these reasons, the most prominent toolkits in this area GraspIt!, OpenGRASP and OpenRAVE were chosen to be reviewed.

GraspIt! is an open-source interactive grasping simulation environment, elaborated, primarily, for grasp analysis [18]. On one hand, it can be used as a development tool, to execute and test various robot control algorithms. On the other hand, it can be used as a computational platform that maintains a robot that does operate in the real world. Since, GraspIt! can be applied for both grasp planning and robotic hand design, several research groups are already using it. For example, the Robonaut group at NASA Johnson Space

Center and researchers in the Robotics and Mechatronics group at Rutgers University applied GraspIt! for testing their robotic hands [19].

Another existing publicly available cross-platform software architecture is OpenRAVE — the Open Robotics and Animation Virtual Environment [20]. It has been elaborated for real-world autonomous robot application in order to perform a simple integration of 3D simulation, visualization, planning, scripting and control of robot systems.

OpenGRASP is a simulation toolkit, which extends the OpenRAVE functionality towards the realization of an advanced grasping simulator [21].

All these simulators are applicable to robotic grasping and manipulation, as they possess a number of features, which make them powerful simulation tools. The most significant functionalities as well as some benefits and drawbacks of these tools are listed below. These parameters are chosen as categories for the simulators comparison.

Robot models

- GraspIt! includes a library of several hand models, including a simple model of the human hand. GraspIt! allows easily import new robot designs due to the flexible robot definition. Moreover, it is possible to attach multiple robots, defining a tree of robots, to create robotic platforms.
- OpenRAVE supports a variety of different kinematic structures for robots. Providing implementations for various classes of robots enable users to better exploit their structures. In addition to the similar to GraspIt! functionality it enables the development of virtual controllers for numerous robot arms models.
- Additionally to OpenRAVE models in OpenGRASP a model of the Schunk PG70 parallel jaw gripper was created. Furthermore, the development of the modeling tool Robot Editor, which main goal is to facilitate modeling and integration of numerous popular robot hands, has been started.

Robot files formats

- Starting from version 2.1, GraspIt! uses an XML format for storing all of its data. In general, a Robot configuration file contains the following data: a pointer to the Body file containing the palm and also information about DOF and kinematic chains.

- In OpenRAVE robots are standardized with the COLLADA 1.5 file format. COLLADA [22] is an open, extensible XML-based format, that supports the definition of kinematics and dynamics and provides a possibility to convert to and from other formats. Since version 1.5 the standard contains constructs, which are very useful to describing kinematic chains and dynamics. Thus, OpenRAVE defines this format for storing information like manipulators, sensors, and collision data. Moreover, OpenRAVE defines its own format to help users quickly get robots into the environment, because COLLADA is a rather difficult format for edition. The conversation between these formats are completely supported.

- OpenGRASP also uses COLLADA file format, but extends it with some original OpenRAVE definitions in order to support specific robot features like sensors and actuators. These additions are hidden inside the simulator. This fact guarantees the compatibility.

Environment building

All three simulators provide a possibility of building complex environments with imported obstacle models.

Architecture

- GraspIt! relies on several components to run smoothly and efficiently. It includes components which perform main operations. There are assembling geometric link models of the chosen hand, locating contacts, evaluating grasp quality, and producing wrench space projections. Figure 2 represents an overview of GraspIt! components.

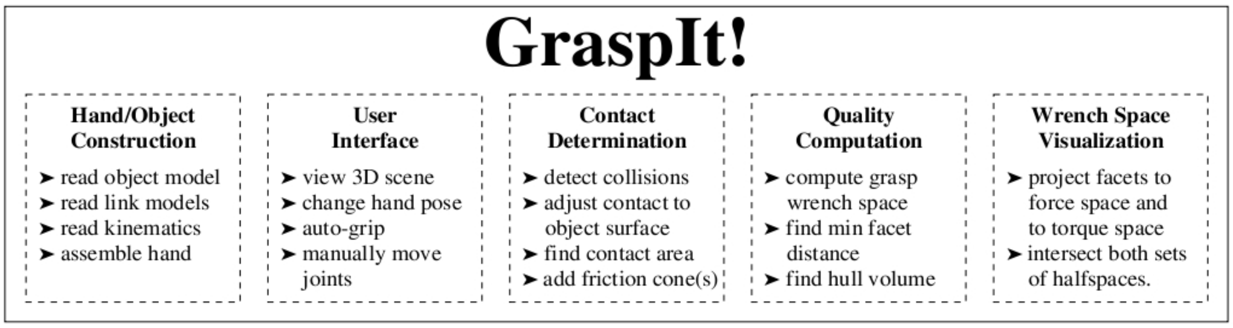


Figure 2. The internal components of GraspIt! and their functions [18].

- Plugin-based architecture of OpenRAVE, described in figure 3, allows its extension and further development by other users. It consists of three layers: a core, a plugin layer for interfacing to other libraries, and scripting interfaces for easier access to functions. One of GraspIt! disadvantages is its rather monolithic architecture. This fact restricts the possibilities to add new functionalities and complicates the integration with third frameworks.

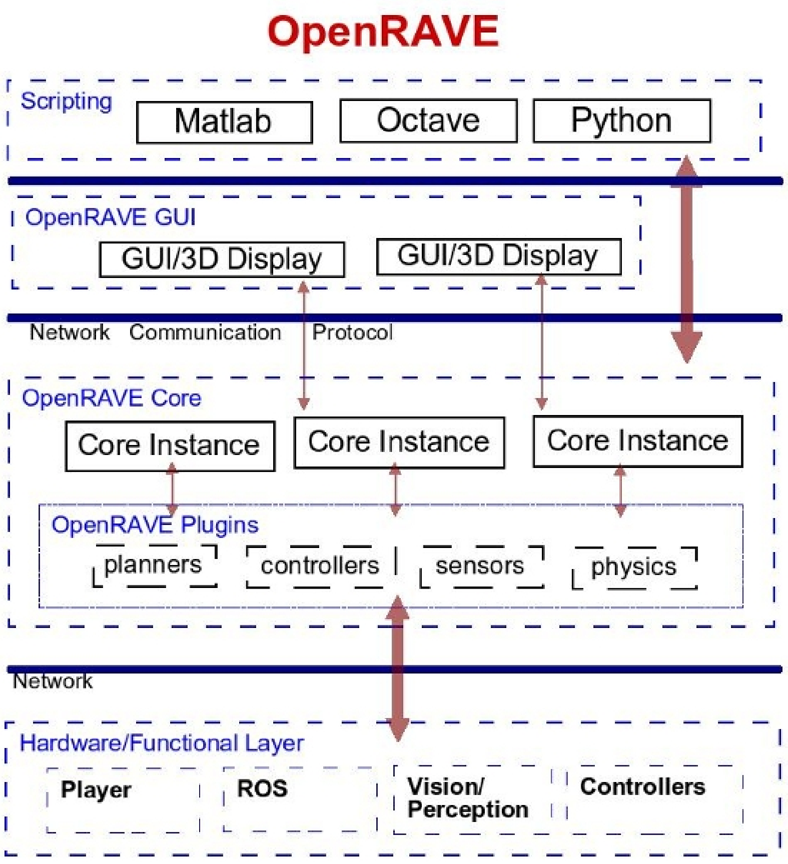


Figure 3. The OpenRAVE architecture is divided into several layers: the scripting layer, the GUI layer, the Core OpenRAVE layer, and the plugins layer that can be used to interact with other robotics architectures [20].

- Basically, OpenGRASP core is an improved version of OpenRAVE. Some extra plugins, which enrich grasp simulation functionality were designed. The most important among them are new sensor plugins and a plugin interface of actuators ActuatorBase (as OpenRAVE does not include actuator models).

User interface

- An interactive GraspIt! user interface implemented with Qt library allows to port the code to different operating systems. Currently, both Linux and Windows system interfaces are supported. Visualization methods are capable to show the weak points of the grasp and create arbitrary 3D projections of the 6D grasp wrench space.
- A GUI can be optionally attached to OpenRAVE core to provide a 3D visualization of the environment. At the moment OpenRAVE uses Coin3D/Qt to render the environment.
- OpenGRASP enriched OpenRAVE visualization capabilities by communication with Blender, which is one of the base technique of a Robot Editor. More than that, as OpenRAVE and Blender both provide Python API, Blender can be used not only for visualization, but also for calling planners, controlling robots and editing robot geometry.

Network interaction and scripting

- In addition to direct user interface, GraspIt! also supports the communication with external programs via TCP connection and simple text protocol. Moreover, a preliminary MATLAB interface, using mex-files for sending commands and queries to simulator was created. The interaction with MATLAB makes possible an implementation of dynamic control algorithms.
- In OpenRAVE implementation, network commands are sent through TCP/IP and are text-based. Text-based commands are easy for data interpretation and directly maintain various scripts. Currently, OpenRAVE support Octave/MATLAB, Python and Perl scripting environments. The most powerful tool is a Python API, using for scripting demos, which makes it simple to control and monitor the demo and environment state. The scripting environment can communicate with multiple OpenRAVE instances at once. At the same time, OpenRAVE can handle multiple scripting environments interacting with the same instance simultaneously (figure 4).
- OpenGRASP provides same network communication possibilities as OpenRAVE, due to the fact that OpenRAVE is a base of OpenGRASP. Thus, its scripting layer handles network scripting environments like Octave, Matlab and Python, which communicate with the core layer in order to control the robot and the environment.

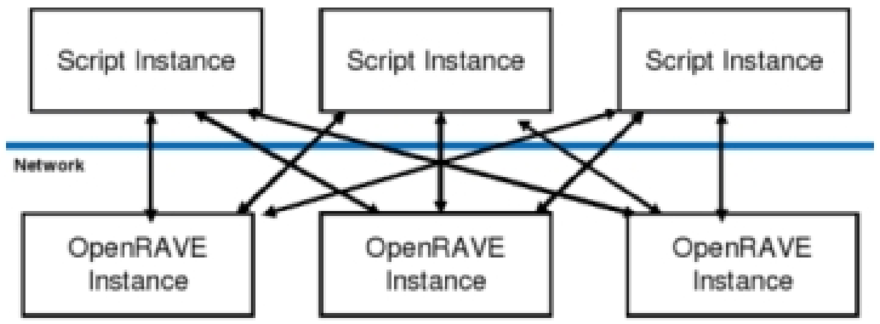


Figure 4. Multiple OpenRAVE and script instances interactions through the network [20].

Hence, another shortcoming of GraspIt! is a lack of convenient Application Programming Interface (API) for script programming.

Physics engine

- GraspIt! incorporates its own dynamic engine for robot and object motion computations considering the influence of external forces and contacts.
- The OpenRAVE physics engine interface allows to choose the concrete engine for particular simulator run. The ODE (Open Dynamic Engine) implementation is provided by the oderave plugin.
- OpenGRASP plugin palrave uses the Physics Abstraction Layer (PAL) [23]. This layer provides an interface to different physics engines and a possibility to dynamically switch between them. Therefore, palrave plugin release the need for creation of separate plugins for each engine.

Collision detection

- GraspIt! system performs real-time Fast collision detection and contact determination system, based on Proximity Query Package [24], that allows a user to interactively manipulate a robot or an object and create contacts between them.
- Although OpenRAVE is not tied to any particular collision checker, physics engine has an interface to implement different collision checkers. Each one of them has to be created as a separate plugin. In installation instructions it is proposed to use either ODE Collision, Bullet Collision or PQP Collision library.

- Similarly to physics engine, OpenGRASP palrave plugin gives a possibility to decide what collision checker to use, depending on specific environment and use.

Manipulation and grasping capabilities

- A set of grasp quality and stability metrics for grasp evaluation on the fly is incorporated into GraspIt! The simulator also provides a simple trajectory generator and control algorithms that compute joint forces needed to follow the trajectory.
- OpenRAVE not only performs similar computations for grasping analysis, but includes path planning algorithms to solve more general planning and manipulation tasks.
- By combining the automatically generated grasp sets, inverse kinematics solvers and planners, provided by OpenRAVE, the robots developed in OpenGRASP RobotEditor are able to manipulate various objects in their environments.

Sensor simulation

- GraspIt! does not possess sensor simulation.
- OpenRAVE identifies Sensor and SensorSystems interfaces. The fist one allows to attach a sensor, like a range finder or a camera to any part of a robot. The second interface is responsible for the update of object pose estimates based on sensors measurements.
- Additionally to OpenRAVE capabilities two sensor plugins, that can be used mainly for antromorphic robot hands, were developed within OpenGRASP simulator.

Initially, for the application and experiments implementation GraspIt! simulator was chosen. But in progress, there occurred the problems associated with the MATLAB - GraspIt! interaction. GraspIt! does not include enough MATLAB functions for step simulation collision analysis. As a result, it was decided to use OpenRAVE as a simulation tool in this work. Regarding robot grasping manipulation, OpenRAVE provides similar functionality to GraspIt!, but in contrast to GraspIt!, it enables the sensor simulation and the development of virtual controllers for numerous robot arms models. Moreover, its open component architecture permits the robotic research community easily exchange and compare algorithms. It can help in standardization of formats and paradigms development. OpenRAVE is still a project in progress. The main development areas are standardization of the network protocol, parallel execution of planners and real-time control to increase the performance, further experiments with monitors and execution models and the ways to integrate them within OpenRAVE. The last improvements are vital for failure recovery and robustness in uncertain conditions. Unlike GraspIt!, OpenRAVE incorporates a large set of functions which allow to obtain information about robot and object parameters like link transformations or information about DOF and at the same time send commands to the simulator over the network.

3 WORLD MODELS AND PERCEPTION OF THE WORLD

3.1 World models in intelligent systems

A world model is a key component of any intelligent system. The construction of models of the environment is crucial to development of several robotics applications. The world model should build an adequate representation of the environment from sensor data. It is through these environment models the robot can adapt its decision to the current state of the world. However, there are exist some challenges associated with the construction of such models [25]. The primary challenge is the presence of uncertainty in the real world because of its dynamic nature. The representation has to cope with the uncertainties and update its states. More problems are created by the fact that the uncertainty presents in both sensor data and to the robot's state estimation system. In addition, environment models should be compact. This means that the model can be efficiently used by other components of the system, like path planners [26].

The world model can be considered as a knowledge base with the internal representation of the external world [25]. Figure 5 demonstrates how the world model fits into a generic robot control architecture. Allen in [27] suggested criteria for world model design. These are computability from sensors, explicit specification of features, modeling simplicity and easy computability of attributes. A lot of studies in world modeling area were caused by the increased interest in autonomous mobile systems [25]. Unfortunately, most of proposed models were designed for navigation and obstacle avoidance tasks and are useless for robotic manipulation applications. Most world models contain only geometric information in different representation. The location, size, and shape of objects in the environment are certainly crucial parts of a world-model. However, inclusion of only geometric data is very limiting in a model designed to support robotic manipulation. The information about contacts as well objects mass distribution, compliance and friction properties can improve grasping computations [28]. In this section some world-modeling techniques and their ability to support manipulator control are discussed.

3.2 Spatial Semantic Hierarchy

Kuipers in [29] proposes a model, named the Spatial Semantic Hierarchy (SSH), for both the human cognitive map, and as a method for mobile robot exploration and mapping. The SSH consists of five different levels of representation of the environment, arranged in

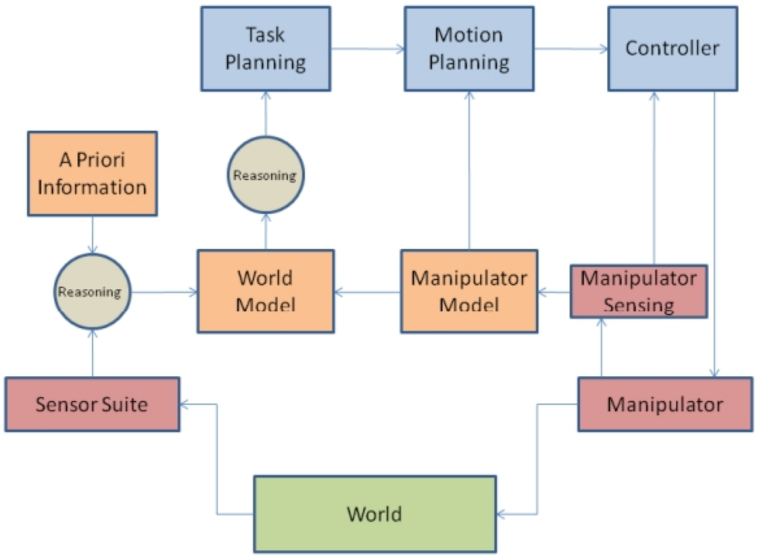


Figure 5. Robot Control Architecture [28].

a hierarchy, where higher levels depend on lower levels. Figure 6 shows levels, the types of information in each level, and some relationships between parts of the model. The sensory level is the interface to the robot's sensor suite. The control level describes a set of control laws that determine the robot's behavior in the environment. The causal level can be considered as the planning level, where appropriate control laws are chosen and passed down to the control level. The topological level handles the relationships between places, paths, and regions. The types of these relationships may be connectivity, order, or containment. The metrical level represents a global geometric map of the environment. This level applies mostly to mobile robotics where it is useful to relate several local frames of reference to a global frame of reference.

The hierarchy presented in Figure 6 is significantly different from the schematic control system presented in Figure 5. Since the SSH was developed primarily for mobile robotics, it does not explicitly support the idea of contact-interaction. It works well for mobile system control and mapping, but it is not designed for manipulation purposes. However, the idea of capturing topology as well as geometry is important to both mobile robots and robotic manipulation. For example, for non-contact interaction, where the existence of a collision-free path can be determined with only very coarse geometric information. If this topological relation between manipulator states were detected in the world model, significant computation time could be saved by not recomputing the path between the two states each time.

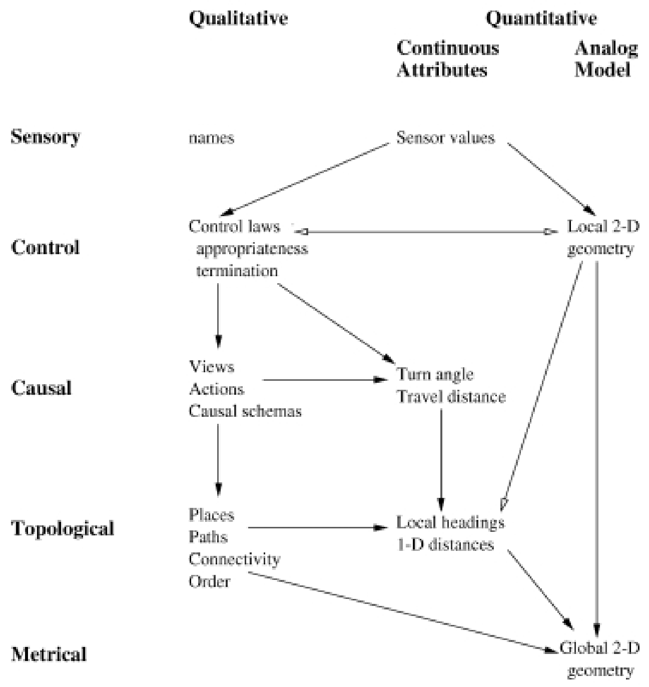


Figure 6. The distinct representations of the Spatial Semantic Hierarchy. Closed-headed arrows represent dependencies; open-headed arrows represent potential information flow without dependency [29].

3.3 Feature Space Graph Model

Merat and Hsianglung's paper [30] presents a Feature Space Graph world model, which describes objects by the hierarchy of their features such as points, patches, corners, edges, and surfaces (Figure 7). The process of creating the model can be expressed in following stages. First, low level features are extracted from sparse range vision data and the partial object description is generated from these data. After that, Feature Extraction by Demands (FED) algorithm is applied. As a more detailed object description is formed, FED converges from bottom-up image processing to top-down hypothesis verification to end up with a complete hierarchical object definition.

The main benefit of this model is that it does not require prior knowledge. In addition, it is conceptually rather simple and fast to update. This model can be applied in object recognition applications, which can be subtasks in a global manipulation task.

Face-to-face composition graph model

Another graph-based world model – Face-to-Face Composition (FFC) graph, was proposed by De Floriani and Nagi [31]. It can be thought as a formal representation of

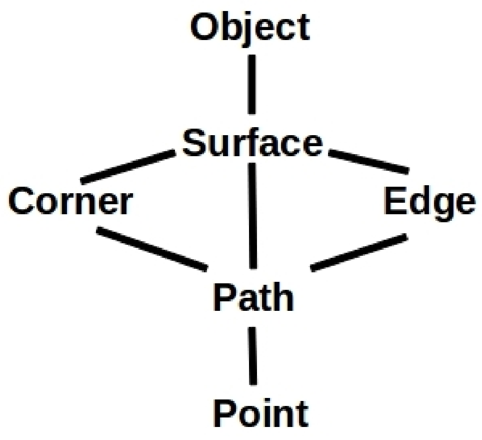


Figure 7. Hierarchical Feature Space for Object Description.

a family of models of solid objects. Its multi-rooted hierarchical structure is based on boundary representation and unites various conceptual views of the same object. Each node of the graph describes the topology and geometry of a volumetric component of object. Each volumetric component has to be composed of one shell, where the shell is any maximal connected set of faces on the bounding surface of an object. In its turn, each branch of FFC graph connects faces between two components.

This model differs from other graph-like models due to a number of reasons. A primary reason is the possibility to set, either by the user or an algorithm, an arbitrary but valid partial order on object components. More than that, it provides a flexibility in the representation of single-shell components. It makes sense to use this extensible model in robot manipulation tasks.

Multiple world model

Unfortunately, there is no universal world representation which fits all possible robotic problems. The natural approach to use a combination of several models was suggested by Allen [27]. The particularity of this scheme is the use of multiple shape representations for objects. The choice of a specific model depends on the type of the used sensory system. So, the basic idea of this approach is to use the most suitable model for each sensor and then join these independent components in order to receive a full world picture. Sensors can work either independently or in parallel. They can share the information, not affecting each other. Furthermore, the same data can be collected and processed by several sensors, which makes the resulting model more accurate. This well extensible and rather easily implementable approach can be tuned for different applications, including manipulation area.

In conclusion, one can mention that despite the existence of numerous world models, most of them are dedicated to mobile robots and mainly for navigation. There are few environment representations that are designed exclusively for robot manipulation. A considerable part of suitable models are hierarchical or graph-based.

3.4 World models in simulation

Simulators should provide world model definitions similar to the reality. The main goal is to guarantee that the robot will execute the task in the most efficient way given such a world definition. Different simulators provide their own world models.

The simulation world in Graspit! consists of different world elements presented on Figure 8. The world elements are either rigid bodies or robots, which impose restrictions on the motion of these bodies relative to each other [18].

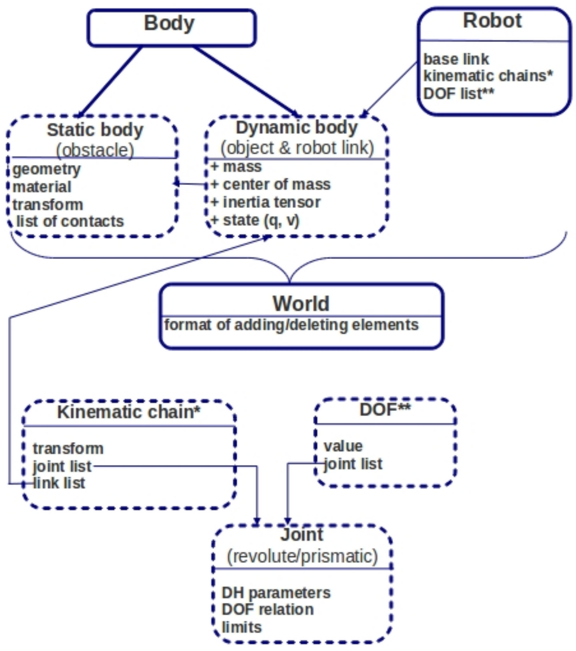


Figure 8. The simulation world in GraspIt!

Different types of bodies and flexible definition for robot make possible to model complex articulated robots and construct robotic platforms consisting of multiple robots. A basic body is described by its geometry (in Inventor scene graph), material specification, list of contacts and transformation that specifies the body's pose relative to the world coordinate frame. The dynamic body includes basic body's properties and extends them with the mass, inertia tensor definitions and the location of the center of mass (COM) relative to the body frame. It also specifies the body's dynamic state parameters: the pose and velocity of a body frame located at the COM relative to the world coordinate system. Two types of dynamic bodies, robot links and graspable objects, are defined in GraspIt!

A robot consists of a base link, a number of kinematic chains and a list of its degrees of freedom (DOF). Each kinematic chain in turn represents a set of lists of its links and joints and a transform locating its base frame with respect to the robot's base frame.

OpenRAVE has its particular features of world representation. Diankov in his thesis [32] introduces the OpenRAVE architecture. The key characteristic of OpenRAVE that distinguishes it from other simulation packages is the idea of application of various algorithms with minimal scenario modifications. Thus, users can concentrate just on the development of planning and scripting aspects of a problem without explicitly affecting the details of robot kinematics, dynamics, collision detection, world updates, sensor modeling, and robot control.

An interface is the base element for OpenRAVE functionality extensions. The main interfaces which form the world model are:

- Kinematic body interface. A kinematic body is a basic object in OpenRAVE. It consists of a number of rigid-body links connected with joints. The interface provides a possibility to set and get angle values for joints and transformations for all body's links. The kinematics is just a graph of links and joints without specific hierarchy.
- Robot interface. A robot is a special type of kinematic body with additional higher level functionality for its control and movement in the environment and its interaction with other objects. These extra features are:
  - A list of manipulators describing the links participated in manipulation of objects of the environment. Usually manipulators consist of a serial chain with a base link and an end effector link. Each manipulator is also decomposed into two parts: the hand contacting the objects and the arm transferring the hand into the target position.

- Active DOF. OpenRAVE allows to determine a set of active DOF, which will be used in a particular task.
- Grabbing bodies. It is possible to attach a body to one of robot's links in such a way that it temporarily becomes a part of the robot and moves simultaneously with this link. During this state all collisions between the robot and the object are ignored. This functionality is one of the key aspects for manipulation planning.
- Attached sensor. Any number of sensors can be attached to the robot's links. In this case, all sensor transformation parameters are added to robot parameters set.

To create more realistic environmental conditions, OpenRAVE includes some other interfaces, like a collision checker interface, which provides a wide range of collision and proximity testing functions. Similar to collision checkers, a physics engine can be set for the environment to model the interaction of the objects in the simulation thread. It allows object to maintain velocities and accelerations and it affects the world state only inside the simulation step.

3.5 Sensors used for manipulation

The previous subsection was concentrated on different ways of world modeling, but another equally important aspect are the information sources that can be used to acquire knowledge about the environment. The control of a robotic system would be relatively simple task if a complete world model is given. However, in practice a perfect world model is not available. Perception with sensors is a way to compensate the lack of prior information. Perception provides the data needed for updating environment and robot system states. This chapter surveys different types of sensors and focuses especially on tactile sensors, which were used in order to update the information about the environment in this study. Moreover, vision sensors are also reviewed, as they are widely used to solve various manipulation tasks.

3.5.1 Sensors classification

For the purposes of the discussion sensors can be classified to proprioceptive and exteroceptive. Proprioceptive sensors measure values to recover the state of the robot itself.

Motor speed or robot arm joint angles are examples of such values. Exteroceptive sensors, in contrast, acquire information from the environment. For example, distance measurements or light intensity are among them. Namely exteroceptive sensors are responsible for the extraction of meaningful world features. Another important categorization is a division into passive and active groups [33, pp. 89–92].

Passive sensors measure the external environment energy, which enters them. For example, microphones, CCD or CMOS cameras are passive sensors. Conversely, active sensors emit energy by themselves and measure the environmental reaction. Active sensors are able to handle more controlled interactions and often achieve better performance. On the other hand, the emitted energy can significantly affect the parameters that sensor is trying to measure. Another risk is a possibility of interference between sensor's signal and external signals, which are not controlled by it.

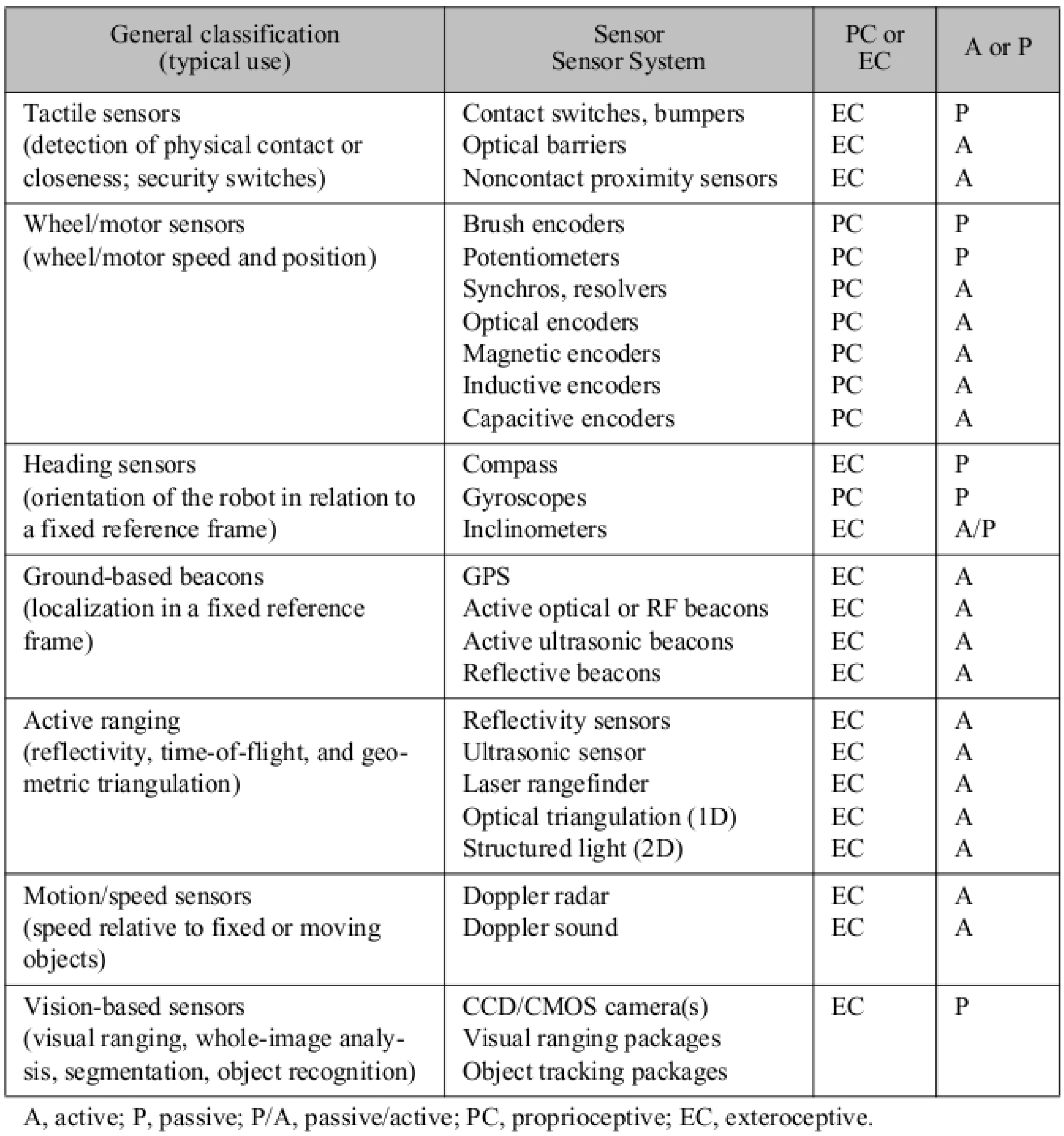
Table 9 shows the classification of sensors according to their sensing objective and method. This study pays attention to the fist class of tactile sensors, as the world model is updated based on the measurements of contact sensors.

3.5.2 Vision-based sensors

Vision-based sensors occupy a special place among all sensors. Vision is the most powerful human sense, which provides an enormous amount of information about the environment and supports an intelligent interaction in dynamic environments. These capabilities substantiate the process of design robot sensors, which provide functions similar to human vision system. Two main technologies underlie the creation of vision sensors: CCD (Charged Coupled Device) and CMOS (Complementary Metal Oxide Semiconductor). They both have some limitations compared to the human eye, because these chips both have far poorer adaptation, cross-sensitivity, and dynamic range. As a result, the current vision sensors are not very robust [33, p. 117].

Visual ranging sensors

Range sensors provide distance information. They are very important in robotics, as they generate the data using for successful obstacle avoidance. There exist a number of technologies to obtain depth information. Ultrasonic, laser rangefinder, optical rangefinder are among them [33, pp. 117–145]. Visual chips are also used in implementation of ranging functionality. This is a challenging process, due to the loose depth information after



General classification (typical use)	Sensor Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers	EC	P
	Optical barriers	EC	A
	Noncontact proximity sensors	EC	A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A/P
Ground-based beacons (localization in a fixed reference frame)	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensor	EC	A
	Laser rangefinder	EC	A
	Optical triangulation (1D)	EC	A
	Structured light (2D)	EC	A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision-based sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s) Visual ranging packages Object tracking packages	EC	P

A, active; P, passive; P/A, passive/active; PC, proprioceptive; EC, exteroceptive.

Figure 9. Classification of sensors frequently used in robotics [33].

projection of 3D world into 2D plane. The general solution is to recover depth by looking at several images of the scene to obtain more information and be able to recover, at least, partial depth. These images should be, first of all, different, so that taken together they provide additional information. One way is to have images from different viewpoints, which is the base of stereo and motion algorithms. An alternative is to create different images by changing the camera geometry, such as the focus position or lens iris. The last approach is a part of depth from focus and depth from defocus techniques. One more way to calculate the depth and surface information of objects in the scene is structured light. This process is based on projection of a known pattern of pixels (often grids or horizontal bars) on to a scene [34]. This technique is used in structured light 3D scanners, such as MS Kinect [35].

Approaches, in which the lens optics are varied in order to maximize focus, in general, are called depth from focus. This is one of the simplest visual ranging techniques, as the sensor when focusing just moves an image plane until maximizing an object sharpness. The position of image plane, corresponding to the maximum sharpness, determines range. One of the most common method for sharpness evaluation is a calculation of subimage intensity gradient. A drawback of the technique is slowness. The technique is an active search method and time is needed to change the focusing parameters of the camera.

Depth from defocus is a way to recover the depth using a series of images that have been taken with different camera geometries. The method uses the direct relationships among the scene properties (depth and irradiance), camera geometry settings and the amount of blurring in images to derive the depth from parameters which can be directly measured. The basic advantage of the depth from defocus method is its extremely fast speed, because the equations which describes the relationship between all parameters do not require search algorithms to find the solution. More than that, in contrast to depth from stereo methods, it does not require images from different viewpoints, therefore occlusions or disappearance of object in a second view do not affect the result [33, pp. 122–129].

Stereo vision is an example of technique applied to recover the depth information from several images taken at different perspectives [36, pp. 523–526]. The fundamental concept is triangulation. A scene point and two camera points form the triangle. Knowledge about the baseline between cameras as well as the angle between camera rays allow to recover the distance to the object. Most of the difficulties in applying this technique in robotic applications arise in finding matches for pixels in two images corresponding to one scene point.

Color-tracking sensors

An important aspect of vision-based sensing is that the vision chip can provide sensing possibilities that no other sensor can provide. One such novel sensing modality is detecting and tracking color in the environment [33, pp. 142–145]. Color is an environmental characteristic that is orthogonal to range and can be a source of new information for robot. As an example, the color can be useful both for environmental marking and for robot localization. Color sensing has two important advantages. First, detection of color can be directly derived from a single image, which means that the correspondence problem does not take place in such algorithms. Furthermore, because color sensing provides a new, independent environmental signal, in combination with existing signals, such as data from stereo vision or laser rangefinding, can provide significant information payoffs.

3.5.3 Tactile sensors

Robot grasping and manipulation of objects is one of the robotics fields most affected by the uncertainties in the real world, as robot manipulators interact with objects, for which physical properties and location are unknown and can vary [37].

In robotics, visual information is a rich source of information about the environment and task. However, vision and range sensor estimations often incorporate some residual uncertainty. In its turn, tactile sensors can give highly reliable information about unknown objects in unstructured environments [38]. By touching an object it is possible to measure contact properties such as contact forces, torques, and contact position. From these, object properties such as geometry, stiffness, and surface condition can be estimated. This information can then be used to control grasping or manipulation, to detect slip, and also to create or improve object models. In spite of this, studies on the use of tactile sensors in real-time control of manipulation have begun to appear just in the last few years.

Initially, studies in tactile sensing area focused on the creation of sensor devices and object recognition algorithms [39]. There exist several aspects which make tactile sense a key to advanced robotic grasping and manipulation [40]. First of all, only tactile sensors can provide a reliable information about the existence of a contact with an object. Secondary, tactile information can inform on the contact configuration. The type of the contact can be defined by force and torque sensing. More than that, contact data can be used as feedback for control.

Over the years, many sensing devices have been developed for robotic manipulation. An outline of a robot hand with some of the most common types of contact sensor is depicted in Figure 10. The description of these sensors is given in Table 2.

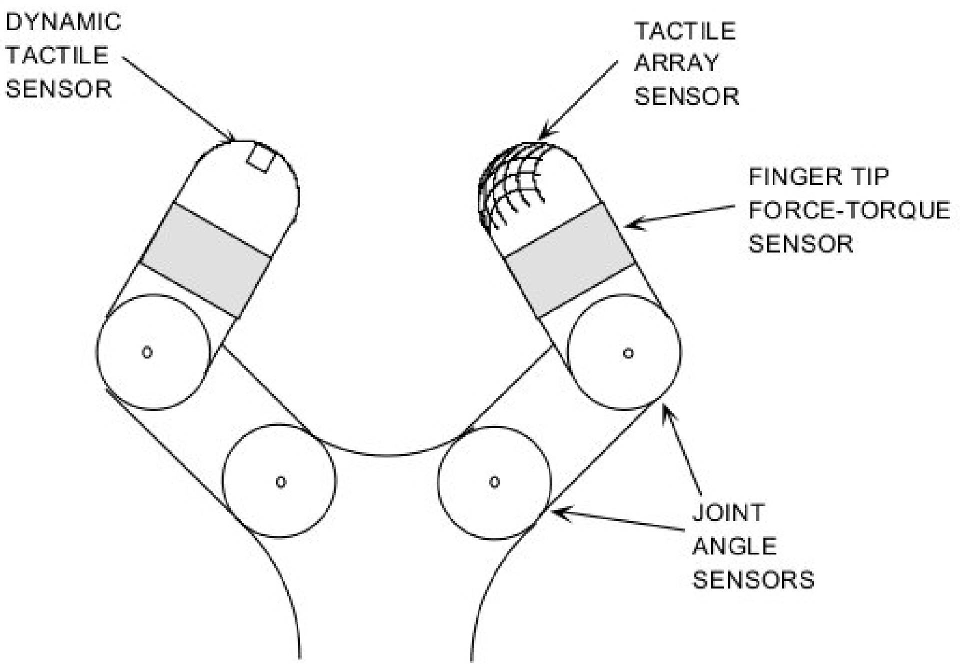


Figure 10. Schematic drawing of a robot hand equipped with several types of contact sensor [38].

Table 2. Contact sensors parameters [38].

SENSOR	PARAMETER	LOCATION
Tactile array sensor	pressure distribution, local shape	in outer surface of finger tip
Finger tip force-torque sensor	contact force and torque vectors	in structure near finger tip
Finger joint angle sensor	finger tip position, contact location	at finger joints or at motor
Actuator effort sensor	motor torque	at motor or joint
Dynamic tactile sensor	vibration, stress changes, slip, etc.	in outer surface of finger tip

Contact sensors can be divided into extrinsic and intrinsic groups [40]. The first class measures forces that act upon the grasping mechanism. Intrinsic sensors measure forces within the mechanism.

Manipulation of an unknown object with rolling, sliding, and regrasping is a task, that certainly requires a great deal of contact information. Uses of touch sensing in manipulation are shown in Figures 11 and 12.

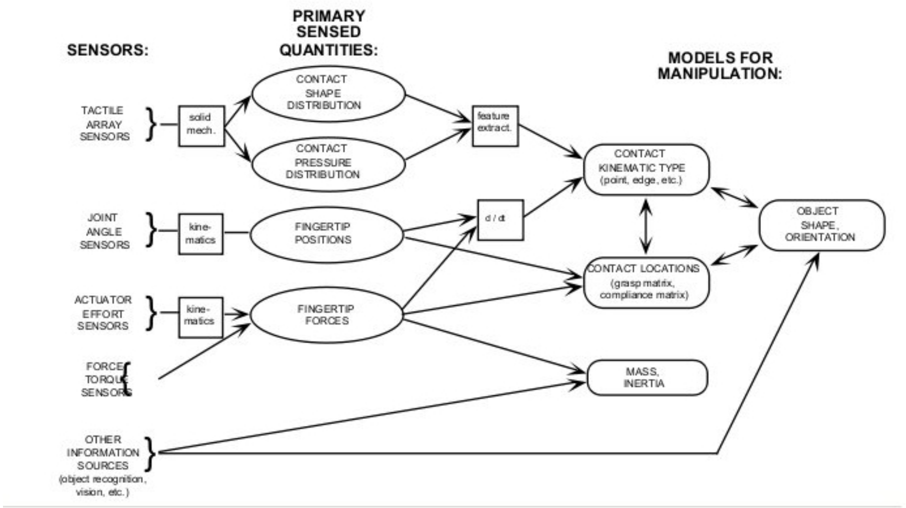

SENSORS:
PRIMARY SENSED QUANTITIES:
MODELS FOR MANIPULATION:
TACTILE ARRAY SENSORS
solid mech.
CONTACT SHAPE DISTRIBUTION
CONTACT PRESSURE DISTRIBUTION
feature extract.
CONTACT KINEMATIC TYPE (point, edge, etc.)
OBJECT SHAPE, ORIENTATION
JOINT ANGLE SENSORS
kine-matics
FINGERTIP POSITIONS
d / dt
CONTACT LOCATIONS (grasp matrix, compliance matrix)
ACTUATOR EFFORT SENSORS
kine-matics
FINGERTIP FORCES
FORCE TORQUE SENSORS
MASS, INERTIA
OTHER INFORMATION SOURCES (object recognition, vision, etc.)

Figure 11. Uses of touch sensing in manipulation: geometric information [41].

As can be seen in Figure 11, sensed quantities derived from the sensor data can be used to update models of the geometric aspects of the manipulation task, such as grasp configuration, object shape or contact kinematics [41].

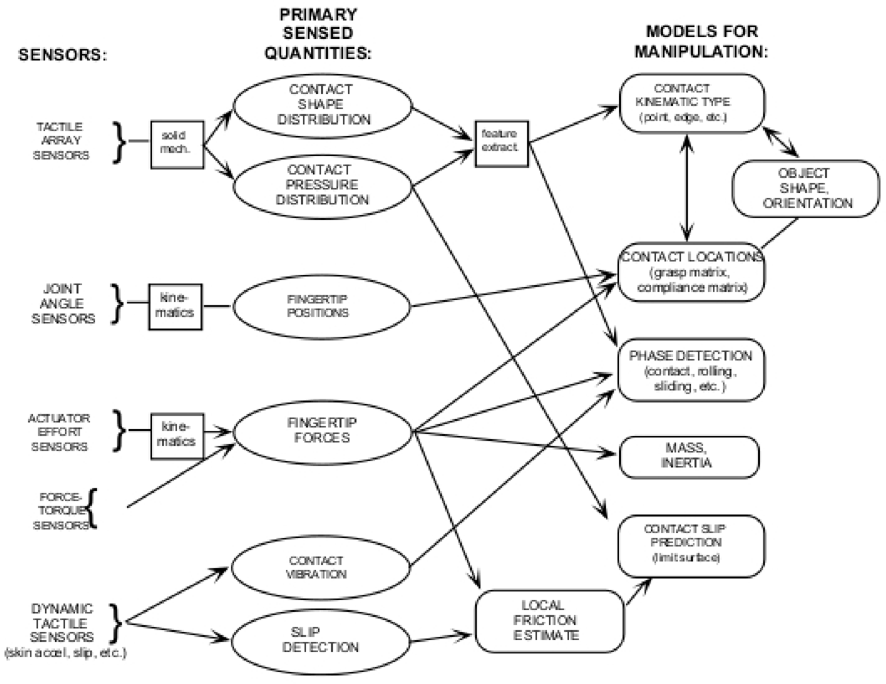

SENSORS:
PRIMARY SENSED QUANTITIES:
MODELS FOR MANIPULATION:
TACTILE ARRAY SENSORS
solid mech.
CONTACT SHAPE DISTRIBUTION
CONTACT PRESSURE DISTRIBUTION
feature extract.
CONTACT KINEMATIC TYPE (point, edge, etc.)
OBJECT SHAPE, ORIENTATION
CONTACT LOCATIONS (grasp matrix, compliance matrix)
JOINT ANGLE SENSORS
kine-matics
FINGERTIP POSITIONS
PHASE DETECTION (contact, rolling, sliding, etc.)
ACTUATOR EFFORT SENSORS
kine-matics
FINGERTIP FORCES
MASS, INERTIA
FORCE-TORQUE SENSORS
CONTACT SLIP PREDICTION (limit surface)
CONTACT VIBRATION
DYNAMIC TACTILE SENSORS (skin accel, slip, etc.)
SLIP DETECTION
LOCAL FRICTION ESTIMATE

Figure 12. Uses of touch sensing in manipulation: contact condition information [41].

In Figure 12 sensed quantities are used to update models of contact conditions, such as local friction, slip limits, contact phase (making and breaking of contact, rolling, sliding) [41].

4 WORLD MODEL UPDATE

In this study world state means knowledge about the object position. Existence of uncertainty in this knowledge may lead to the fail in the execution of given manipulation task. Thus, in order to finally complete the task the updated information about the object position is needed. As a result, one has to maintain the world model updating based on sensory feedback.

This section is focused on the problem of robot manipulation under uncertainty. Several models of uncertainty representation are discussed, as well as methods, which applied to decrease this uncertainty. Most attention is given to the Steepest Descent algorithm for world model update in order to determine the exact location of an grasped object. This optimization method, based on motion in the direction of negative gradient, was chosen for implementation in this work. Moreover, the attention is also paid to approaches allowing to choose the length of each optimization step, because this choice affects the speed of convergence to an optimum. Golden search procedure as one of common techniques to calculate the step size is explained in this chapter.

4.1 Representing uncertainty

Uncertainty is one of central problems when executing manipulation and especially grasping tasks. Particularly, the initial state of the environment is not exactly known. There are different ways to represent the existing uncertainty. One way is to treat actions as transitions between sets of world states and then find a plan that succeeds for each state in the initial set. Some models are based on the set of possible states. They are known as possibilistic models [42]. Another way is to model uncertainty with a probability distribution. This is a probabilistic approach. This classification was stated by Goldberg in [42].

More mechanical designer's treatment of uncertainty is to determine the worst-case "tolerance"boundaries for each element–state and thereby guarantee the execution of the whole set when tolerances of all components are satisfied. In most cases, planners tuned by possibilistic models transfer this approach on planning algorithms. In other words, planners try to find a plan which ensures the execution of a specific goal for any possible initial state. The main problem of this approach is the inability to find a plan which is guaranteed to succeed. Sometimes due to model errors finding such a plan is impossible.

Probabilistic models of uncertainty are commonly used in industrial automation applications. Performance criteria optimization strategies are treated both in decision theory, where the optimal strategy is a function of sensor data, and stochastic optimal control theory, which considers strategies as an additive, often Gaussian noise. These models are used to combine the data from multiple sensors.

In conditions of unstructured environment, planning can be replaced by exploratory procedures like contour following [43, 44]. This model is useful, for instance, when interaction can not be repeated.

4.2 Sensor uncertainty

Sensors are imperfect devices, which incorporate systematic and random errors. As they are main sources of information that form the environmental model, they also add a component into the uncertainty about the world state. Uncertainty of sensor readings can be represented in three forms based on the information about the uncertainty [45]. In following notation a is some real-valued attribute.

- Point Uncertainty is the simplest model, which assumes an absence of uncertainty associated with data at all. Measured value is supposed to be equal to real one.
- Interval Uncertainty. Instead of an exact sensor value an interval, denoted by $U(t)$, is considered. Objectively, $U(t)$ is a closed interval $[l(t), u(t)]$, where $l(t)$ and $u(t)$ are real-valued functions of t, bounding the value of a at time t. Thus, imprecision of data is expressed in the form of an interval. As an example, $U(t)$ is an interval restricting all values within a distance of $(t - t_{update}) \times r$ of a, where t_{update} is the time of a last update, and r is the current rate of change of a. Therefore $U(t)$ increases linearly with time until the next update of a is received.
- Probabilistic Uncertainty model is proposed by Cheng, Kalashnikov and Prabhakar in [46]. Compared with interval uncertainty, it requires also the probability density function (pdf) of a within $U(t)$. This function is denoted by $f(x, t)$. This function should have specific properties: $\int\limits_{l(t)}^{u(t)} \{f(x,t)\}dx = 1$ and $f(x,t) = 0$ if $x \notin U(t)$. An exact form of $f(x, t)$ is application-dependent. For example, in modeling sensor measurement uncertainty, where each U is an error range containing the mean value, $f(x, t)$ can be a normal distribution around the mean value.

Probabilistic model is the most complex but at the same time most precise sensors uncertainty representation in given classification. Probabilistic models are very commonly used in robotics [47, 48].

4.3 World model optimization

This subsection describes the world model uncertainty state in the context of this thesis and presents some approaches which are able to optimize the model based on obtained sensor information in order to implement the goal task.

4.3.1 World model uncertainty state

In this work the world model state is restricted to knowledge about the position of a graspable body. The position consists of translation and rotation components. In three dimensional space the translational part can be described by its coordinates x, y and z in a three dimensional Cartesian reference frame. The orientation is represented by angle of rotation around the z-axis φ. The study was restricted to 3 DOF. At this stage, the assumption is the existence of uncertainty in x and y coordinates and angle φ. As a result, the vector of uncertain object location, is given in:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ \varphi \end{bmatrix}. \tag{1}$$

As a model for uncertainty, interval uncertainty is chosen. Thus, the real values of world state vector's components lie in some intervals. Sizes of these intervals were determined during the experiments.

4.3.2 Optimization algorithms

One of the objectives of this thesis is to find an approach which allows to update the information about the environment combining the initial predicted values and sensors measurements. These values are usually not equal, there is some difference between them. This difference can be included in the error function. The goal of optimization algorithm is to minimize this error function in order to obtain a new world model state as close to reality as possible. The choice of the optimization algorithm is highly dependent on the type of world model representation. As in this study the world is represented as a linear vector of numbers the optimization should be able to update these values.

In general, optimization algorithms can be divided to ones which try to find the optimum of a single function and ones which optimize sets of objective functions [49, pp. 25–26]. This work is concentrated just on single function optimization problem. In the case of optimizing a single criterion f, an optimum can be either its maximum or minimum, depending on the task. In global optimization, a significant part of optimization problems is defined as minimizations. Thus, the goal of optimization algorithm in relation to this study is to find the global minimum of a error function $f(\mathbf{x})$, where $\mathbf{x}$ is a vector of uncertain object position.

The important point is to distinguish between local and global optima. Figure 13 illustrates function f defined in a two-dimensional space $X = (X1, X2)$ and outlines the difference between these two types of optima.

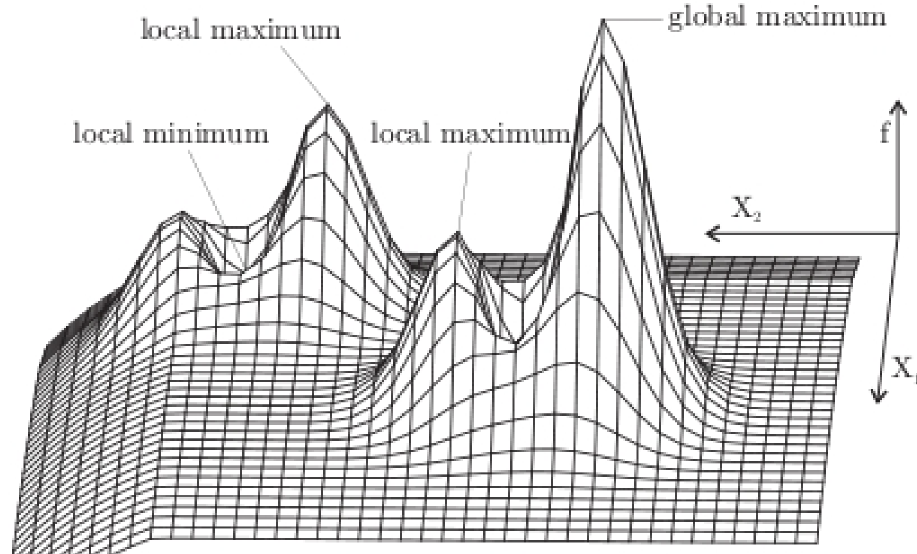


Figure 13. Global and local optima of a two-dimensional function [49].

Local minimum $\breve{x}_l \in X$ of an objective function $f : X \to R$ is an element for which: $f(\breve{x}_l) \geq f(x), \forall x$ neighboring $\breve{x}_l$. If $X \subseteq R$ then:

$$\forall \breve{x}_l \quad \exists \varepsilon > 0 : f(\breve{x}_l) \geq f(x) \quad \forall x \in X, \quad |\breve{x}_l - x| < \varepsilon \tag{2}$$

In its turn, a global minimum $\breve{\mathbf{x}} \in X$ of an objective function $f : X \to R$ is an element for which $f(\breve{\mathbf{x}}) \geq f(x), \forall x \in X$. Equations for local and global maximum can be derived just by changing the signs in the comparison of function values.

The minimum of an error function is a global optimum, therefore a global optimization algorithms should be taken into consideration. Global optimization algorithms apply measures that allow to prevent convergence to local optima and increase the probability of finding a global optimum [49, p. 49]. An unconstrained optimization procedure starts with search for initial point – guess for initial values of unknown parameters $\mathbf{x_0}$ in a function $f(\mathbf{x})$. Good choice of $\mathbf{x_0}$ is able to significantly decrease the computation time. This means that all available information about function behavior should be taken into account in order to prevent the situation where the initial guess is too far from the optimum. This needed to ensure the fact that the optimization algorithm converges to the global optimum. It is said that an algorithm has converged if it cannot reach new solution candidates anymore or if it keeps on producing solution candidates from a limited subset of the problem space [49, p. 58]. Next actions after the start point has chosen are, firstly, to pick the direction along which the next point will be taken and, secondary, to decide the step of what size to make in this direction. Thus, the iterative procedure can be formed:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \lambda_k d_k, \quad k = 1, 2, ..., \tag{3}$$

where d_k is a direction and $|\lambda_k|$ is a step size. Various optimization algorithms suggest different methods to define d_k and $|\lambda_k|$. All optimization approaches can be classified into three big categories [50]:

1. Direct methods that use the optimized function values only.
2. Methods based on first-order derivative calculations.
3. Methods which, in addition, make use of second-order derivatives.

Two last categories are also known as gradient methods. Main criteria in a choice of specific algorithm are computational time and accuracy. Methods of the first type are not widely used in error minimization problem. Methods of the third category, in general case, provide precise enough results in a low number of steps, but their optimality is not guaranteed. The computational costs of second-order derivatives can be considerable compared to computations of several first-order derivatives. Therefore, the second group of methods would be a reasonable choice for a large set of optimization problems, including error minimization.

4.3.3 Method of Steepest Descent

The method of Steepest Descent is the simplest algorithm among gradient–descent methods [50]. The direction where to choose the next value is determined by the direction opposite to the direction of the gradient $\nabla f(\mathbf{x})$, as the gradient points the direction in which the function $f(\mathbf{x})$ maximally increases and the optimum is a function minimum. The procedure starts from an arbitrary point $\mathbf{x_0}$ and follows down the gradient until the point close enough to real solution will be found. Figure 14 shows the schematic trajectory of the Steepest Descent method. As can be seen, the minimum is reached on a

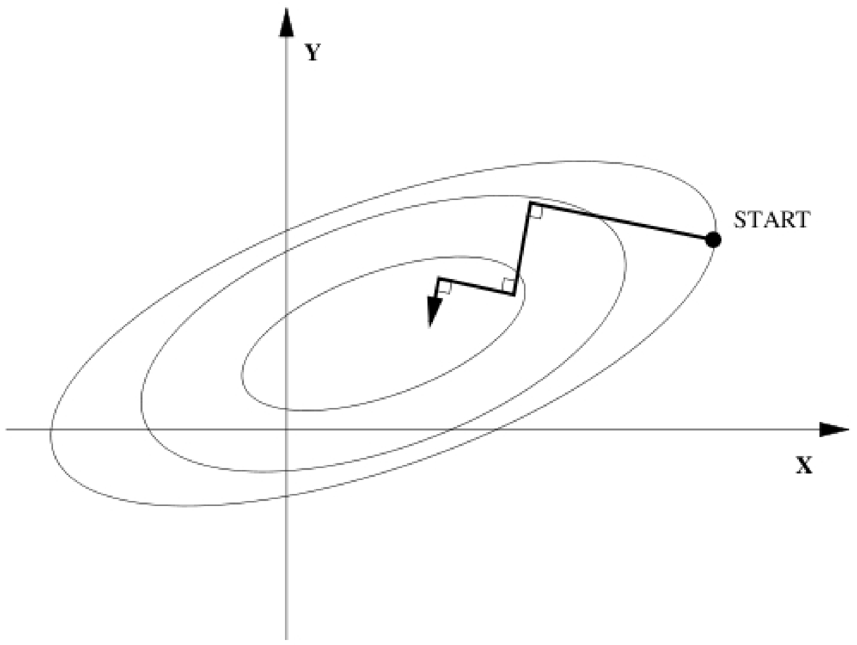


Figure 14. Steepest Descent approach[50].

zig–zag trajectory, where new search direction on each step in case of the best choice of the step size λ_k is orthogonal to the previous. Thus, the directional derivative of the minimizing function f should be equal to zero:

$$\frac{d}{d\lambda_k} f(\mathbf{x_{k+1}}) = \nabla f(\mathbf{x_{k+1}})^T \cdot \frac{d}{d\lambda_k} \mathbf{x_{k+1}} = -\nabla f(\mathbf{x_{k+1}})^T \cdot g(\mathbf{x_k}) = 0, \tag{4}$$

where $g(\mathbf{x_k})$ is the gradient in one given point. Equation 4 acknowledges the type of trajectory, as λ_k should be chosen such that $\nabla f(\mathbf{x_{k+1}})$ and $g(\mathbf{x_k})$ are orthogonal. The next step is taken in the direction of negative gradient at new point. The procedure ends when achieved extrema becomes less than set accuracy ε. The actual problem is a minimization along a line, where the line is determined by Eq. 3 for different values of λ_k. That is why, this problem is often solved by implementing line search. Consequently, the problem of minimization of the function $f(\mathbf{x})$ transforms into sequence of line search. Such implementation of Steepest Descent algorithm is usually referred to as optimal gradient method [50].

Another option is to start with some λ_k and if is needed change its value during the process to guarantee that function $f(\mathbf{x})$ decreases at each iteration. This approach is more simple and more convenient when calculations using line search are laborious. It requires more iterations to reach the minimum, but each of them takes less time compared to one step in line search.

The iterative algorithm for line search can be expressed by following steps:

Initialization: $\mathbf{g_0} = \nabla f(\mathbf{x_0}), \mathbf{d_0} = -\mathbf{g_0}$

1. Determine the step size $\lambda_k : \min_{\lambda_k > 0} f(\mathbf{x_k} + \lambda_k \mathbf{d_k})$.
2. Obtain new point: $\mathbf{x_{k+1}} = \mathbf{x_k} + \lambda_k \mathbf{d_k}$.
3. Calculate the gradient: $g(\mathbf{x_{k+1}}) = \nabla f(\mathbf{x_{k+1}})$.
4. Establish the search direction: $\mathbf{d_{k+1}} = -\mathbf{g_{k+1}}$

Key benefits of this approach are the simplicity of use and speed of each iteration execution. Furthermore, Steepest Descent algorithm is very stable. If minimum points exist, this method is guaranteed to find one at least in an infinite number of steps. Nevertheless, this algorithm has a significant drawback – slow convergence. Thus, it can be concluded that this approach works well when there exists some indication where the minimum is located, but not in general situations. In the case of the task of the study the Steepest Descent approach can be applied for error function minimization, as there is a predefined interval within which the minimum is located.

4.3.4 Method of Conjugate Gradients

In Steepest Descent method it is necessary to take a right angle after each iteration and, moreover, consistently search in the same direction as previous steps. As a result, Steepest Descent method has slow conjugacy. The method of Conjugate Gradients tries to overcome this problem by “learning”from experience [50]. Term “conjugacy”of two vectors $\mathbf{d_i}$ and $\mathbf{d_j}$ means that these vectors are orthogonal to any symmetric positive definite matrix $\mathbf{Q}$. It can be expressed in formula:

$$\mathbf{d_i}^T \cdot \mathbf{Q} \cdot \mathbf{d_j} = 0. \tag{5}$$

The main idea of this approach is to make search direction $\mathbf{d_i}$ dependent on previous search directions to locate the minimum of the function $f(\mathbf{x})$ through Eq. 5. The principal difference to the Steepest Descent method is the choice of next search direction on each iteration. Conjugate Direction ensures that the minimization of $f(\mathbf{x_k})$ along one direction does not damage the minimization along others. Detailed description of Conjugate Gradients methods is given in [51].

Conjugate Gradients algorithm was designed for quadratic function minimization. The n–dimensional quadratic function has the form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \cdot \mathbf{Q} \cdot \mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c, \tag{6}$$

where $\mathbf{x}$ and $\mathbf{b}$ are vectors and c is a scalar constant.

However, the algorithm is applicable also for any continuous function $f(\mathbf{x})$ for which the gradient $\nabla f(\mathbf{x})$ can be calculated. In general form, the algorithm of Conjugate Gradients can be described by the scheme:

Initialization: $\mathbf{g_0} = \nabla f(\mathbf{x_0}), \mathbf{d_0} = -\mathbf{g_0}$

1. Determine the step size $\lambda_k : \min_{\lambda_k > 0} f(\mathbf{x_k} + \lambda_k \mathbf{d_k})$.
2. Obtain new point: $\mathbf{x_{k+1}} = \mathbf{x_k} + \lambda_k \mathbf{d_k}$.
3. Calculate the gradient: $g(\mathbf{x_{k+1}}) = \nabla f(\mathbf{x_{k+1}})$.
4. Establish the search direction: $\mathbf{d_{k+1}} = -\mathbf{g_{k+1}} + \beta_k \mathbf{d_k}$,

$$\beta_k = \frac{\mathbf{g_{k+1}}^T \cdot (\mathbf{g_{k+1}} - \mathbf{g_k})}{\mathbf{g_k}^T \cdot \mathbf{g_k}}. \tag{7}$$

Eq. 7 is known as Polak–Ribière formula [52]. In case of quadratic function minimization a Fletcher–Reeves formula can be applied instead:

$$\beta_k = \frac{\mathbf{g_{k+1}}^T \cdot \mathbf{g_k}}{\mathbf{g_k}^T \cdot \mathbf{g_k}}. \tag{8}$$

The difference between these two formulas is not big, but Polak–Ribière formula in most cases performs better for non-quadratic functions.

It should be noticed that this approach has also some limitations. First of all, one of the practical problems is a loss of conjugacy that results from the non-quadratic terms. The Conjugate Gradients method is often applied in problems with big number of variables, that is why usually method can start producing inefficient directions of search after few iterations. For this reason the process should be implemented in cycles with the first step of Steepest Gradient method. Second requirement is an accuracy of linear search to limit the loss of directional conjugacy. But in practice, linear search can be computation expensive.

Finally, it should be mentioned that Conjugate Gradients method is the most prominent approach to solve sparse systems of linear equations. Generally, it is more preferable to use than Steepest Descent approach. However, if minimized function is not quadratic, the probability for search directions to lose conjugacy is high, and the method may even not converge at all.

4.3.5 The Newton–Rhapson Method

In this section the Newton–Rhapson method is described as an example of methods belonging to second derivative methods. Thus, it takes into consideration the information about the second derivative of function $f(\mathbf{x})$. As a result, faster convergence of a result can be achieved but the method can require more computation time especially for large systems [50].

The approach is based on the idea of approximation of any function $f(\mathbf{x})$ by a quadratic function in form given in Eq. 6 and search for minimum of this quadratic function. The quadratic function at point $\mathbf{x}$ can be approximated by truncated Taylor series:

$$f(\mathbf{x}) = f(\mathbf{x_k}) + (\mathbf{x} - \mathbf{x_k})^T \cdot \mathbf{g_k} + \frac{1}{2}(\mathbf{x} - \mathbf{x_k})^T \cdot \mathbf{H_k} \cdot (\mathbf{x} - \mathbf{x_k}), \qquad (9)$$

where both gradient $\mathbf{g_k}$ and Hessian matrix of second-order derivatives $\mathbf{H_k}$ are evaluated in a point $\mathbf{x_k}$. Then, the derivative can be calculated:

$$\nabla f(\mathbf{x}) = \mathbf{g_k} + \frac{1}{2}\mathbf{H_k} \cdot (\mathbf{x} - \mathbf{x_k}) + \frac{1}{2}\mathbf{H_k}^T \cdot (\mathbf{x} - \mathbf{x_k}). \qquad (10)$$

The Hessian matrix is always symmetric when $\mathbf{x_k}$ is twice continuously differentiable. In this case the Eq. 11 can be simplified:

$$\nabla f(\mathbf{x}) = \mathbf{g_k} + \mathbf{H_k} \cdot (\mathbf{x} - \mathbf{x_k}). \tag{11}$$

If the function $f(\mathbf{x})$ has its minimum in point $\mathbf{x} = \mathbf{x}^*$, the gradient of the function is zero:

$$\nabla f(\mathbf{x}) = \mathbf{g_k} + \mathbf{H_k} \cdot (\mathbf{x} - \mathbf{x}^*) = 0. \tag{12}$$

Equation 12 is a linear system. The Newton–Rhapson method uses $\mathbf{x}^*$ as a next current point in iterative procedure:

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \mathbf{H_k}^{-1} \cdot \mathbf{g_k}, \quad k = 0, 1, ..., \tag{13}$$

where term $-\mathbf{H_k}^{-1} \cdot \mathbf{g_k}$ is known as Newton direction. If (9) is valid, the method converges in few steps. Regardless this fast convergence, the computational costs on each step depend on second derivative and Hessian matrix calculations. The performance of the algorithm relies on qualities of Hessian matrix like its positive definiteness and size. The size of the Hessian matrix can be a crucial factor for the whole method effectiveness. For systems with large number of dimensions the calculation of the matrix and also calculations that uses it are time-consuming processes. Another disadvantage of Newton–Rhapson method is that Newton–Rhapson method is not always globally convergent. The convergence depends on initial guess for the starting point. To ensure that the function value decreases on each step, the adjustment of step size in Newton direction is required:

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \lambda_k \mathbf{H_k}^{-1} \cdot \mathbf{g_k}, \quad k = 0, 1, ..., \tag{14}$$

There exist several schemes for doing this, like line search with constant step, but more preferable is to use backtracking algorithm [52]. The resultant algorithm can be expressed:

1. Calculate the gradient: $g(\mathbf{x_k}) = \nabla f(\mathbf{x_k})$.
2. Calculate the Hessian: $g(\mathbf{x_k}) = \nabla^2 f(\mathbf{x_k})$.
3. Establish the search direction: $\mathbf{d_k} = -\lambda_k \mathbf{H_k}^{-1} \cdot \mathbf{g_k}$.
4. Obtain the length of the step λ_k from backtracking procedure.
5. Update the point: $\mathbf{x_{k+1}} = \mathbf{x_k} + \lambda_k \mathbf{d_k}$

Conversely to Steepest Descent method Newton–Rhapson algorithm starts out slowly but ends with very rapid convergence. Thus, a combination of these two methods (Hybrid method), when starts with Steepest Descent method and when the convergence gets slow switches to Newton–Rhapson method, can be applied in some practical tasks.

4.3.6 Step size determination

As was mentioned in Section 4.3.3, the step size λ_k can change on each iteration to guarantee the the value of function $f(\mathbf{x})$ decreases. Search of optimal size step in it turn can be considered as an one–dimensional optimization problem. Thus, in this context an error function $f(\mathbf{x_{k+1}}) = f(\mathbf{x_k} + \lambda_k \mathbf{d_k})$ is seen as a function of one variable λ_k. For one–dimensional minimization without calculation of derivative, the routine for bracketing the minimum in combination with Brent's method can be applied. If the function has a discontinuous second- or lower order derivative, then simple golden section search can be invoked instead of the parabolic interpolation of Brent's method. In this study for step length determination it was decided to use a simple one–golden section search. This optimization approach is described in [53, pp.398–400].

The golden search procedure requires a preliminary stage which is a bracketing of a minimum. The minimum of a function is known to be bracketed if there exists a triplet of points a, b and c ($a < b < c$ or $c < b < a$), such that $f(b) < f(a) \quad and \quad f(b) < f(c)$. In this case a nonsingular function has a minimum in interval (a, c). For example, if the triplet(a, b, c) is known the function is evaluated in a point x which resides either between a and b or b and c. If $f(b) < f(x)$, then the new triplet is (a, b, x), contrariwise the new bracketing triplet is (b, x, c). Figure 15 demonstrates the successive bracketing of a minimum. The process continues until the difference between the two boundary points of a triplet becomes tolerably small. Determination of a strategy for choosing the new point x, given an initial bracketing triplet (a, b, c) needs several assumptions, that can be expressed in following equations:

$$\frac{b-a}{c-a} = w \quad \frac{c-b}{c-a} = 1 - w \quad \frac{x-b}{c-a} = z \tag{15}$$

Therefore, length of the next bracketing interval is either $w + z$ or $1 - w$. To minimize the worst case possibility the choice can be made to have:

$$z = 1 - 2w \tag{16}$$

The Eq. 16 reveals the fact that the new point is symmetrical to point b in the original interval, so that $|b - a| = |x - c|$. This leads to the conclusion that x lies in larger interval,

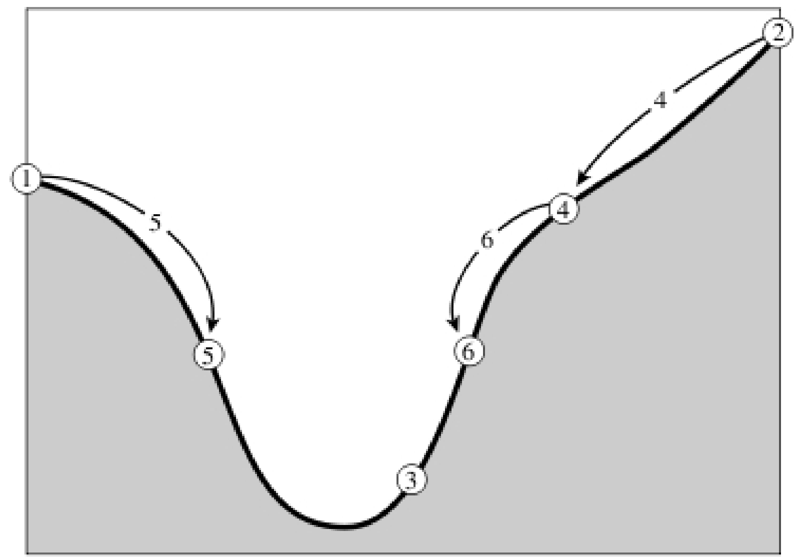


Figure 15. Successive bracketing of a minimum. The initial triplet is $(1, 3, 2)$. After the steps shown, the result is $(5, 3, 6)$ [53, p.398].

because $z > 0$ if only $w < 1/2$. Optimal w value is chosen by applying scale similarity: new point x should be the same fraction of the way from b to c (if (b, c) is bigger segment), as was b from a to c. Thus:

$$\frac{z}{1-w} = w \tag{17}$$

Equations 16 and 17 produce the quadratic equation:

$$w^2 - 3w + 1 = 0 \implies w = \frac{3-\sqrt{5}}{2} \approx 0.38197 \tag{18}$$

As a result, the Eq. 18 indicates that the middle point b of a bracketing interval (a, b, c) lies at fractional distance 0.38197 from one end and 0.61803 from the other. These fractions are called golden section or golden mean. The method of function minimization which make use of these fractions is known as golden section search. It can be described by following stages:

1. Obtain an initial bracketing triplet (the procedure is described later in this section).
2. Take a point which is a fraction 0.38197 from the center point of the triplet into the larger of the two intervals (a, b) and (b, c).
3. Continue until self-replicated ratios will be achieved.

If initial bracketing values are not in the golden ratios, the procedure of choosing new points at the golden mean point of the larger segment guarantees rapid convergence to self-replicated ratios.

An initial bracketing of a minimum is an essential part of any one–dimensional optimization problem [53, pp.400–401]. The main goal of the bracketing is to step downhill by taking larger and larger steps starting from some initial subjective guess. The step size increasing factor can be either the constant or the result of a parabolic extrapolation of the preceding points that is designed to reach the extrapolated turning point. The middle and left points of the triplet are obtained after the step downhill. Then, a big enough step is taken in order to stop downhill trend and get a high third point.

This section covered just one method for one–dimensional minimization. Golden section search procedure was designed to handle the worst case of function minimization. However, there exist some other methods which work with more smooth functions. For instance, if minimized function is sufficiently smooth, that is looks like a parabola near the minimum, then the parabola fitted through any three points leads in a one step to the minimum or to a point that is very close to the minimum. This technique is called inverse parabolic interpolation. Figure 16 illustrates this approach.

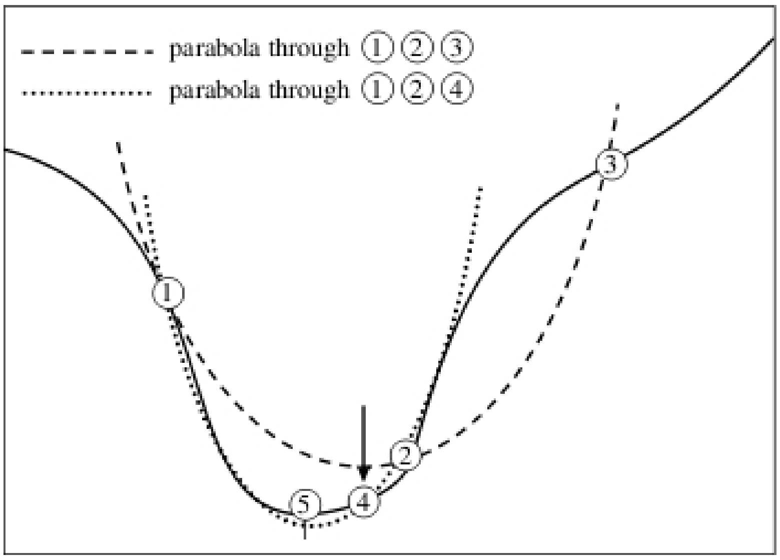


Figure 16. Convergence to a minimum by inverse parabolic interpolation. The initial parabola is drawn through the points $(1, 2, 3)$. The function (solid line) is evaluated in parabola's minimum (point 4) and the new parabola is drawn through $(1, 4, 2)$. Point 5 is the minimum of parabola, which is very close to the real one. [53, p.403].

4.3.7 Examples of problems in optimization

The reason for necessity of variety optimization algorithms is the existence of numerous optimization tasks. The convergence of an algorithm is highly dependent on the function behavior, its shape. In this section illustrations of some problematic function shapes are shown in Figure 17.

One of the problems in global optimization is that is often impossible to determine whether the currently reached solution is a local or a global optimum and thus, if convergence of the optimization algorithm is acceptable. Figure 17 illustrates various situations in which it is not clear whether the optimization process can be stopped. One possible problem, shown on the upper right figure, is a premature convergence to a local optimum. This is the case when the optimization process is no longer able to explore other parts of the search space than the area currently being examined and there exists another region that contains a superior solution [54]. Next problem is a function multi-modality, which means an existence of several (local) optima. If the objective function is unsteady or fluctuating it becomes more complicated for the optimization process to find the right directions to proceed to. Thus, the other type of functions, which are problematic for optimization, are rugged functions [55]. Some functions can show deceptiveness in their shapes. This means, that their gradients lead the optimizer away from the optima. Existence of large flat (neutral) areas leads to the situation when optimizer cannot find any gradient information and thus, no direction in which to proceed in a systematic manner [56]. One of the worst cases of function shapes is the needle-in-a-haystack (NIAH) problem, where the optimum occurs as isolated spike in a plane [49, p.68]. In other words, small instances of extreme ruggedness combine with a general lack of information because of neutral areas. Functions with very sharp peaks compose one more problem for optimization process, which in such situation degenerates to a random walk.

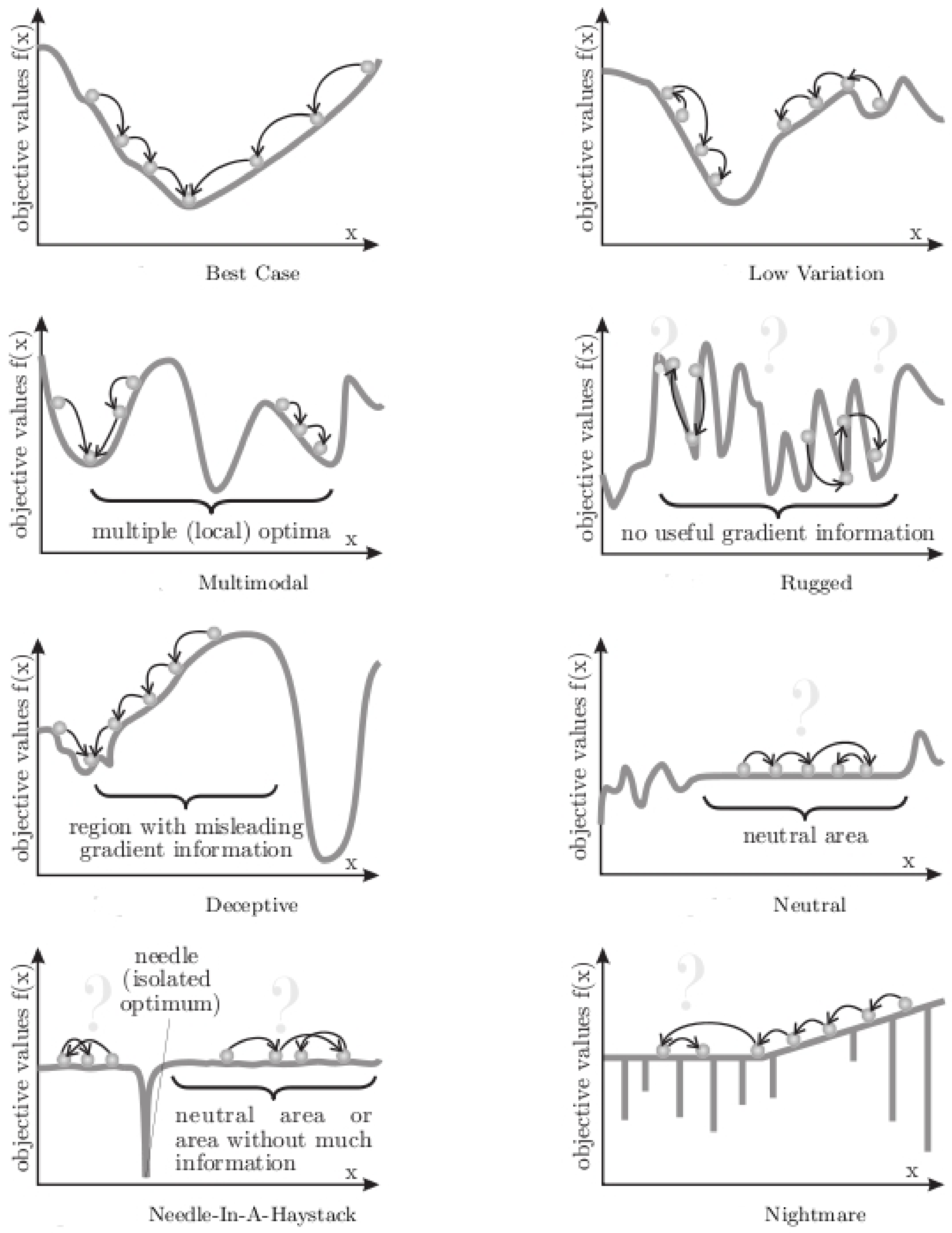

objective values f(x)
x
Best Case
Low Variation
multiple (local) optima
Multimodal
no useful gradient information
Rugged
region with misleading gradient information
Deceptive
neutral area
Neutral
needle (isolated optimum)
neutral area or area without much information
Needle-In-A-Haystack
Nightmare

Figure 17. Different possible properties of function shapes [49, p.57].

5 SYSTEM IMPLEMENTATION

The initial motivation for the present work was to use the simulator as a “memory”for a robot, which allows the robot to try out actions in simulation before the real execution. To ensure that the robot will succeed with its plan the simulation model should be updated. The basis of this update is the error minimization between parameters predicted by the plan and actual sensor readings. For that reason, the optimization method considered in Section 4.3.3 is implemented and tested with different amounts of initial uncertainty. As a result, a simple demonstration of the chosen approach is developed.

At the beginning of this section the experimental task and its environment is considered. After that a general process structure and more detailed description of the method and its MATLAB implementation are provided. The last subsection demonstrates the integration of the proposed method with OpenRAVE simulator.

5.1 Task definition

Transportation of an object was chosen as a practical task for the robot in this study. The robot should grasp a box and move it to another location on a table. OpenRAVE simulator, which is examined in the Section 2.2.2, was chosen as a simulation environment. For the experiment following objects in OpenRAVE environment were selected:

- Barrett WAM was chosen as a robot model. Figure 18 illustrates robot’s DOF’s (totally 10):
  j = joint
  f = finger
  $[j1 \text{ yaw}, j1 \text{ pitch}, j2 \text{ yaw}, j2 \text{ pitch}, j3 \text{ yaw}, j3 \text{ pitch}, j4 \text{ yaw}, f1 \text{ pitch}, f2 \text{ pitch}, f3 \text{ pitch}]$.

  The robot consists of two parts: WAM arm and Barrett hand. WAM arm has 7 DOF’s: shoulder yaw, shoulder pitch, shoulder roll, elbow joint, wrist yaw, wrist pitch and wrist roll. Barrett hand has 4 DOF (spread angle of the fingers and 3 angles of proximal links) in the hand which has 3 fingers and 8 hand axes. Two fingers can spread up to 180° and third finger is stationary. Each finger has one actuated proximal link and 1 coupled distal link. Schematically, Barrett hand is depicted in Figure 19.

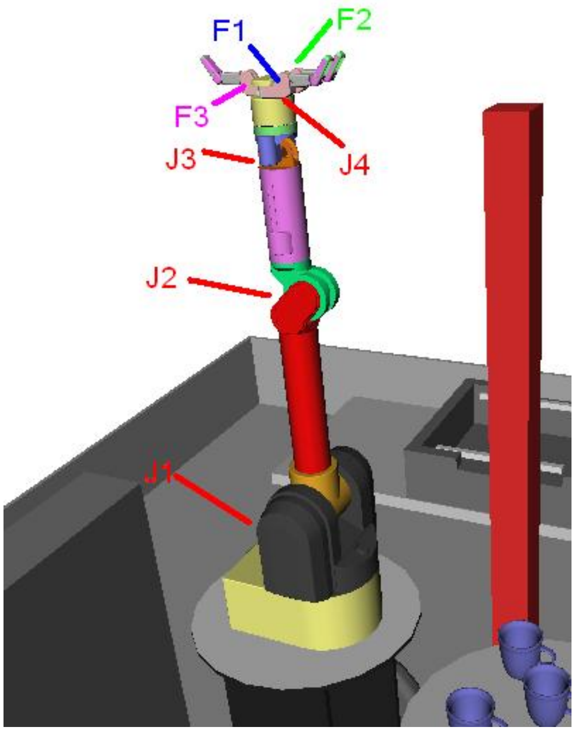


Figure 18. Barrett WAM robot joints.

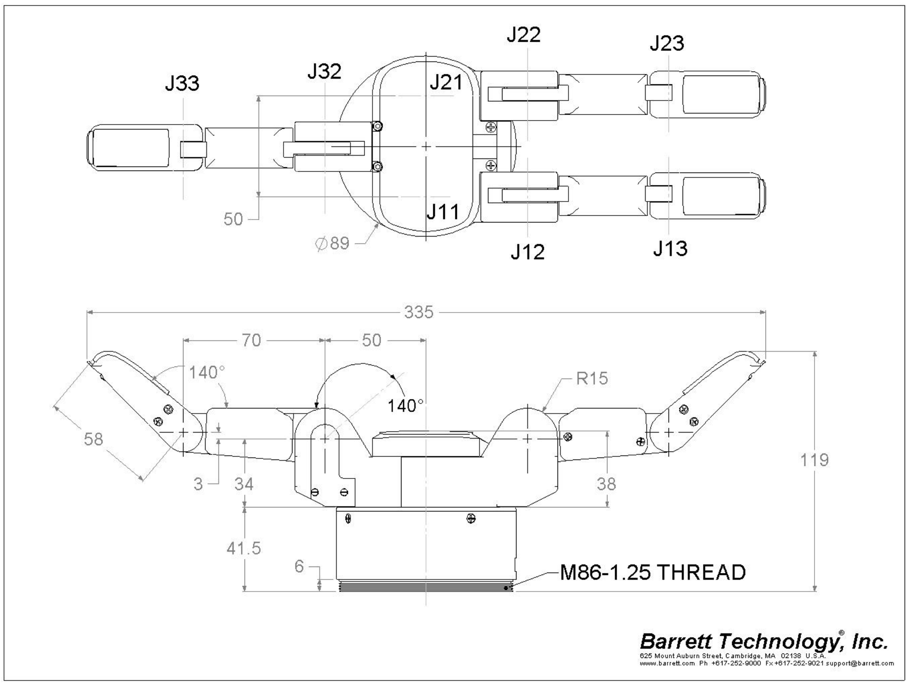
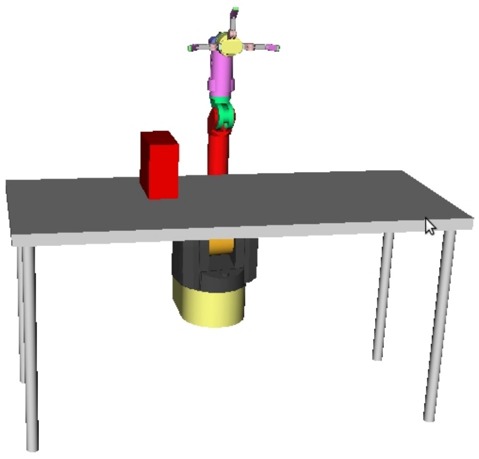


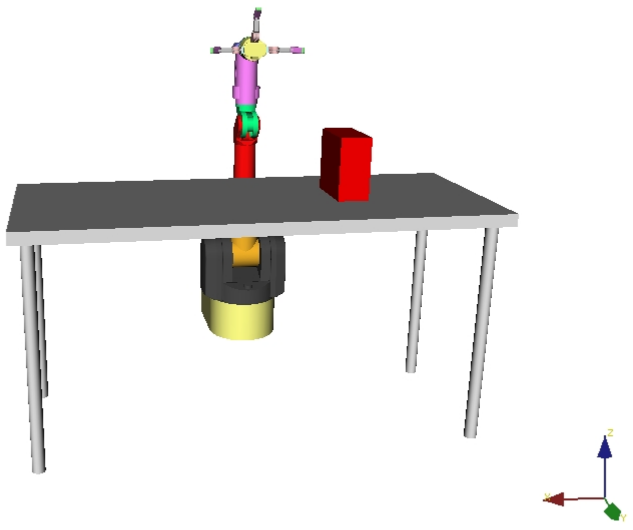
Figure 19. Barrett hand [57].

- Box was chosen as a graspable object. The object has simple rectangular geometric shape with dimensions $0.2m \times 0,2m \times 0.1m$ (heigth $\times$ width $\times$ depth). Thus, it is rather small, easy to move and does not roll.
- Table was selected as a surface on which the object is located and and on which the movement will be fulfilled. Table is 1.83m long, 0.61m wide, 0.035m thick.

The initial configuration of the scene is shown in Figure 20a. As a result, the robot should move the box to a new location, which is 0.5 meters from the initial along negative direction of the x-axis. The final configuration is presented in Figure 20b.



(a)



(b)

Figure 20. Environment configurations: (a) Initial scene; (b) Final scene.

5.2 General process structure

After the practical task is defined the next step is to define stages of its implementation. Figure 21 presents a general process structure for the world model update.

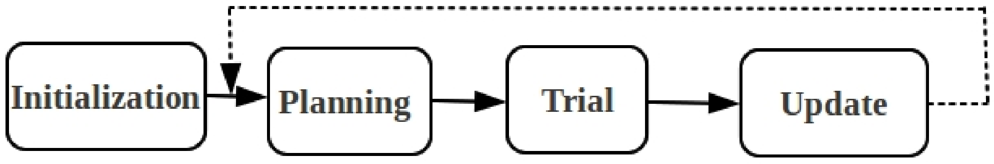


Figure 21. General process structure.

The diagram consists of four basic blocks. At the *Initialization* part the robot makes use of its internal mental view and obtains the initial guess for the object location. With use of this predicted value the robot *plans* a trajectory to complete the task. After that this trajectory is *tried* out and actual sensor readings are obtained. In order to minimize the difference between planned and real values the *Update* block is executed. It includes optimization procedures, which allow to find new value for a simulation model. If one optimization step is not enough the procedure is repeated starting with planning new trajectory for a optimized location.

It is important to point that simulation is used in three of four blocks: *Planning*, *Trial* and *Update*. Use of a simulation for planning or trying some actions is what can be met quite often in robotics. However, application of simulation for the world model update is a new idea, which transforms the simulation process in a robot internal mental view.

5.3 MATLAB implementation

The optimization algorithm is completely implemented in MATLAB. Several functions have been developed to solve each part of the error minimization problem. The results of the main optimization method execution are an array of optimized object location coordinates and corresponding error values. Sometimes, during the simulation work the robot is not capable to find non colliding trajectory. Another problem is an absence of inverse kinematics solutions for the range of predicted values. To handle these problematic situations an optimization process can be repeated several times for a given initial location. If it is still impossible to find a solution, the function returns zero value as a result.

Figure 22 schematically presents the hierarchy of developed MATLAB functions: More

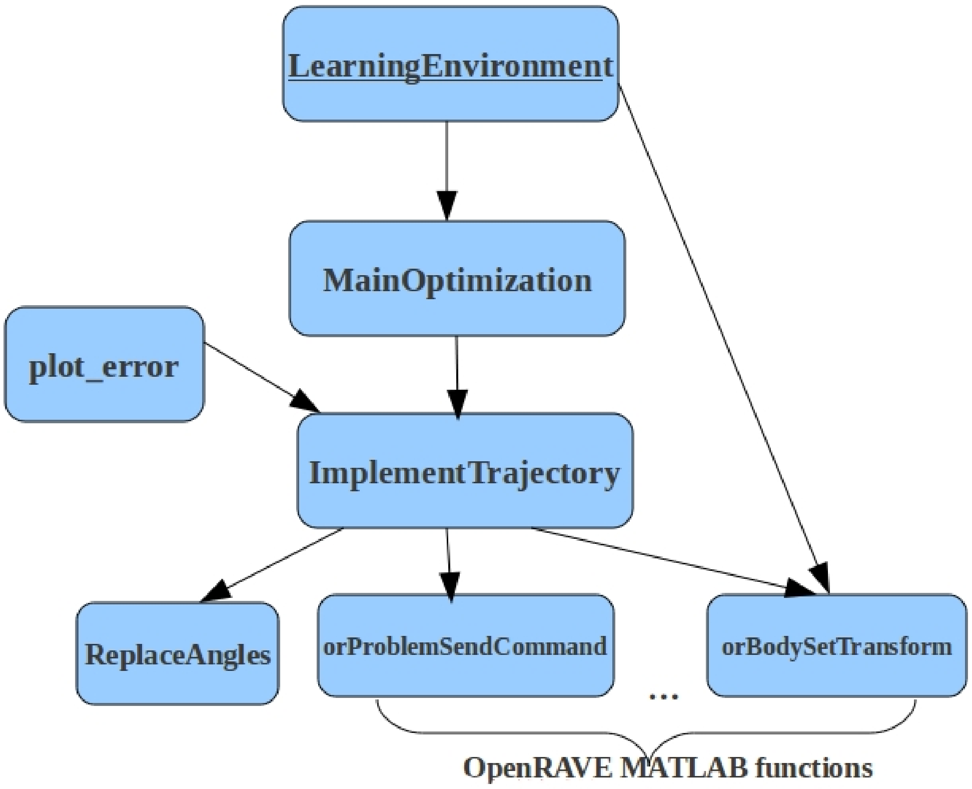


Figure 22. Functions hierarchy.

detailed description of the functions is now provided. Main function *LearningEnvironment* contains all scene initialization information. It implements the environment creation. Moreover, planned box position and matrix of different variant of real positions for testing are also specified in the main function. Thus, *Initialization* process depicted in Figure 21 is completely implemented by this function. *LearningEnvironment* in its turn calls function *MainOptimization* which is responsible for the optimization process, included in the *Update* block in Figure 21. It includes several subfuctions, which execute different subtasks, like gradient and error value calculations, bracketing minimum and golden section search. The theoretical background for these procedures is given in Sections 4.3.3 and 4.3.6 of this thesis. The function *MainOptimization* uses function *ImplementTrajectory*, which binds MATLAB interface with OpenRAVE simulator. All actions, related to the *Planning* and *Trial* steps shown in Figure 21, are executed within this function. *ImplementTrajectory* sends commands, including the information about object locations and manipulation goals to the simulator through OpenRave MATLAB functions and simulator executes these commands. The result can be seen in the visualization window. For coupling with the difference in hand joint angles when applying the planned trajectory for grasping the box in real location the function *ReplaceAngles* were designed. It replaces the planned angles value by actual ones.

Specifically for the evaluation of proposed error function, function for plotting the error values for a grid of different locations was created. It calls the function *ImplementTrajectory* for communication with the simulator and produced a 3D plot of calculated error values.

Finally, it is worth mentioning that MATLAB environment is a good choice for implementation, as it can provide very accurate numerical approximation to solve the optimization problem. The result parameters allowed to evaluate the effectiveness of the algorithm.

5.4 Integration with OpenRAVE

One of the biggest features that distinguishes OpenRAVE from other simulation/planning environments is that it supports scripting over the network. This makes it possible to free OpenRAVE of complex GUIs. At initialization, OpenRAVE starts listening on specific ports for the commands. This allows any computer to open a socket connection with OpenRAVE, which can be running on a different computer, and communicate with it. The official release supports MATLAB and Octave as scripting engines. All script commands are sent as text through the network [58].

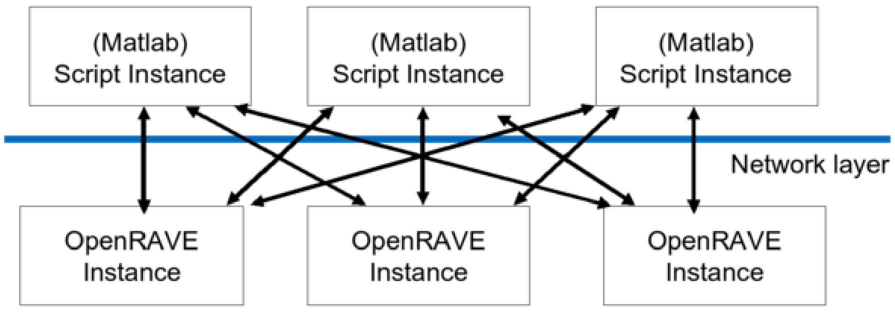


Figure 23. Communication of several OpenRAVE/scripting instances [58].

Communication with an OpenRAVE instance running on the same computer as the scripting environment is executed by calling the methods in the MATLAB folder directly without the necessity to set an IP address. The scripting environment can communicate with multiple OpenRAVE instances at once, orConnectionParams need to be set to the appropriate instance. Also, OpenRAVE can handle multiple scripting environments talking to the same instance simultaneously. Figure 23 illustrates the interaction process.

Thus, the process of MATLAB integration with OpenRAVE simulator is relatively simple. Functions, implemented in MATLAB (*LearningEnvironment*, *ImplementTrajectory*, *ReplaceAngles*) call OpenRAVE functions with specific format, which access robot and object parameters.

6 EXPERIMENTS AND DISCUSSION

The method presented in this thesis can be considered successful if it converges to the global optimum, which means that the optimized world state is close to the real one. Different aspects of the method performance are now assessed by a set of experiments. The intention of these experiments is to validate the conditions that are needed to obtain acceptable results. Moreover, the limitations of the method will be clarified.

In the beginning of this section experiment design is described. Second, the experimental results and their analysis are presented. Finally, open questions are stated and several possibilities to extend and improve the method in a future work part are proposed.

6.1 Design of experiments

Several particular and more general questions were posed within this study. All experiments were designed in order to solve these questions. These general and more particular questions were:

- Particular questions:
  - The sensitivity of the method to different parameters:
    - Calculation of a gradient.
    - Number of DOF.
- General questions:
  - How many unknowns can be evaluated using the implemented method? How far can it be extended? What are the limitations?
  - How big errors in initial estimates are allowed? Where this method can/can't be used?
  - What is a proper choice of the error function?

First of all, the sensitivity of an approach to its component values, such as gradient and error function should be evaluated. Calculation of a gradient is an essential part of the whole method and its speed should be analyzed. For this reason, one-sided and two-sided gradients were used in error function calculation. In this work a simplification was made to approximate a gradient by a difference between error function values on different steps divided by the step size. Thus, one–sided gradient can be expressed in equation

$$grad_{one} = \frac{E(x+dx) - E(x)}{dx}, \tag{19}$$

where $E(x)$ is a value of an error function in current point, $E(x+dx)$ is an error function value on the next optimization step and dx is the step size.

The difference of two–sided gradient is that instead of current error value it uses the value in a point obtained by subtracting the step size from the current location. Then, the difference is divided by the step value multiplied by two.

$$grad_{two} = \frac{E(x+dx) - E(x-dx)}{2dx}, \tag{20}$$

Another important question is how big the errors can be in initial expectations, so that the method is capable to couple with this uncertainty in the data. Thus, the number of unknowns, which can be evaluated by implemented method is also a key parameter. Initially, the uncertainty was added in one state vector parameter – coordinate along x–axis. Such experiment is needed to prove the applicability of proposed algorithm for solving given optimization problem. Greater interest is in a possibility to increase the uncertainty and test the method at least for 2DOF case, which implies that y–coordinate is also exactly unknown. Application areas of the method are highly dependent on the number of dimensions in which this approach can succeed.

The choice of an error function, in its turn, can affect dramatically the method performance. As was shown in Section 4.3.7, the geometry of a function is a characteristic, which influences directly on the algorithm convergence to a global minimum. That is why another group of experiments was concentrated on the examination of behavior of different error functions with respect to the method convergence. The first option for an error function is just a sum of squared differences of collision matrices. Collision matrix is a matrix consisting of ones and zeros. Each value corresponds to existence or absence of a contact between one of the robot hand links and a graspable object at a defined time instant. Mathematically, the error function can be described by

$$E_{in} = \sum_{t} \sum_{i} (coll_p^{(t,i)} - coll_r^{(t,i)})^2, \tag{21}$$

where $coll$ is a matrix of contacts between hand links and an object at defined time instants. The error function can be modified by multiplying it by a weight function. One choice is to use exponential function:

$$w = e^{\frac{-\left(t_{cur}-t_{first}\right)}{a}}, \tag{22}$$

where the t_{cur} is a time instant when the first difference between planning and real contacts occurred, t_{first} is a current time instant and a is a damping coefficient that affects the rate of exponential decrease. So, the first occurred error has the weight one, and the smaller the value a the smaller the weight for later errors. The modified error function is expressed by:

$$E_w = \sum_t w \sum_i (coll_p^{(t,i)} - coll_r^{(t,i)})^2. \tag{23}$$

Moreover, an error function can take into account not only the information related to the contact occurrence. It can be complemented by adding an information about finger joint angles at the grabbing time instant. For this reason, the sum of squared differences in finger angles at the moment of complete close around the graspable object can be added in the error function equation:

$$E_{mod} = E_w + \sum_j (ang_p^j - ang_r^j)^2. \tag{24}$$

After error function comparison is made, an analysis of regions of convergence can be performed and limits of the method applicability can be specified.

6.2 Analysis of results

In general, the process of solving the optimization problem was implemented in stages, gradually complicating the task. Figure 24 illustrates an overall workflow:

After a range of major issues was defined, a sequence of experiments was conducted and results, which allowed to solve or clarify these problematical aspects were obtained. First type of experiments was focused on the analysis of several variants of an error function. The second one was aimed at solving the problem of determining areas of method convergence for an error function, which choice was based on results obtained from the first experiment.

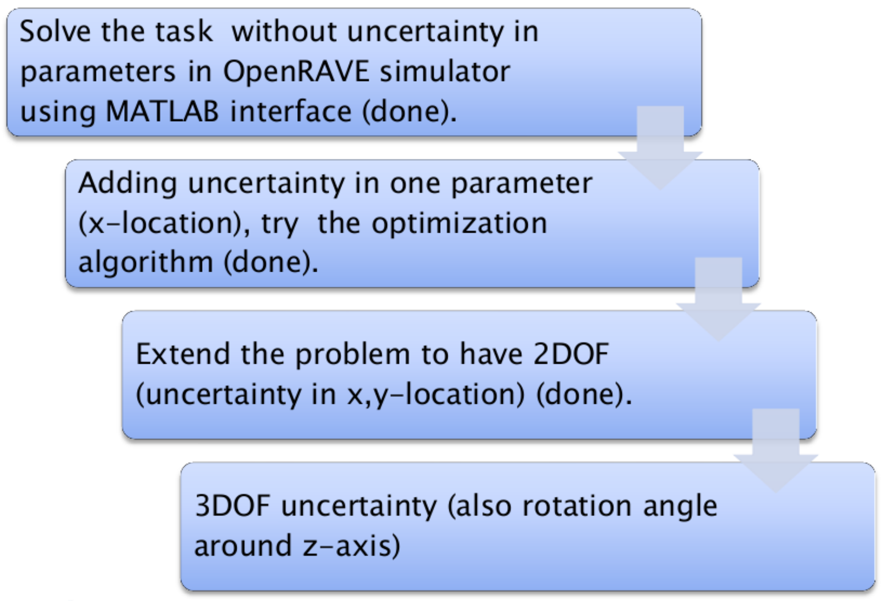

Solve the task without uncertainty in parameters in OpenRAVE simulator using MATLAB interface (done).
Adding uncertainty in one parameter (x-location), try the optimization algorithm (done).
Extend the problem to have 2DOF (uncertainty in x,y-location) (done).
3DOF uncertainty (also rotation angle around z-axis)

Figure 24. Main stages of the work.

6.2.1 Group of experiments 1. Error function shape

Primarily, the difference between real and predicted measurements was determined by the weighted sum of squared differences of collision matrices, which is described by (22). To establish if some modifications for a function are required it was decided to look at the shape of the function. For this purpose an error was calculated for a grid of points around a chosen real object location, so that all these points were considered as robot initial guesses. All calculations were made for 2DOF case.

The parameters for the error calculation experiment are given in Table 3.

Table 3. Parameters for errors calculation.

Parameter	Value
real object location	(0.2, -0.1)
interval along x	[0.1, 0.3]
step size dx	0.01
interval along y	[-0.15, -0.04]
step size dy	0.005

All units are in meters. The considered interval along x–axis is larger than along y, but the step size dx is also bigger than dy. Such values were chosen due to the fact that collision detection along y–axis is more sensitive than along x and changes in contacts along x can be registered only after taking bigger step.

The resulting surface plot of error values is depicted in Figure 25a. The function shape reveals several difficulties in applying it in this study. These problems can be described by situations presented in Section 4.3.7 and shown in Figure 17. First of all, there is a lot of flat or so–called *neutral areas* in the function, which mean that no gradient information can be found and thus, the direction can not be systematically determined. Search operations in all directions are equally meaningful. The error function has quite large intervals in which values are the same. This situation is explained by the fact, that collisions for neighboring locations occur in same links and at same time instants. Even weight coefficients can not cope with this problem.

Neutral areas cause another significant challenge, which is known as function *multi–modality*. Error function is not equal to zero only in one point. There exists a whole range of locations with zero values, which means that the task does not have a unique solution.

Deceptiveness is another shortcoming of the error function. The gradient of deceptive function leads the optimizer away from the optima, which results in the fact that a search algorithm performs worse than a random walk or an exhaustive enumeration method [49, p.64].

One more observing problem is a *premature convergence* to a local minimum. This is explained by existence of multiple local minima. If a new location on some iteration is found in a subarea around such a local optimum, the method will not be more able to explore other parts of the search space except the area where it is now. As a result, the global optimum can not be reached at all.

All of the listed problems lead to the conclusion that the proposed error function can be effectively optimized only in a small area around the real value and more than that it is impossible to be sure of results accuracy because of several global optima. As a result, to solve the ambiguity problem and improve results, it was decided to use a more smooth error function. Thus, in the next experiment the modified error function, which includes additional information about finger angles, was used. A mathematical representation of this function is given in (24).

An experiment with the same parameters as for the first trial (shown in Table 3) was repeated and, consequently, a new error function surface plot was obtained, which is demonstrated in Figure 25b. Several improvements were achieved by making modifications in the error function comparing to the previous results. First of all, new function has less flat areas. Variations are still low, but there are not big areas with same error values at least in the subregion around the global optimum. Moreover, the modified function is not more globally multi–modal, which means that there is only one global minimum. On the other hand, the function still has several local optima and at some point the function starts to show deceptivity in its behavior. Thus, it can be concluded that the modified function is better than initial one regarding of its usage in optimization process, but further improvements are still needed to eliminate remaining problems.

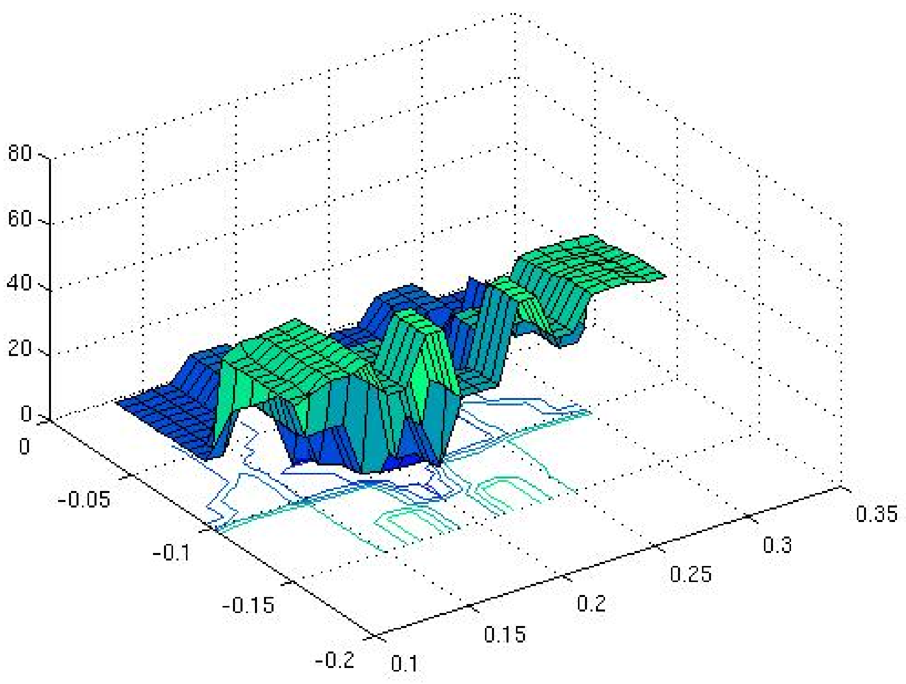
Conducted experiments allowed to compare two types of error functions. Their benefits and drawbacks were analyzed. It was decided at this stage to choose the modified function version and use it in subsequent experiments.

6.2.2 Group of experiments 2. Regions of convergence

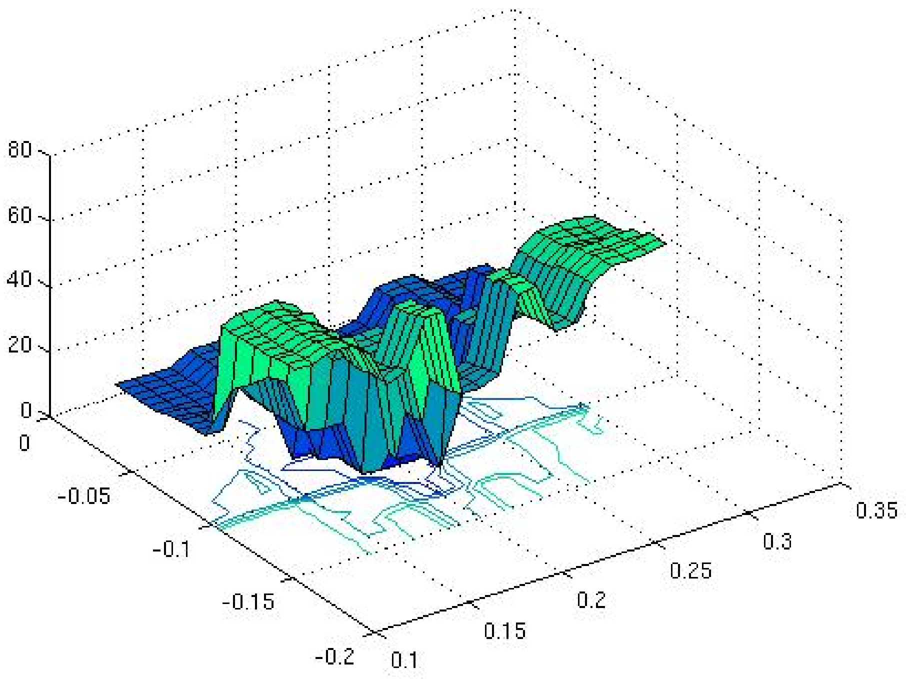
After the suitable error function was selected, the analysis of regions of convergence for the implemented optimization approach was performed. This experiment was carried out in order to determine the limitations of the method, maximal errors in initial predictions for object locations which can be minimized.

The set of parameters was the same as in preceding experiments (see Table 3). Numerical experimental results are provided in Figure A1.1 in Appendix 1. The threshold value for the error value as a stop condition for an optimization process was set to one. Such assumptions were done to be able to match found regions of convergence with error function shape. Figure 26 summarizes in a graphical representation the results of method performance in 2DOF case. White areas show areas in which algorithm converged, which means that the final error was less than the defined threshold. In gray areas the method did not converge (error values were greater than threshold), but the direction of search was found correctly, which means that with more iterations there would be a possibility to converge. Black areas correspond to failure regions. Even directions of a search were incorrect in these points.

As can be seen from the plot, the method is quite stable in intervals $[0.14, 0.26]$ along x–axis for all considered y values. The real location of an object was in $(0.2, -0.1)$. Thus, possible errors in x are about ± 0.05m. Along y–axis the method perform even more precisely: admissible errors are also about ± 0.05m, but it could be even larger if considered larger interval along y. More accurate optimization along y–axis can be



(a)



(b)

Figure 25. Error function shapes: (a) Initial error function; (b) Modified error function.

explained by the behavior of an error function, which is more sensible to changes in y–direction, so that gradient values along y–axis vary more than along x.

Thus, experimental results proved the workability of the proposed gradient–based method and, in addition, reaffirmed the applicability of designed error function in this method. However, the accuracy of the method is not very high. Values obtained after optimization sometimes are not close to the real location, as errors in these values reach $\pm 0.01m$. Usually, the location is determined precisely only along one direction. More than that, the method fails in large areas, so that it can be applied just for the limited regions around the real position. Thus, to apply it a good initial guess as a start point is required. One more disadvantage of the gradient method with a linear search algorithm for step size determination is its slowness. Simulation should be implemented for each new location at all intermediate steps and these operations take a considerable amount of time. The optimization process for one initial guess lasts 15–25 minutes if no errors occur. In conclusion, it can be noted that all revealed problems are associated both with the error function and the gradient method.

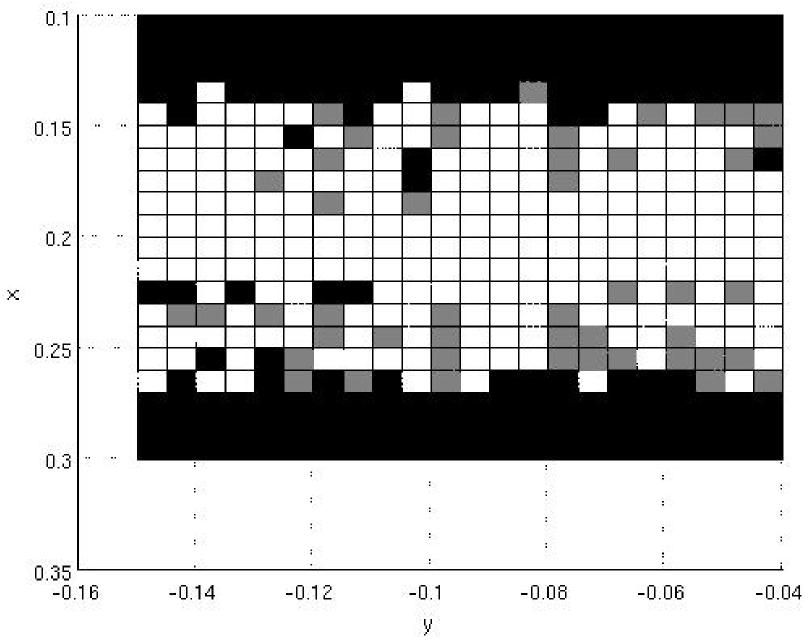


Figure 26. Resulting regions of convergence for 2DOF case.

6.3 Future work

Drawbacks of the method revealed in experimental results form the basis of future work. Several directions of extension are proposed in this subsection. One possible way is to continue the modification of the chosen gradient approach. Another way is to select another optimization algorithm. Both directions aim to solve in the most efficient form the problem of the difference minimization between predicted values and actual readings because of uncertainty and updating the world model.

Considering the technique proposed in this work, the next stage is to extend the method, primarily, to 3DOF case and further in higher dimensional spaces. This will allow to answer the question about the possible number of unknowns with which the method can effectively deal with. Another possibility is to improve the error function in such way that it will contain less problematic features.

More global changes can be also done. First of all, the changes concern an optimization method. In this study the simplest gradient–based approach was applied but instead of it more intelligent gradient methods, like the method of Conjugate Gradients described in Section 4.3.4 can be implemented. Furthermore, not only gradient methods are convenient for such problem solving. Another possibility is to use, for example, the Simulated Annealing method [59]. The basic ideas of simulated annealing are applicable to optimization problems with continuous N–dimensional control spaces, for instance, finding the global minimum of some function $f(\mathbf{x})$, in the presence of many local minima. It is exactly the same problem that is examined in this study. The advantage of this algorithm is also in its implementation simplicity in MATLAB environment, as a ready toolbox for it already exists. Besides Simulated Annealing method genetic or evolutionary algorithms can be also selected.

An opposite evolutionary direction is not the method, but task changing. The robot model or a graspable object can be replaced. Also, new objects and obstacles can be included in the environment. The world state can be extended to contain not only information about one object but also about its surroundings and relations between them. Information sources, which provide knowledge about the environment, can be also completed by other sensor types, like vision or range sensors. This will add accuracy to obtained data and therefore, will make the whole model more robust.

Also, the final goal of this learning process is not to use it only in simulation, but test it on a real robot. The simulation in this case would be its internal mental view, which executes all optimization procedures based on knowledge about the real world state from sensor readings. As an output it provides an optimized world model which will allow to succeed with the given task in uncertain conditions.

7 CONCLUSION

The motivation for this work was develop a technique, which would allow to make the simulation a kind of robot “memory”, its internal mental view of the environment. This is a new idea that has received virtually no attention especially in the robot manipulation research area. The method proposed in this thesis aims to provide the way to try out actions in simulation before executing them for real by solving the error minimization problem.

During the study the first objective, determination of world state components, which compose the input in a simulator, was accomplished. The world state vector was restricted by the object location in 3D Cartesian coordinate system and its orientation with respect to z–axis. The selection of perception information sources, contact sensors, determined the choice of parameters.

The second objective was defined as the analysis of types of environment models for the robot that can be used in simulation, and the choice of the most suitable model. Different world models applied in intelligent systems and simulation software were reviewed. Particular attention was paid to the ability of these models to take into account the environmental uncertainty. As a result, the simulator OpenRAVE with its kinematic body and robot interfaces was selected for practical demonstration purposes. This simulator provides powerful manipulation and grasping functionality, a large library of robot models and allows simple interaction with MATLAB scripting interface through the network.

The world perception aspect, as the main source of knowledge about the actual world state, was taken into account. Various sensor types, used for solving manipulation tasks, were discussed. Tactile sensors, which detect contacts with objects, were selected to be used in application. These sensors are capable of providing highly reliable information about unknown objects compared to vision–based sensors, which usually incorporate some residual uncertainty.

The main problem to solve was to update world model minimizing the difference between predicted values and real sensor readings. A set of optimization algorithms was studied and the Steepest Descent method supplemented by Golden Section search procedure to find the step size was proposed for realization. This is a gradient–based approach, which is simple to use and stable. Moreover, each iteration is fast. However, the significant drawback of this approach is its slow convergence.

The proposed method implemented in MATLAB was integrated with a simulator. A Barrett WAM robot model and a simple rectangular shape box were selected to act in solving the transportation task. Generally, the overall scheme of task execution included four steps: initialization, planning, trial and update. The role of simulation on the update stage is the main contribution of this work. Simulation was used previously for planning and trials, but it has hardly been used for intellectual updating of the world model. All optimization procedures were executed in simulation in order to output new environment state as close to reality as possible.

A set of experiments was proposed to analyze the optimization method. The objective of this analysis was to evaluate the method regarding the most important aspects of its performance, such as its sensitivity to parameters of the approach, like gradient calculation and a type of an error function, and also to the amount of initial uncertainty. The extensibility of the method was also assessed. Two groups of experiments were conducted in order to reach these goals.

Regarding the results obtained from the experiments, it was demonstrated that the proposed method accomplished its main intention with several limitations, which were also defined. The method performs satisfactorily if there is a good initial guess for an object location. More than that, it is very sensitive to the behavior of error function. Separate examination of error function's shape was made and problematic features were revealed. An initial error function was not applicable for a method and it was modified. The algorithm was tested and it was successful for 1 and 2DOF cases. Regions of convergence and areas, where the method failed have been determined.

Thus, breakdown points of the method clarified the directions of future work. First of all, the algorithm should be tested for higher dimensional cases to determine boundaries of its extensibility. Furthermore, the error function should be modified to lose deceptiveness and reduce the number of neutral areas. More generally, other non–gradient optimization approaches can be more effective than the Steepest Descent method. Moreover, new objects and obstacles in the scene can be added, as they will increase the number of unknowns in the world state. The most important improvement is to try out the method on a real robot to evaluate the viability of the proposed idea.

REFERENCES

[1] Leon lajpah. Simulation in robotics. *Math. Comput. Simul.*, 79:879–897, December 2008.

[2] Robotic simulation. white paper. KUKA Robotics Corporation, 2006.

[3] O. Khatib, O. Brock, K. C. Chang, D. Ruspini, L. Sentis, and S. Viji. Human-centered robotics and interactive haptic simulation. *International Journal of Robotics Research*, 23, 1992.

[4] Mike Stilman and James J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Journal of Humanoid Robotics*, pages 322–341, 2004.

[5] Eve Coste-Maniere, Éve Coste-maniére, Louai Adhami, Fabien Mourgues, Olivier Bantiche, David Le, David Hunt, Nick Swarup, Ken Salisbury, and Gary Guthart. Optimal planning of robotically assisted heart surgery: Transfer precision in the operating room. *J. of Robotics Research*, 23:4–5, 2003.

[6] A. Cavalcanti and R.A. Freitas Jr. Nanorobotics control design: A collective behavior approach for medicine. IEEE Trans. NanoBioScience, June 2005.

[7] E. Papadopoulos S. Dubowsky. The kinematics, dynamics, and control of free-flying and free-floating space robotic systems. IEEE Trans. Robotics and Automation, Oct 1993.

[8] K. Hsiao, L. Kaelbling, and T. Lozano-Perez. Task-driven tactile exploration. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[9] Yasemin Bekiroglu, Janne Laaksonen, Jimmy Alison Jorgensen, Ville Kyrki, and Danica Kragic. Learning grasp stability based on haptic data. June–July 2010.

[10] R. Platt. Learning grasp strategies composed of contact relative motions. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 49 –56, December 2007.

[11] V.N. Christopoulos and P. Schrater. Handling shape and contact location uncertainty in grasping two-dimensional planar objects. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1557 –1563, November 2007.

[12] Pablo Inigo Blasco. Simulation in robotics. NanoJornadas Imaginatica, 2010.

[13] Leon lajpah. Planar manipulators toolbox: User's guide. Technical Report DP-7791, Jozef Stefan Institute, 1997.

[14] Symblic Dynamics Inc. *SD/FAST User's Manual*, 1994.

[15] P.I. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, March 1996.

[16] The Mathworks. *SimMechanics, User's Manual*, 2005.

[17] J.F. Nethery and M.W. Spong. Robotica: a mathematica package for robot analysis. *IEEE Robotics & Automation Magazine*, 1:13–20, 1994.

[18] Andrew T. Miller and Andrew T. Miller. Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, 11:110–122, 2004.

[19] Andrew Tesch Miller. *Graspit!: a versatile simulator for robotic grasping*. PhD thesis, New York, NY, USA, 2001. AAI3028562.

[20] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA, July 2008.

[21] Beatriz León, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski, Antonio Morales, Tamim Asfour, Sami Moisio, Jeannette Bohg, James Kuffner, and Rüdiger Dillmann. Opengrasp: a toolkit for robot grasping simulation. In *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots*, SIMPAR'10, pages 109–120, Berlin, Heidelberg, 2010. Springer-Verlag.

[22] Collada. `http://collada.org`.

[23] Pal (physics abstraction layer). `http://www.adrianboeing.com/pal`.

[24] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. Technical report, 1999.

[25] E. Angelopoulou, T.H. Hong, and A.Y. Wu. World model representations for mobile robots. In *Intelligent Vehicles '92 Symposium., Proceedings of the*, pages 293 –297, jun-1 jul 1992.

[26] Wolfram Burgard and Martial Hebert. World modeling. In Siciliano and Khatib [60], pages 853–869.

[27] P.K. Allen. Mapping haptic exploratory procedures to multiple shape representations. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1679 –1684 vol.3, may 1990.

[28] Brian Erick O'Neil. Graph-based world-model for robotic manipulation. Master's thesis, The University of Texas at Austin, August 2010.

[29] Benjamin Kuipers, Rob Browning, Bill Gribble, Mike Hewett, and Emilio Remolina. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000.

[30] F. Merat and Hsianglung Wu. Generation of object descriptions from range data using feature extraction by demands. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 941 – 946, mar 1987.

[31] L. De Floriani. A graph-based approach to object feature recognition. Proc. 3rd ACM Symposium on Computational Geometry, Waterloo, Ontario, June 1987, 1987.

[32] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[33] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.

[34] R. J. Valkenburg and A. M. McIvor. Accurate 3d measurement using a structured light system. *Image and Vision Computing*, 16(2):99 – 110, 1998.

[35] Kinect xbox. `http://www.xbox.com/en-US/kinect`.

[36] Robert B. Fisher and Kurt Konolige. Range sensors. In Siciliano and Khatib [60], pages 521–542.

[37] Antonio Morales, Mario Prats, Pedro Sanz, and Angel P. Pobil. An experiment in the use of manipulation primitives and tactile perception for reactive grasping.

[38] R. D. Howe. Tactile sensing and control of robotic manipulation. *ADVANCED ROBOTICS*, 8(3):245–261, 1994.

[39] W. D. Hillis. Active touch sensing. *International journal of Robotics Research*, (1):33–44, 1982.

[40] J. Tegin and J. Wikander. Tactile sensing in intelligent robotic manipulation - a review. *Industrial Robot*, 32(1), Jan 2005.

[41] Mark R. Cutkosky and Robert D. Howe. *Human grasp choice and robotic grasp analysis*, pages 5–31. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[42] K.Y. Goldberg and M.T. Mason. Bayesian grasping. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1264 –1269 vol.2, May 1990.

[43] Peter Allen and Ruzena Bajcsy. Robotic object recognition using vision and touch. Technical report, Proceedings of the 9th International Joint Conference on Artificial Intelligence, 1987.

[44] Sharon Ann Stansfield. *Visually-guided haptic object recognition*. PhD thesis, Philadelphia, PA, USA, 1987. AAI8804964.

[45] Reynold Cheng and Sunil Prabhakar. Managing uncertainty in sensor database. *SIGMOD Rec.*, 32:41–46, December 2003.

[46] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 551–562, New York, NY, USA, 2003. ACM.

[47] Sven Koenig and Reid G. Simmons. Unsupervised learning of probabilistic models for robot navigation. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2301–2308, 1996.

[48] Julien Diard, Pierre Bessiere, and Emmanuel Mazer. Combining probabilistic models of space for mobile robots: the Bayesian Map and the Superposition operator. In *Proc. of the Int. Advanced Robotics Programme*, pages 65–72, Madrid (ES) France, 10 2003. voir basilic : http://emotion.inrialpes.fr/bibemotion/2003/DBM03a/ note: Int. Workshop on Service, Assistive and Personal Robots. Technical Challenges and Real World Application Perspectives address: Madrid (ES) editor: Armada, M. and Gonzalez de Santos, Pa.

[49] Thomas Weise. *Global Optimization Algorithms - Theory and Application*. Self-Published, second edition, June 26, 2009. Online available at http://www.it-weise.de/.

[50] Trond Hjorteland. *The action variational principle in cosmology*. PhD thesis, Institute of Theoretical Astrophysics University of Oslo, June 1999.

[51] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[52] W.H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *NUMERICAL RECIPES in Fortran 77*. Cambridge University Press, second edition, 1992.

[53] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.

[54] Phd Dissertation, Rasmus K. Ursem, Supervisors Thiemo Krink, Brian H. Mayoh, and Figures Xfig. Models for evolutionary algorithms and their applications in system identification and control optimization, 2003.

[55] Aviv Bergman and Marcus W. Feldman. Recombination dynamics and the fitness landscape. *Phys. D*, 56:57–67, April 1992.

[56] Philippe Collard William Beaudoin, Sebastien Verel and Cathy Escazut. Deceptiveness and neutrality the end family of fitness landscapes. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO' 06, pages 507–514, 2006.

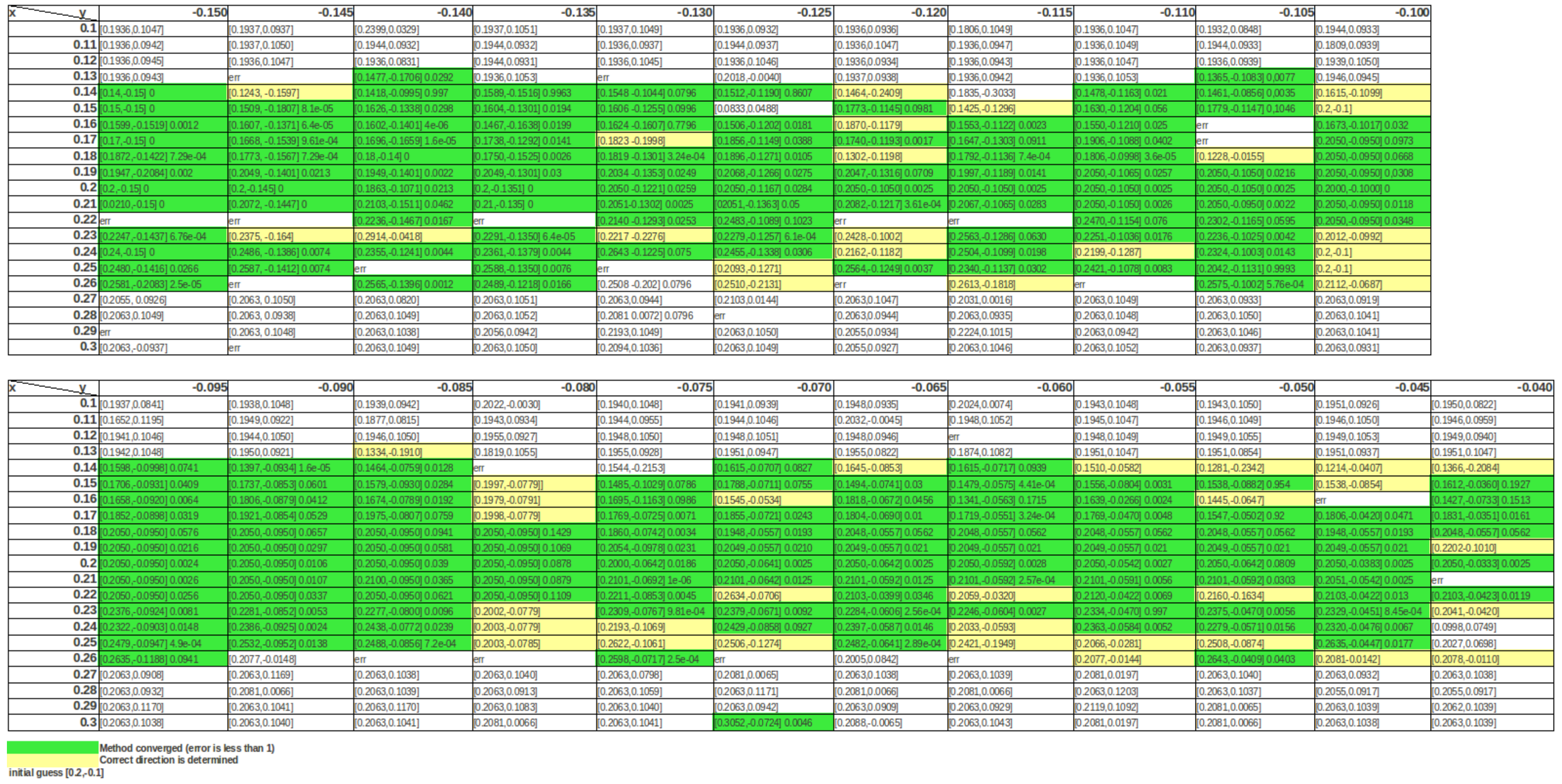
[57] Barrett technology. `http://www.barrett.com/robot/products-hand-specifications.htm`.

[58] Octave/matlab scripting tutorial. `http://openrave.programmingvision.com/wiki/index.php/OctaveMATLAB`.

[59] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the "simulated annealing"algorithm. *ACM Trans. Math. Softw.*, 13:262–280, September 1987.

[60] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

Appendix 1. Results for 2DOF case.



x y	-0.150	-0.145	-0.140	-0.135	-0.130	-0.125	-0.120	-0.115	-0.110	-0.105	-0.100
0.1	[0.1936,0.1047]	[0.1937,0.0937]	[0.2399,0.0329]	[0.1937,0.1051]	[0.1937,0.1049]	[0.1936,0.0932]	[0.1936,0.0936]	[0.1806,0.1049]	[0.1936,0.1047]	[0.1932,0.0848]	[0.1944,0.0933]
0.11	[0.1936,0.0942]	[0.1937,0.1050]	[0.1944,0.0932]	[0.1944,0.0932]	[0.1936,0.0937]	[0.1944,0.0937]	[0.1936,0.1047]	[0.1936,0.0947]	[0.1936,0.1049]	[0.1944,0.0933]	[0.1809,0.0939]
0.12	[0.1936,0.0945]	[0.1936,0.1047]	[0.1936,0.0831]	[0.1944,0.0931]	[0.1936,0.1045]	[0.1936,0.1046]	[0.1936,0.0934]	[0.1936,0.0943]	[0.1936,0.1047]	[0.1936,0.0939]	[0.1939,0.1050]
0.13	[0.1936,0.0943]	err	[0.1477,-0.1706] 0.0292	[0.1936,0.1053]	err	[0.2018,-0.0040]	[0.1937,0.0938]	[0.1936,0.0942]	[0.1936,0.1053]	[0.1365,-0.1083] 0,0077	[0.1946,0.0945]
0.14	[0.14,-0.15] 0	[0.1243, -0.1597]	[0.1418,-0.0995] 0.997	[0.1589,-0.1516] 0.9963	[0.1548 -0.1044] 0.0796	[0.1512,-0.1190] 0.8607	[0.1464,-0.2409]	[0.1835,-0.3033]	[0.1478,-0.1163] 0.021	[0.1461,-0.0856] 0,0035	[0.1615,-0.1099]
0.15	[0.15,-0.15] 0	[0.1509, -0.1807] 8.1e-05	[0.1626,-0.1338] 0.0298	[0.1604,-0.1301] 0.0194	[0.1606 -0.1255] 0.0996	[0.0833,0.0488]	[0.1773,-0.1145] 0.0981	[0.1425,-0.1296]	[0.1630,-0.1204] 0.056	[0.1779,-0.1147] 0,1046	[0.2,-0.1]
0.16	[0.1599,-0.1519] 0.0012	[0.1607, -0.1371] 6.4e-05	[0.1602,-0.1401] 4e-06	[0.1467,-0.1638] 0.0199	[0.1624 -0.1607] 0.7796	[0.1506,-0.1202] 0.0181	[0.1870,-0.1179]	[0.1553,-0.1122] 0.0023	[0.1550,-0.1210] 0.025	err	[0.1673,-0.1017] 0.032
0.17	[0.17,-0.15] 0	[0.1668, -0.1539] 9.61e-04	[0.1696,-0.1659] 1.6e-05	[0.1738,-0.1292] 0.0141	[0.1823 -0.1998]	[0.1856,-0.1149] 0.0388	[0.1740,-0.1193] 0.0017	[0.1647,-0.1303] 0.0911	[0.1906,-0.1088] 0.0402	err	[0.2050,-0.0950] 0.0973
0.18	[0.1872,-0.1422] 7.29e-04	[0.1773, -0.1567] 7.29e-04	[0.18,-0.14] 0	[0.1750,-0.1525] 0.0026	[0.1819 -0.1301] 3.24e-04	[0.1896,-0.1271] 0.0105	[0.1302,-0.1198]	[0.1792,-0.1136] 7.4e-04	[0.1806,-0.0998] 3.6e-05	[0.1228,-0.0155]	[0.2050,-0.0950] 0.0668
0.19	[0.1947,-0.2084] 0.002	[0.2049, -0.1401] 0.0213	[0.1949,-0.1401] 0.0022	[0.2049,-0.1301] 0.03	[0.2034 -0.1353] 0.0249	[0.2068,-0.1266] 0.0275	[0.2047,-0.1316] 0.0709	[0.1997,-0.1189] 0.0141	[0.2050,-0.1065] 0.0257	[0.2050,-0.1050] 0.0216	[0.2050,-0.0950] 0,0308
0.2	[0.2,-0.15] 0	[0.2,-0.145] 0	[0.1863,-0.1071] 0.0213	[0.2,-0.1351] 0	[0.2050 -0.1221] 0.0259	[0.2050,-0.1167] 0.0284	[0.2050,-0.1050] 0.0025	[0.2050,-0.1050] 0.0025	[0.2050,-0.1050] 0.0025	[0.2050,-0.1050] 0.0025	[0.2000,-0.1000] 0
0.21	[0.0210,-0.15] 0	[0.2072, -0.1447] 0	[0.2103,-0.1511] 0.0462	[0.21,-0.135] 0	[0.2051-0.1302] 0.0025	[02051,-0.1363] 0.05	[0.2082,-0.1217] 3.61e-04	[0.2067,-0.1065] 0.0283	[0.2050,-0.1050] 0.0026	[0.2050,-0.0950] 0.0022	[0.2050,-0.0950] 0.0118
0.22	err	err	[0.2236,-0.1467] 0.0167	err	[0.2140 -0.1293] 0.0253	[0.2483,-0.1089] 0.1023	err	err	[0.2470,-0.1154] 0.076	[0.2302,-0.1165] 0.0595	[0.2050,-0.0950] 0.0348
0.23	[0.2247,-0.1437] 6.76e-04	[0.2375, -0.164]	[0.2914,-0.0418]	[0.2291,-0.1350] 6.4e-05	[0.2217 -0.2276]	[0.2279,-0.1257] 6.1e-04	[0.2428,-0.1002]	[0.2563,-0.1286] 0.0630	[0.2251,-0.1036] 0.0176	[0.2236,-0.1025] 0.0042	[0.2012,-0.0992]
0.24	[0.24,-0.15] 0	[0.2486, -0.1386] 0.0074	[0.2355,-0.1241] 0.0044	[0.2361,-0.1379] 0.0044	[0.2643 -0.1225] 0.075	[0.2455,-0.1338] 0.0306	[0.2162,-0.1182]	[0.2504,-0.1099] 0.0198	[0.2199,-0.1287]	[0.2324,-0.1003] 0.0143	[0.2,-0.1]
0.25	[0.2480,-0.1416] 0.0266	[0.2587, -0.1412] 0.0074	err	[0.2588,-0.1350] 0.0076	err	[0.2093,-0.1271]	[0.2564,-0.1249] 0.0037	[0.2340,-0.1137] 0.0302	[0.2421,-0.1078] 0.0083	[0.2042,-0.1131] 0.9993	[0.2,-0.1]
0.26	[0.2581,-0.2083] 2.5e-05	err	[0.2565,-0.1396] 0.0012	[0.2489,-0.1218] 0.0166	[0.2508 -0.202] 0.0796	[0.2510,-0.2131]	err	[0.2613,-0.1818]	err	[0.2575,-0.1002] 5.76e-04	[0.2112,-0.0687]
0.27	[0.2055, 0.0926]	[0.2063, 0.1050]	[0.2063,0.0820]	[0.2063,0.1051]	[0.2063,0.0944]	[0.2103,0.0144]	[0.2063,0.1047]	[0.2031,0.0016]	[0.2063,0.1049]	[0.2063,0.0933]	[0.2063,0.0919]
0.28	[0.2063,0.1049]	[0.2063, 0.0938]	[0.2063,0.1049]	[0.2063,0.1052]	[0.2081 0.0072] 0.0796	err	[0.2063,0.0944]	[0.2063,0.0935]	[0.2063,0.1048]	[0.2063,0.1050]	[0.2063,0.1041]
0.29	err	[0.2063, 0.1048]	[0.2063,0.1038]	[0.2056,0.0942]	[0.2193,0.1049]	[0.2063,0.1050]	[0.2055,0.0934]	[0.2224,0.1015]	[0.2063,0.0942]	[0.2063,0.1046]	[0.2063,0.1041]
0.3	[0.2063,-0.0937]	err	[0.2063,0.1049]	[0.2063,0.1050]	[0.2094,0.1036]	[0.2063,0.1049]	[0.2055,0.0927]	[0.2063,0.1046]	[0.2063,0.1052]	[0.2063,0.0937]	[0.2063,0.0931]

x y	-0.095	-0.090	-0.085	-0.080	-0.075	-0.070	-0.065	-0.060	-0.055	-0.050	-0.045	-0.040
0.1	[0.1937,0.0841]	[0.1938,0.1048]	[0.1939,0.0942]	[0.2022,-0.0030]	[0.1940,0.1048]	[0.1941,0.0939]	[0.1948,0.0935]	[0.2024,0.0074]	[0.1943,0.1048]	[0.1943,0.1050]	[0.1951,0.0926]	[0.1950,0.0822]
0.11	[0.1652,0.1195]	[0.1949,0.0922]	[0.1877,0.0815]	[0.1943,0.0934]	[0.1944,0.0955]	[0.1944,0.1046]	[0.2032,-0.0045]	[0.1948,0.1052]	[0.1945,0.1047]	[0.1946,0.1049]	[0.1946,0.1050]	[0.1946,0.0959]
0.12	[0.1941,0.1046]	[0.1944,0.1050]	[0.1946,0.1050]	[0.1955,0.0927]	[0.1948,0.1050]	[0.1948,0.1051]	[0.1948,0.0946]	err	[0.1948,0.1049]	[0.1949,0.1055]	[0.1949,0.1053]	[0.1949,0.0940]
0.13	[0.1942,0.1048]	[0.1950,0.0921]	[0.1334,-0.1910]	[0.1819,0.1055]	[0.1955,0.0928]	[0.1951,0.0947]	[0.1955,0.0822]	[0.1874,0.1082]	[0.1951,0.1047]	[0.1951,0.0854]	[0.1951,0.0937]	[0.1951,0.1047]
0.14	[0.1598,-0.0998] 0.0741	[0.1397,-0.0934] 1.6e-05	[0.1464,-0.0759] 0.0128	err	[0.1544,-0.2153]	[0.1615,-0.0707] 0.0827	[0.1645,-0.0853]	[0.1615,-0.0717] 0.0939	[0.1510,-0.0582]	[0.1281,-0.2342]	[0.1214,-0.0407]	[0.1366,-0.2084]
0.15	[0.1706,-0.0931] 0.0409	[0.1737,-0.0853] 0.0601	[0.1579,-0.0930] 0.0284	[0.1997,-0.0779]]	[0.1485,-0.1029] 0.0786	[0.1788,-0.0711] 0.0755	[0.1494,-0.0741] 0.03	[0.1479,-0.0575] 4.41e-04	[0.1556,-0.0804] 0.0031	[0.1538,-0.0882] 0.954	[0.1538,-0.0854]	[0.1612,-0.0360] 0.1927
0.16	[0.1658,-0.0920] 0.0064	[0.1806,-0.0879] 0.0412	[0.1674,-0.0789] 0.0192	[0.1979,-0.0791]	[0.1695,-0.1163] 0.0986	[0.1545,-0.0534]	[0.1818,-0.0672] 0.0456	[0.1341,-0.0563] 0.1715	[0.1639,-0.0266] 0.0024	[0.1445,-0.0647]	err	[0.1427,-0.0733] 0.1513
0.17	[0.1852,-0.0898] 0.0319	[0.1921,-0.0854] 0.0529	[0.1975,-0.0807] 0.0759	[0.1998,-0.0779]	[0.1769,-0.0725] 0.0071	[0.1855,-0.0721] 0.0243	[0.1804,-0.0690] 0.01	[0.1719,-0.0551] 3.24e-04	[0.1769,-0.0470] 0.0048	[0.1547,-0.0502] 0.92	[0.1806,-0.0420] 0.0471	[0.1831,-0.0351] 0.0161
0.18	[0.2050,-0.0950] 0.0576	[0.2050,-0.0950] 0.0657	[0.2050,-0.0950] 0.0941	[0.2050,-0.0950] 0.1429	[0.1860,-0.0742] 0.0034	[0.1948,-0.0557] 0.0193	[0.2048,-0.0557] 0.0562	[0.2048,-0.0557] 0.0562	[0.2048,-0.0557] 0.0562	[0.2048,-0.0557] 0.0562	[0.1948,-0.0557] 0.0193	[0.2048,-0.0557] 0.0562
0.19	[0.2050,-0.0950] 0.0216	[0.2050,-0.0950] 0.0297	[0.2050,-0.0950] 0.0581	[0.2050,-0.0950] 0.1069	[0.2054,-0.0978] 0.0231	[0.2049,-0.0557] 0.0210	[0.2049,-0.0557] 0.021	[0.2049,-0.0557] 0.021	[0.2049,-0.0557] 0.021	[0.2049,-0.0557] 0.021	[0.2049,-0.0557] 0.021	[0.2202-0.1010]
0.2	[0.2050,-0.0950] 0.0024	[0.2050,-0.0950] 0.0106	[0.2050,-0.0950] 0.039	[0.2050,-0.0950] 0.0878	[0.2000,-0.0642] 0.0186	[0.2050,-0.0641] 0.0025	[0.2050,-0.0642] 0.0025	[0.2050,-0.0592] 0.0028	[0.2050,-0.0542] 0.0027	[0.2050,-0.0642] 0.0809	[0.2050,-0.0383] 0.0025	[0.2050,-0.0333] 0.0025
0.21	[0.2050,-0.0950] 0.0026	[0.2050,-0.0950] 0.0107	[0.2100,-0.0950] 0.0365	[0.2050,-0.0950] 0.0879	[0.2101,-0.0692] 1e-06	[0.2101,-0.0642] 0.0125	[0.2101,-0.0592] 0.0125	[0.2101,-0.0592] 2.57e-04	[0.2101,-0.0591] 0.0056	[0.2101,-0.0592] 0.0303	[0.2051,-0.0542] 0.0025	err
0.22	[0.2050,-0.0950] 0.0256	[0.2050,-0.0950] 0.0337	[0.2050,-0.0950] 0.0621	[0.2050,-0.0950] 0.1109	[0.2211,-0.0853] 0.0045	[0.2634,-0.0706]	[0.2103,-0.0399] 0.0346	[0.2059,-0.0320]	[0.2120,-0.0422] 0.0069	[0.2160,-0.1634]	[0.2103,-0.0422] 0.013	[0.2103,-0.0423] 0.0119
0.23	[0.2376,-0.0924] 0.0081	[0.2281,-0.0852] 0.0053	[0.2277,-0.0800] 0.0096	[0.2002,-0.0779]	[0.2309,-0.0767] 9.81e-04	[0.2379,-0.0671] 0.0092	[0.2284,-0.0606] 2.56e-04	[0.2246,-0.0604] 0.0027	[0.2334,-0.0470] 0.997	[0.2375,-0.0470] 0.0056	[0.2329,-0.0451] 8.45e-04	[0.2041,-0.0420]
0.24	[0.2322,-0.0903] 0.0148	[0.2386,-0.0925] 0.0024	[0.2438,-0.0772] 0.0239	[0.2003,-0.0779]	[0.2193,-0.1069]	[0.2429,-0.0858] 0.0927	[0.2397,-0.0587] 0.0146	[0.2033,-0.0593]	[0.2363,-0.0584] 0.0052	[0.2279,-0.0571] 0.0156	[0.2320,-0.0476] 0.0067	[0.0998,0.0749]
0.25	[0.2479,-0.0947] 4.9e-04	[0.2532,-0.0952] 0.0138	[0.2488,-0.0856] 7.2e-04	[0.2003,-0.0785]	[0.2622,-0.1061]	[0.2506,-0.1274]	[0.2482,-0.0641] 2.89e-04	[0.2421,-0.1949]	[0.2066,-0.0281]	[0.2508,-0.0874]	[0.2635,-0.0447] 0.0177	[0.2027,0.0698]
0.26	[0.2635,-0.1188] 0.0941	[0.2077,-0.0148]	err	err	[0.2598,-0.0717] 2.5e-04	err	[0.2005,0.0842]	err	[0.2077,-0.0144]	[0.2643,-0.0409] 0.0403	[0.2081-0.0142]	[0.2078,-0.0110]
0.27	[0.2063,0.0908]	[0.2063,0.1169]	[0.2063,0.1038]	[0.2063,0.1040]	[0.2063,0.0798]	[0.2081,0.0065]	[0.2063,0.1038]	[0.2063,0.1039]	[0.2081,0.0197]	[0.2063,0.1040]	[0.2063,0.0932]	[0.2063,0.1038]
0.28	[0.2063,0.0932]	[0.2081,0.0066]	[0.2063,0.1039]	[0.2063,0.0913]	[0.2063,0.1059]	[0.2063,0.1171]	[0.2081,0.0066]	[0.2081,0.0066]	[0.2063,0.1203]	[0.2063,0.1037]	[0.2055,0.0917]	[0.2055,0.0917]
0.29	[0.2063,0.1170]	[0.2063,0.1041]	[0.2063,0.1170]	[0.2063,0.1083]	[0.2063,0.1040]	[0.2063,0.0942]	[0.2063,0.0909]	[0.2063,0.0929]	[0.2119,0.1092]	[0.2081,0.0065]	[0.2063,0.1039]	[0.2062,0.1039]
0.3	[0.2063,0.1038]	[0.2063,0.1040]	[0.2063,0.1041]	[0.2081,0.0066]	[0.2063,0.1041]	[0.3052,-0.0724] 0.0046	[0.2088,-0.0065]	[0.2063,0.1043]	[0.2081,0.0197]	[0.2081,0.0066]	[0.2063,0.1038]	[0.2063,0.1039]

🟩 **Method converged (error is less than 1)**
🟨 **Correct direction is determined**
initial guess [0.2,-0.1]

Figure A1.1. Numerical results of method working in 2DOF case.