Bachelor Thesis

MODULAR SOFTWARE ARCHITECTURE FOR COGNITIVE ROBOTS PERFORMING MANIPULATION TASKS

March 14, 2013

Machiel Bruinink
Technical University of Delft
m.bruinink@student.tudelft.nl

Supervisors:
Dr.ing. M. Rudinac
Dr. W. Caarls
Dr.ir. D.J. Broekens

Contents

1 Introduction **1**

2 Problem Statement **3**
2.1 Requirements 4
2.1.1 Functional Requirements 4
2.1.1.1 Functional requirements for software architecture 4
2.1.1.2 Functional requirements for object manipulation 5
2.1.2 Technical Requirements 6
2.2 MoSCoW 6
2.2.1 Must have 7
2.2.2 Should have 7
2.2.3 Could have 8
2.2.4 Won't have 8

3 Related Work **10**
3.1 Software Architecture 10
3.2 Object Manipulation 11

4 Delft Personal Robot Platform **14**
4.1 Mobile Platform 15
4.2 Arm and Gripper 15
4.3 Neck and Head 15

5 Software Architecture **18**
5.1 Core 18
5.2 Subcores 20
5.3 Individual Modules 21
5.4 Low-Level 22

6 Active Grasping **23**
6.1 Object localization, selection and exploration 23
6.2 Grasping process 25
6.3 Approach and grasp phase 25
6.3.1 Planar Error Correction 25

6.3.2 Depth error correction 27
6.4 Recovery Mechanisms 28

7 Experimental Setup and Results **30**
7.1 Software Architecture Testing and Application 30
7.1.1 Doctor Demo 30
7.1.2 Who is Who 31
7.1.3 MoSCoW Requirements Analysis 31
7.2 Grasping experiments 33
7.2.1 Experiment 1: Grasping precision in case of grasping objects from a table top 34
7.2.2 Experiment 2: Grasping precision in case of different heights 36
7.2.3 Experiment 3: Interactive grasping 36
7.2.4 MoSCoW Requirements Analysis 37

8 Conclusion **41**

A Teamwork **43**

B Task Description **45**

C RoboCup@Home Tasks **47**
C.1 Who Is Who 47
C.1.1 Focus 47
C.1.2 Task 47
C.1.3 Additional rules and remarks 48
C.1.4 Referee Instructions 49
C.1.5 Organizing Committee Instructions 49
C.2 Demo Challange 49
C.2.1 This year's focus 50
C.2.2 Task 50
C.2.3 Presentation 50
C.2.4 Changes to the environment 51
C.2.5 Jury evaluation 51
C.2.6 Additional rules and remarks 51

D Software Improvement Group **53**

Abstract

In the near future a large increase in elderly population is expected. One of the solutions to this problem is to introduce service robots which can provide a day to day care, allowing elderly to live longer independently. However performing basic household tasks proposes many challenges as the robot will have to act in a dynamic environment and reuse its previous knowledge to adapt to novel situations. Therefore this thesis introduces a modular and reusable software architecture able to efficiently and robustly solve a variety of complex tasks in household environments. Most of such tasks require robust object manipulation independent of object properties, location and orientation or motion of the object. In order to solve this problem most of the state of the art methods require prior knowledge on object shape properties, optimal points to grasp it as well as the best grasping configurations. In this thesis a novel method is presented that allows manipulation of unknown objects for which no prior information on object properties or environment is necessary and all grasping knowledge is built online. This method consists of locating the object, tracking the object and finally grasping the object using a constant feedback loop from the object tracker. This method together with the software architecture have been implemented and tested on the robot Robby, equipped with a mobile base, a single DOF robot arm and an underactuated gripper. Through numerous tests performed in household environments benefits of using the proposed approach is shown.

Chapter 1

Introduction

Research in design of personal robots is rapidly growing. Over the past few years novel results in areas of vision and control brought robots much closer to their use as personal assistants in home environment. Bringing robots to homes and elderly care centres is also becoming more essential as the number of elderly people is increasing whilst the number of young people to take care of the elderly is decreasing [18]. This problem can be solved by using robots to perform daily household tasks that are difficult for elderly or disabled people, these include: cleaning the house, getting the mail, fetching and delivering food or drinks, cooking a meal, setting the dining table, and many others. A robot in the house might also have to serve drinks or snacks to the host and its guests or to set up the dining table and serve guests during dinner. Or, for example, a service robot may be placed in an elderly care center, where it may have to select and pick medicine and bring these to a patient. All these actions involve highly developed manipulation skills as well as developed "brain" that will allow planning and execution of different actions.

Example of such a robot is PR2 (figure 1.1(a)), a personal robot for research and innovation developed by Willow Garage. It is a household robot that can navigate and manipulate objects in a household environment. For object manipulation PR2 uses information from different sensors to perceive the world, allowing it to find objects and grasp them [10]. Another example is REEM, a personal robot designed by PAL Robotics (figure 1.1(b)). REEM is the robot which performs tasks such as guiding, providing information and entertaining [1] and has good human interaction skills. All these robots compete in an annual service robotic competition Robocup@Home [2] that tries to simulate the challenges of a real environment and to test the most common household scenarios. Main focus of this thesis is to design architecture and algorithms necessary for a robot to perform well in this competition.

All tasks preformed by personal robots can be divided into many smaller actions, for example navigating from point 'a' to point 'b', recognizing an object among many other objects, grasping an object or tracking an object. By combining simple actions a more complex task can be performed. Keeping the

(a) PR2 Willow Garage

(b) REEM Pal-Robotics

Figure 1.1: Examples of personal robots providing assistance.

actions more general will allow them to be reused as a part of many different complex tasks. This thesis introduces such a system architecture that integrates these action modules to complete a complex household tasks and reuses these modules to execute other varying tasks.

Manipulation of objects belongs to almost all complex task performed in the household, making object manipulation one of the most essential actions that must be very robust and efficient on service robots. Therefore, this thesis introduces a novel active grasping method developed for a service robot equipped with an underactuated gripper. Object localization and tracking are two key components of the proposed grasping method, giving the robot the capability of grasping moving objects. The requirements of the grasping method and its implementation are discussed in further detail.

The outline of this thesis is as follows. In Chapter 2 problem statement is defined as well as requirements for software design given. Next, in Chapter 3 related work on architecture of personal robots is presented and drawbacks of state of the art object manipulation algorithms shown. In Chapter 4 specific robotic platform and hardware that is used for testing is presented. Chapter 5 introduces software architecture design while the active grasping method is proposed and explained in Chapter 6. Finally, detailed experimental setup and results can be found in Chapter 7 and conclusion in Chapter 8.

Chapter 2

Problem Statement

Service robots providing assistance within the home or care centres need to be able to perform many complex tasks such as cleaning up, delivering drinks, etc. All these actions demand robust object manipulation. The object manipulation in a personal robot is one of the most essential functionalities and it includes both localization and picking up of the required object. In some cases the robot might also be asked to manipulate unknown objects for which it has no prior knowledge.

One of the scientific challenges in a home environment is that the environment is actively changing over time. Illumination conditions differ during the day, there might be people walking around, kids playing on the floor, chairs or other furniture might have been moved as well as many other small objects. Thus the robot must safely act in an unknown, dynamic and cluttered environment without crashing or failing. This posses a challenge for robot software design as the system must act accordingly to the changes in the environment[14] and should reuse its previous knowledge to be able to act in novel situations. Therefore in this thesis first a modular software architecture is implemented that allow performing complex tasks in home environment. Secondly a novel method is introduced to allow for robust object manipulation on a simplistic robotic platform. The definition of the problem this thesis aims to resolve is the following:

Implement a modular and reusable software architecture able to efficiently and robustly solve a variety of complex tasks.

Implement a robust and effective method to manipulate objects in an unknown environment using a simplistic robot platform. The method needs to cope with both known and unknown objects as well as the different object positions/grasping heights.

The first contribution of this thesis lies in a software architecture developed

to easily interface different modules within the system to work together to perform complex tasks. An individual module is not only used for one specific task but can be part of many different complex tasks. The software interface should handle existing modules as well be capable to easily expand to handle new modules developed in future.

The most essential module is the grasping module and novel algorithms for object manipulation in the unknown environment are necessary to be developed. Therefore the second contribution of this theses lies in a grasping algorithm capable of manipulating both known and unknown objects from different grasping heights. This algorithm needs to handle object manipulation in cluttered environments and interactive grasping cases. The next section will discuss the requirements to resolve problems stated above. Both functional and non-functional requirements are discussed.

2.1 Requirements

The main requirement is to solve the problem using robot platform Robby discussed in Section 4. Other functional and non-functional requirements are described in the following section followed by a requirements list based on the MoSCoW method.

2.1.1 Functional Requirements

Functional requirements define the requirements of the task (function) that must be accomplished. To have a functional robot a software architecture is required. The function of the software architecture is to combine all simple function of the robot so that it can complete a complex task. To solve the problem of object manipulation, functional requirements consist of locating an object, tracking this object and finally grasping it. The final requirement is to use such manipulation algorithm in the proposed architecture, such that it is robust, fail-safe and communicates with other modules of the system. An example of a statemachine that combines different modules to complete a manipulation task is shown in Figure 2.1.

2.1.1.1 Functional requirements for software architecture

The robot software system is required that consist of different modules which must work together to accomplish a complex task. Each module has no information about other modules and how these work. A software architecture system is required which integrates the different modules to accomplish a complex task. Figure 2.1 is an example of such a module which commands other lower level modules to perform their task.

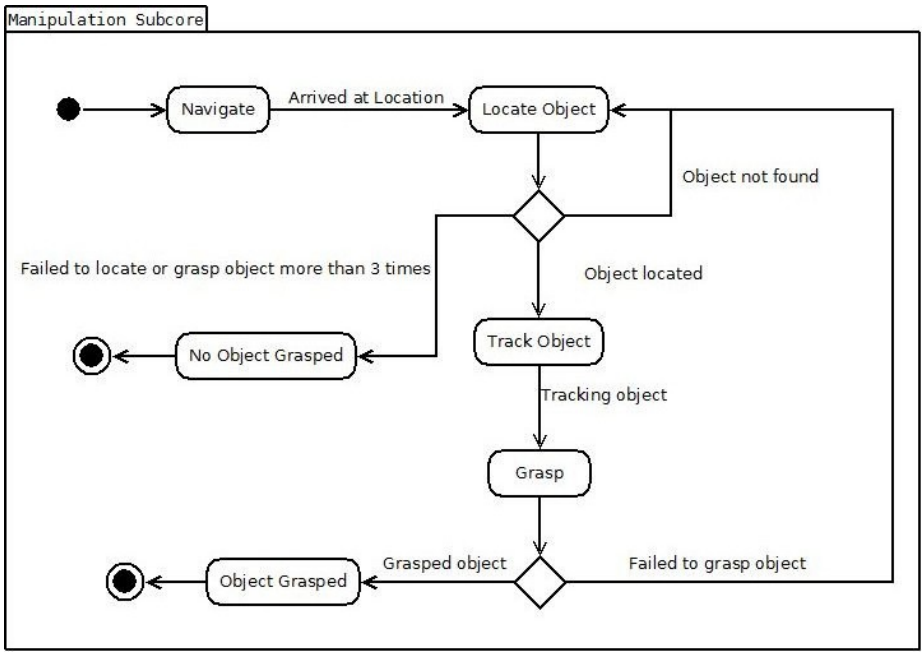


Figure 2.1: A state machine of an object manipulation task.

2.1.1.2 Functional requirements for object manipulation

Object Localization In order to locate a desired object in a cluttered background a segmentation algorithm is required. By segmenting an image all the different objects in the scene can be located. In the case the robot is looking for a specific known object, the segmented objects should be recognized by matching it to a database of known 2D object models. In the case that the object to be manipulated is unknown to the robot, selection algorithm will choose an object to be manipulated and its appearance model should be learned during the process. In either case of manipulating a known or unknown object an grasping algorithm must be designed in such a way that a full 3D model of the object is not required.

Object Tracking To actively grasp an object the location of the object must be updated continuously. Therefore to keep constant track of an object while grasping it despite challenges of home environment robust tracking algorithm is required. Constant interaction between grasping and tracking should allow the robot to grasp the object even in the case the object is dynamical moved. To allow fail safe scenario, when track of the object is either lost or regained this information must be communicated to the other modules so that the robot can act accordingly.

Object Grasping A grasping algorithm is required which uses constant feedback from the tracker to grasp the object. Grasping process must be able to grasp both known and unknown objects of both simple or complex shape. Another important function of the grasping algorithm is to ensure that in the case it fails to grasp the desired object it communicates this correctly to the rest of the robot or it tries to grasp the object again. It is also required that the robot does not collide with any object during the grasping process.

2.1.2 Technical Requirements

Technical requirements are non-functional requirements but are quite important to be considered. The technical requirements include real time performance, grasping precision and modularity of the software architecture.

Real time performance is a non-functional requirement as it does not affect the functionality of the problem stated. The duration of the grasping task is highly dependent on the distance to the object, the maximum movement speed of the robot as well as the speed of the object tracker. Entire grasping procedure in case of manipulating both known and unknown should be less than a minute. This requirement is defined by the time a disabled or elderly person requires to grasp an object. The robot should be atleast as efficient as the person that it is helping.

The precision of the grasping module is crucial to the general performance of the robot since this is one of the most important functionalities. The performance of grasping is highly depended on the object localization and tracking methods. The performance of the grasping algorithm is measured by performing a series of test where robot has to grasp both known and unknown objects from different heights. In order for a robot to be efficient it is required that grasping has a success rate of above 80% despite the complexity of the object shape or environmental conditions. A success rate lower than 80% could cause users to view the robot as ineffective.

Last non-functional requirement is modularity of software architecture. Each module in the system should have its own functionality. Every module should be independent but may use other modules to achieve the function required. Making the system modular allows for easy maintenance and updating of individual modules. Making changes in a module should have no effect on other modules as long as the functionality of the module is not changed. For example if the object tracker is updated to a new and better object tracker the grasping algorithm should still function without having to make any changes to it.

2.2 MoSCoW

The MoSCoW method divides the requirements into four categories: must have, should have, could have and won't have[3]. For each category the requirements related to this thesis will be listed.

2.2.1 Must have

Must have requirements are requirements which are considered to be necessary in order for a method to be successful. If these requirements are satisfied the robot will be able to perform basic tasks.

- Software Architecture
  - Must be modular.
  - Must be able to combine different modules to accomplish a complex task.
  - Must be applicable on robot Robby.
  - Must be usable to accomplish all RoboCup@Home tasks.
- Active Grasping
  - Robot must locate atleast one object in the scene and find the position of this object in relation to the robot.
  - Robot must track the object and constantly update the relative position, to assist the grasping process. While the arm is moving, the object should be static.
  - If the tracker looses the object it must communicate this correctly to the robot core and the grasping procedure must be stopped.
  - Robot must move its arm till the gripper is within proximity of the object and pick up this object.
  - Robot must be able to grasp from a table top.
  - Robot must be able to grasp objects with a simple shape (boxes, cans, etc.).

2.2.2 Should have

The should have requirements are requirements which are not critical for the method to function but have a very high priority.

- Software Architecture
  - System modules should be reusable such that they can be part of different complex tasks.
  - System architecture should be usable on different robotic platforms with a different body morphology.
  - Should have a fail-safe mechanism, in the case of a module failure.
- Active Grasping
  - Robot should be able to grasp an unknown object.

  - Robot should localize multiple objects in the scene and select which object to grasp.
  - If the tracker has temporarily lost the object position, the robot should continue grasping when track of object is regained.
  - Robot should correctly grasp a dynamically moving object.
  - Robot should correctly handle the case that the desired object could not be grasped by communicating this to the core of the robot.
  - Robot should be able to grasp objects from different heights.
  - Robot should be able to grasp objects with a complex shape.

2.2.3 Could have

Could have requirements will not affect the effectiveness of the method, but will be useful to implement if time and resources are available.

- Software Architecture
  - Use modules in a computational and energy efficient way.
  - Adjust the action based on the information given.
  - Optimal division of tasks and sub tasks, such that there are no two task having similar functionalities.
- Active Grasping
  - Be able to grasp the object even if the object is touching an other object.
  - Grasp an object of the same or similar category if the user requested item is not available.
  - Update and learn the object appearance model while manipulating the object.
  - Detect whether an object is graspable before executing the grasping algorithm.
  - Grasp objects from constrained surfaces such as an open fridge or a bookshelf.
  - Avoid all obstacles while executing the grasping algorithm.

2.2.4 Won't have

The won't have requirements are requirements which might be considered for future work but will not be implemented in this thesis.

- Software Architecture
  - System architecture will be able to decide automatically which modules are required to complete a complex task.

  - Will be able to automatically re-plan its actions by using other available modules, in the case a module fails.

- Active Grasping
  - Robot will be able to grasp objects that have a vertical grasp orientation.
  - Detect a cabinet door or fridge door and open it before finding and grasping the desired object.
  - Move objects that are obstacles to clear a path to the desired object to be grasped.

Chapter 3

Related Work

In this chapter related work of both robot system software architecture and grasping algorithms are discussed. The advantages and disadvantages of such existing methods are argued and possible applications towards solving the problems stated in this thesis are proposed.

3.1 Software Architecture

A robots system architecture is an essential part of the system as it defines the possibilities and limitations of a robot. In this section past and current work on such systems is discussed and evaluated how they impact the functionalities of a robot.

In the late 1960s robot Shakey developed at Stanford University used an architecture that was divided into three functional elements: sensing, planning and executing also as the sense-plan-act(SPA) approach [24]. Each of the elements in the this approach are serially linked and thus executed in a specific sequence(first sense, then plan and finally act). The problem with the SPA approach is that the robot was depended on a static environment as it could fail or propose danger if the environment changed while the robot was executing the plan. In the early 1980s new robot architectures were introduced that relied on online planning of the actions such that a robot could safely act in a dynamic environment. One of the most influential works during this time was the subsumption architecture of R. Brooks [11]. Brooks designed an architecture build out of layers of interacting finite-state machines. The state machines were called behaviors. Activating multiple behaviors at the same time opened up the possibility for higher-level behaviors to over rule the actions of a lower-level behavior. For example the execution of driving in a certain direction could be overwritten by a higher-level behavior which was responsible for ensuring the robot does not collide into any obstacles. Many robots at this time started using this approach as it was a significant improvement to other approaches available at the time. However using the behavior-based approach it is very

difficult to compose complex behaviors capable of achieving long-ranged goals and that allow the robot to reason. The next step in robot architectures was the developments of layered robot architectures.

Layered robot architectures are capable of combining different behaviors to complete complex tasks and allow for reasoning withing such a layer. One of the first developments towards such a system was the reactive action package(RAPs) developed by R. Firby [15]. Firby is the first to introduce a three-layer architecture consisting of a planner, execution and robot control layer. Each of the three layers has a specific functionality. The planning layer is responsible for receiving the goal and generating a sequence of tasks to complete a particular goal. Execution is responsible taking a plan from the planner and dividing each task into more specific details based on the environment. Finally the robot control layer is for receiving the actions from the execution layer and translating these to the robot hardware level. The three layered architecture is very popular and has been implemented on many different systems, some examples of three layered architecture systems are: ATLANTIS[16], Saridis[26], LAAS [6], Remote Agent [23] and many more exist.

To design the architecture proposed in this thesis a similar approach to the three-layer architecture was used. The architecture design has been adapted to meet the requirements of the Robocup@Home league [8]. The designed architecture uses two of the three layers, the execution and robot control layer. The execution layer consists of the a core module and several subcore modules. A core module is responsible for dividing a complex task into more specific simplistic tasks using the available subcore modules. Subcore modules define more generic simplistic tasks which combined can perform a complex task. The second layer, the robot control layer, consists of individual modules that translate the actions from the execution layer to the actual robot hardware. The higher level planning layer was not implemented as the tasks at Robocup@Home are statically defined. Although the architecture has been adapted, it is designed such that it can be extended with the third layer of high-level planning. The architecture is also designed such that it is usable for other household applications not applicable to Robocup@Home. Such an architecture must be robust and fail-safe, fail-safe mechanisms have been implemented at different levels of the proposed architecture. Further details of the proposed system architecture are discussed in chapter 5.

3.2 Object Manipulation

In order for a service robot to efficiently accomplish tasks within a household environment robust manipulation skills are required. Within such an environment robots should be capable of dealing with both known and unknown objects, manipulate object from different heights and acquire all grasping knowledge online. To commercialize household robots they need to be affordable, therefore a simple body with minimal complexity should be designed. This adds additional constraints to the grasping algorithm. There have been many imple-

mentations of grasping algorithms (e.g [27],[9],[10]) yet there are few which use a simplistic robot and vision feedback to continuously update the arm trajectory to robustly grasp an object which is unknown to the robot. Also most current methods require prior grasping knowledge. Prior work on grasping and vision based grasping will be discussed in detail in this section.

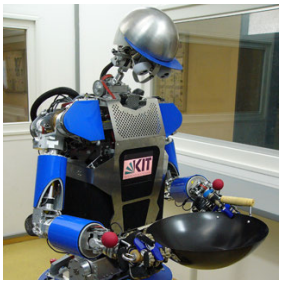
Reem (see figure 3.1(a)) a robot developed by PAL-Robotics [1] is an example of a commercial robot with two 6 DOF arms placed on a mobile platform. Reem uses prelearned knowledge of objects and optimal grasping points to perform a successful grasp. This method proposes a problem when it comes to grasping of unknown objects of which no prior knowledge is available. Cosero (figure3.1(b)) a robot developed at Bohn University and winner of the RoboCup@Home competition is another example of a complex robot. It has a total of 30 ROF, consisting of a mobile base and two 7 DOF arms[29]. Cosero uses online 3D point-cloud data to solve the problem of grasping unknown objects. The method used solves for the optimal grasping points based on the model obtained online. For object manipulation both Reem and Cosero use standard methods based on global path planning[20] followed by inverse kinematics [30]. These methods are based on an open loop approach where the robot executes a preplanned trajectory and requires the use of complex and expensive hardware to achieve precise grasping. In the case of a dynamic environments such methods can fail in the situations where the object is moved by external factors. To solve this problem in this thesis a grasping algorithm using constant vision feedback is proposed.

ARMAR-III (figure 3.1(c)) is another complex robot with a total of 43 DOF consisting of two complex arms each with 7 DOF and two hands each with a total of 8 DOF[7]. The ARMAR-III uses an approach known as Rapidly Exploring Random Trees (RRT)to plan its grasping [31]. The RRT method consist of three main tasks: finding a feasible grasp, solving the inverse kinematics for this grasp and searching a collision-free trajectory to bring the hand to the grasping pose. The ARMAR-III is capable of grasping unknown objects using online computation of the possible grasp poses and arm trajectories resulting in a optimal grasp. The PR2 (see Figure 3.1(d)) is a commercialized robot, costing around half a million Euros, that is being used by different research groups around the world. It is a two-armed wheeled robot with complex 7 DOF arms and actuated gripper able to grasp household objects. Many different grasping methods have been implemented on the PR2, one of these methods is introduced in [13]. The method used here allows for both grasping of known and unknown objects. For the known objects grasping points are calculated offline using the *GraspIt!* simulator[22]. For unknown objects grasping points are computed from 3D sensor data using heuristics based on the shape of the object and its local features. The method computes a motion plan based on the environment and the optimal grasping points, it then executes the planned motion to the grasping point and gives feedback of whether or not the object was successfully grasped. However one problem is that the ARMAR-III and PR2 require prior knowledge of grasping force for different types of objects, in order to achieve a stable grasp. This problem is approached in this thesis from a

mechanical perspective using an underactuated, adaptive gripper [19]. Another problem with these robots is their complexity and high costs to be realized on a household robot market. Therefore one of the requirements of this thesis is the use of a simplistic affordable robot platform.

(a) PAL-Robotics Reem

(b) Cosero



(c) ARMAR-III

(d) Willow Garage PR2

Figure 3.1: Robots capable of grasping

Chapter 4

Delft Personal Robot Platform

Robot Robby (figure 4.1), the robot used in this thesis, is based on the Delft personal robot 2 (DPR2) a simplistic and affordable robot platform. Though the design might seem simplistic, due to smart control and smart mechanics it can perform complex tasks.

Figure 4.1: Robot Robby

The DPR2 platform consist of a mobile base with a one degree of freedom (DOF) pivoting arm. Robot Robby has been extended with a gripper, a head

and additional sensors. A detailed explanation of the robot configuration will follow in the next sections.

4.1 Mobile Platform

The mobile platform of the robot shown in figure 4.2, consist of four wheels of which two are driven wheels and two are caster wheels. The caster wheels allow the robot to steer while the two driven wheels represent the differential drive mechanism. The mechanical design of the platform reduces the overall cost of the robot but provides a scientific challenge of using non-holonomic wheels for navigation[8]. In addition to the DPR2 platform, robot Robby has been equipped with a laser scanner on the base. The laser scanner provides information about the surrounding environment which is necessary for autonomous navigation.

4.2 Arm and Gripper

The DPR2 has a simplistic mechanical arm design with one degree of freedom (see figure 4.3(a)). The design of the arm uses a counter weight to keep the robot mechanically stable, while the counter weight also provides a housing for the electronics, battery and a laptop [8]. The arm has been extended with an underactuated gripper (Delft Hand 3) as shown in figure 4.3(b)[19]. The gripper is controlled using a single motor while having the ability to adapt its fingers around objects of different sizes and shapes. This property of the gripper will be highly used in developing object manipulation algorithm that can handle unknown objects. Additionally a proximity sensor which serves as a touch sensor has been mounted on the gripper. It can be used to detect that an object is grasped by the gripper as well as to measure the distance of an object up to 40cm away from the the gripper.

4.3 Neck and Head

Robot Robby has been equipped with a two degrees of freedom neck, arranged in a pan and tilt configuration. On top of the neck a head with a Kinect, high resolution camera, speakers and microphone have been placed (see figure 4.4). The Kinect provides the robot with 3D information of the world acting as the peripheral vision system and allows detection of the structure and shape of objects as well as their localization. The high resolution camera provides a high resolution image (about twice the resolution compared to the Kinect) and is used as the focal vision system of the robot. The high resolution image is thus used for object recognition especially for recognition of texture, color and edge information of objects. Calibrating the Kinect and high resolution camera the peripheral and focal visions system are combined to allow for robust object localization and recognition using both 2D and 3D input.

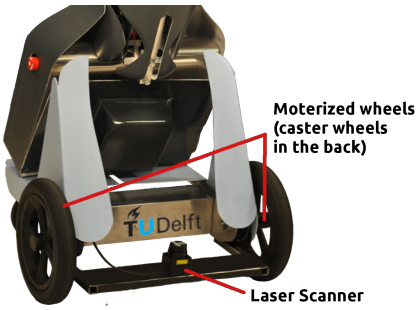
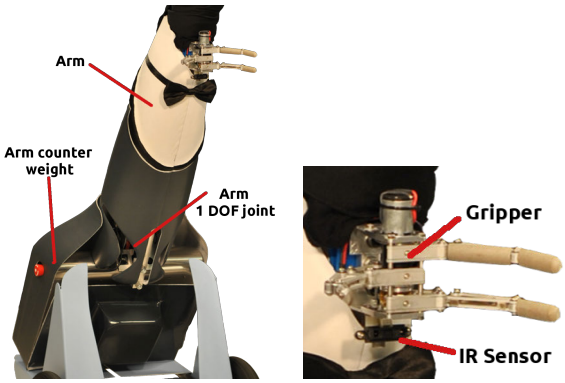


Figure 4.2: Robot base with range finding sensor.



(a) Robot arm and counter weight

(b) Robot hand with proximity sensor

Figure 4.3: Robot arm and hand.

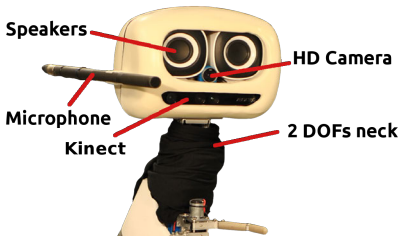


Figure 4.4: Robot head with a 2 DOFs neck.

The speakers and microphone on the head are used as communication devices on the robot, where the microphone can be seen as the ears of the robot and the speakers as the mouth. As it is the most natural way for humans to communicate is through speech, using a speech system on the robot should make it easy for humans to interact with the robot.

Chapter 5

Software Architecture

Robot Robby has been equipped with a basic client-server software architecture. The architecture has been divided into 2 layers the execution-layer and the robot-control-layer. The execution-layer is divided into two sub-layers the core and subcores. The robot-control-layer is also divided into two sub-layers individual modules and low-level control. All the layers of the software architecture are depicted in Figure 5.1. Every layer in the system is designed to be modular and has its own specific task, allowing any part of the system to be changed or updated without affecting the other parts of the system unless the functionality of such a module has been modified. The software architecture has been designed to work with Robot Operating System (ROS). ROS provides many existing packages and drivers built specifically for robots and it provides a client-server communication interface for passing messages between different nodes running on the robot[4].

5.1 Core

The core acts as the brain of the robot, it is responsible for making the decisions and executing the requested task. The robot receives the task commands from the user through speech. When the robot receives a task from the user, it will formulate a list of planned steps that have to be executed in a specific order to complete the required task. To further formulate the steps, the robot must know at all times what its current state is. Therefore, a state machine has been implemented to keep track of the robot's current state and to initiate the modules performing the planned steps. Once a step is completed the core is informed so that the next step can be initiated. When all steps have been executed, the core informs the user of the task being completed and waits for the next user command.

The implemented state machine contains a list of states for a certain complex task. Each state only has knowledge of what activity has to be executed, what the next state should be and what the previous state was. The activity executed

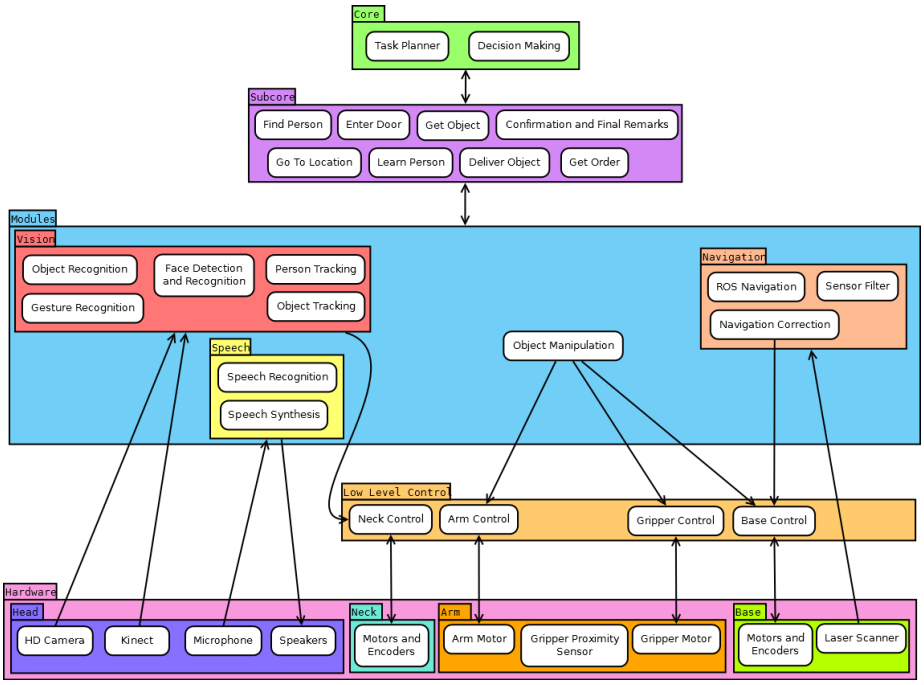


Figure 5.1: Diagram showing the software architecture.

by a state in the core is considered as a subcore. In Figure 5.2 an example of a state machine for the "Who is Who" core is shown. For "Who is Who" scenario, the robot has to act as a waiter and serve drinks to a group of people. At first it has to detect persons calling the waiter, take their orders, navigate to a bar, grasp desired drinks and bring them back to correct customers. The state machine contains all states/subcores required to execute such a complex task including learning person, going to a specific location or grasping a drink. Going from one state to the next requires that specific requirements have been met. For example, moving from the "go to location" to the "grasping" state requires that the robot has reached the destination where the object is located.

One of the benefits of such state representation is modularity of software architecture. In Figure 5.2 it can be seen how are different states/subcores being reused to complete a complex task. For example, "find person" subcore has been used in multiple situations with different environmental settings to detect people seeking robot attention. To prevent a system failure in the case that one of modules fails to execute specific action, fail-safe mechanisms have been implemented to allow a robot to continue with execution of a given task. In such cases more than one possible state is possible depending on the outcome of the action performed. In Figure 5.2 an example of failed grasping action is shown. In such a case, instead of the robot going straight back to the user to report the failure, it continues to the next order and informs all users at the end

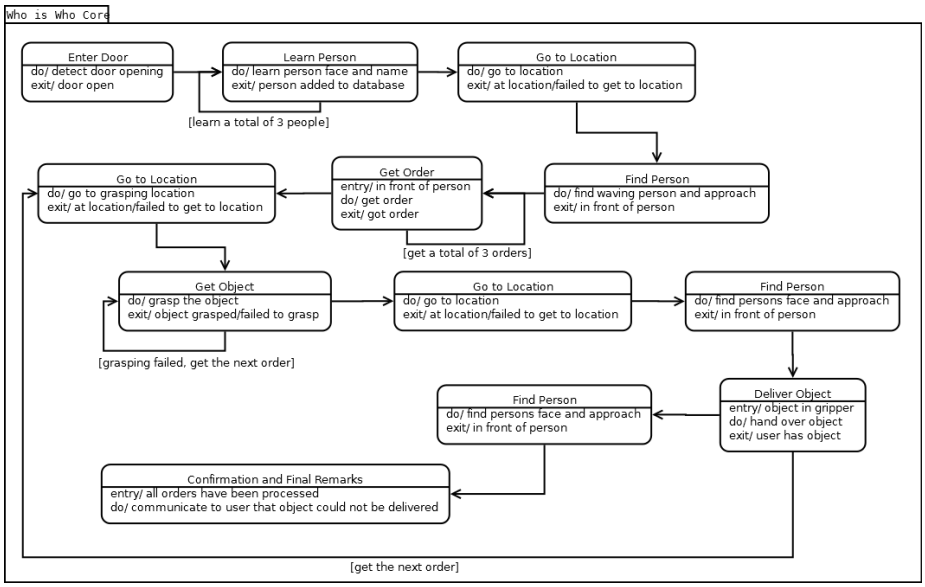


Figure 5.2: Diagram showing the Who is Who core.

of the completed task that it was unable to provide a certain order.

Regrading communication and data flow, one communication channel between the core and all subcores has been created using ROS messages. Every subcore is listening to the topic being published by the core. A message from the core contains a paramater stating the subcore which it is addressed to. In that way only the required subcore will respond to the message. Next the message contains the action the subcore has to execute, which is either a 'start' or 'stop' action commanding the subcore to start or stop its designed functionality. Finally, the message can contain several parameters providing additional information required for the action the subcore has to perform, eg. the specified drink the robot has to grasp, or to whom it should bring the order.

In the next part detailed description of subcores will be given.

5.2 Subcores

Each subcore has been created as generic as possible so that it can be used as part of many different complex tasks. For example, finding a person subcore can be used in the tasks involving bringing a person a drink or also finding a person in emergency situations. The subcore initiates the individual modules based on the input given by the core, and returns its output to the core.

The "find person" subcore is represented in the state diagram in Figure 5.3. Each of the actions within the diagram can be directly related to an individual module in the system. As can be seen in the diagram the robot is looking for a

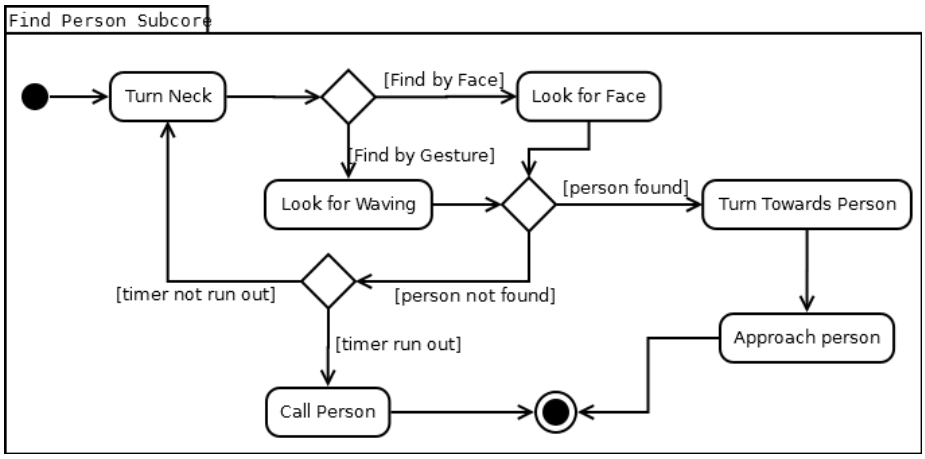


Figure 5.3: Diagram showing the find person subcore.

waving person in the room. It is doing so by continuously turning the neck and trying to detect the waving gesture. To avoid that the search takes a long time or in case the waving detection fails, a timer has been set so that after the time elapses robot calls a person to approach. If the waving person is found before time runs out the robot turns toward this person and approaches him. As can be seen in 5.2 the find person subcore can also be initialized giving the name of a person in which case robot searches for the specific person instead of any person waving to the robot.

As a subcore finishes its task it sends a response to the core using a ROS message. The message to the core contains the information about the subcore which it is coming from as well as the information about the final output of the subcore stating whether it was successful or possible there was some failure. The communication between the subcore and the core is very important since the core must be informed when it has to continue to the next state. In the case a subcore has failed, the core is informed so that it can take actions accordingly to prevent failure of the entire system. Additionally subcores have its own fail-safe mechanisms in the case a individual module fails as can bee seen in Figure 5.3. Finally a subcore output message can contain knowledge acquired about the environment or action performed such as the name of a learned face or the name of a location.

5.3 Individual Modules

Individual modules are responsible for the main actions of the robot. These include navigation, manipulation, speech recognition, face recognition, object recognition, object tracking as well as person tracking. These modules interact with the low-level control layer to receive and process input directly. For ex-

ample the speech recognition module constantly receives audio input from the microphone driver and process this input into a string variable. Another example is the face recognition module as it receives the video feed from the camera it puts a name to each face in the frame. Individual modules are responsible for constantly returning their processed output or when it is requested.

Subcores interact directly with the individual modules. The communication between these two layers is in the form of ROS messages, services or actions. Since the functionality of the individual modules can be very different from each other a separate communication method is required for each of them. Therefore every individual module uses a unique form of communication through the available ROS protocols. For example the navigation stack uses ROS actions to receive the navigation goals and publish the status of navigation, yet speech recognition uses just a single parameter ROS message to communicate the audio input translated to a string.

5.4 Low-Level

Low-level control layer consists of drivers for the robot hardware such as wheels, gripper, arm, cameras and distance sensors. Most these drivers are preexisting and ready for use, some drivers had to be implemented such as the platform motor control drivers as a custom made platform was used. The low-level control facilitates the interaction of the individual modules with the hardware.

Just as the individual modules, each low-level control driver uses its own communication method through the available ROS protocols, to receive input or publish its output. No standard communication method has been developed for the drivers since each driver functionality is very different from the other drivers.

Chapter 6

Active Grasping

Working in a changing environment there is a need for action-perception coordination. As the environment changes the robot must act accordingly and be able to reuse its knowledge to interact and learn from novel situations. Interaction with the environment requires that the robot has robust manipulation skills which allow him to handle both known and unknown objects. Therefore in this thesis a novel method for robust object manipulation has been proposed that uses a constant visual feedback to guide the grasping process. Contrary to the state of art methods this method requires no prior knowledge and allows that all the information about the object is learned online through exploration. To achieve such functionality several steps have been proposed. The object is first localized using a 3D segmentation method. Next, the location of the object is used by the visual tracker to constantly update the object position and relearn the object model. Finally, simultaneously with tracking, the closed loop grasping sequence is initiated which decreases the relative distance between the robot and the object until the object is in the gripper. Detailed explanation of the introduced method is given in the rest of this section.

6.1 Object localization, selection and exploration

The first step prior to object manipulation is to locate this object in the environment. Assuming the object is in the field of view of the robot, the target object can be localized using an object detection and selection module.

To differentiate background from the object in the scene 3D data from the Microsoft Kinect has been used. Using 3D spatial information has high advantages over using a 2D image since the objects can be more easily segmented in the case of a cluttered background which is often the case in a household environment. Using the 3D point-cloud data planar regions can be identified and removed from the scene, since they are generally used as support surfaces for objects. Once the major planar regions have been eliminated the remaining of the point-cloud has to be clustered into different objects which is done by the

process of Euclidean Clustering [12].

Once the objects in the scene have been found in 3D the location of the objects in 2D can be obtained as well as the target object to be picked can be selected. Two selection methods have been developed, one for selecting known objects and the other for selecting an unknown object. In the case that the user has specified to the robot which object is required, the recognition module finds the target object among the localized objects. In the case of unknown objects robot has no prior knowledge about the objects or the environment. This situation occurs when robot has to perform clean up task in different environments or when it needs to learn to manipulate novel objects online. In such a case objects can be selected either according to an optimized picking order in case of clean up or using a 2D attention module in the case of object learning. More details on object selection and recognition used in the proposed framework can be found in [25]. Finally, when the target object has been localized, the initial model of the object is created and used to initialize the tracker.

Once the initial position of the object is known, the robot should be able to navigate around it to inspect it from multiple viewpoints, therefore very fast and robust object tracking method must be applied. In order to track the object despite the challenges of cluttered environment and changing illumination conditions, the initial object model needs to be constantly updated and relearned. To solve these problems an advance tracking method OpenTLD (Tracking, Learning, Detecting)[21] has been used. The idea of this method is to keep learning the object while it is being tracked and in that way improve the tracking performance. The learning of the target object is done by constantly adjusting the detection classifier with positive patches from the target object and negative patches from the background [17]. In initialization step both the detector and the tracker are initialized simultaneously. As long as the tracker is confident enough that it is still tracking the correct object it will communicate the position to the detector. Simultaneously the detector tries to detect the object in the same frame, in the case the detector fails or detects something different from what the tracker gave as a target location, the detector is updated. Every iteration the algorithm updates the set of positive and negative patches of the target and the background, so that the detector is always up to date with the current appearance of the target. In the case that the tracker looses the object the detector takes over to find the target in the frame and reinitialize the tracker when the target is found. This method is specifically designed to allow tracking of the object in the case that the object is highly occluded as well as fast regain of tracking in the case the object is temporary lost.

After the object has been located in the scene and the tracker is continuously refining the object model and updates its position relative to the robot. With the tracking being robust and operating in real-time it can be employed as a feedback for grasping.

6.2 Grasping process

The novelty of the grasping method lies in achieving reliable manipulation with a very simplistic yet robust algorithm. Although this is a generic concept applicable to any robot, it has been further explored by implementation on the simplistic robot Robby. The controller of this method has been implemented as a state-machine which is segregated into a approach phase and a grasp phase as depicted in Figure 6.1.

6.3 Approach and grasp phase

The approach phase consists of manoeuvring the end-effector to a position where the tracked dominant object is close to the gripper, such that a simple linear motion is enough to grasp the object. This can be modelled as a control problem in 3D. The problem is simplified since the location of the object with respect to the end-effector is known. If $[XobjYobjZobj]$ is the current location of the object in the gripper frame of reference and $[XreqYreqZreq]$ is the location of the object in the same frame of reference, then a simple proportional control law can be formulated as in equation 6.3.

$$V_x = -K_x(X_{obj} - X_{req}) \tag{6.1}$$

$$V_y = -K_y(Y_{obj} - Y_{req}) \tag{6.2}$$

$$V_z = -K_z(Z_{obj} - Z_{req}) \tag{6.3}$$

For obtaining the object position in the gripper frame, we perform two transformations: From the frame of the Microsoft Kinect to the neck and from the neck to the gripper. Once these transforms are available, the approach takes place in two stages: Planar(xy) error correction and depth (z) error correction.

6.3.1 Planar Error Correction

The desired positions as in (6.3) are set as $X_{req} = 0$ and $Y_{req} = 0$. Firstly any large error in X, Y directions are controlled and then the depth error is corrected. Since the frequency of the visual feedback from the tracking modules is not always consistent, a constant control frequency of 10 Hz is used to control the actuators. While the V_x controls the angular motion of the base, the V_y varies the height of the arm by controlling the revolute joint in the hip. We can then set the gains to $K_x = 1.3$ and $K_y = 0.8$ based on response of the robot in different circumstances. If these gains are higher, the robot moves very fast and it can lose track of the object leading to a grasping failure. In case the gains are very low, the robot responds very slowly to the object and in circumstances where the object is externally moved, the robot can lose the object from its visual field again. Hence the above specified parameters were found to be optimal by testing on different objects and scenarios.

It was also observed from the implementation of the controller that in practice that there exists oscillation around the desired set point. This is due to

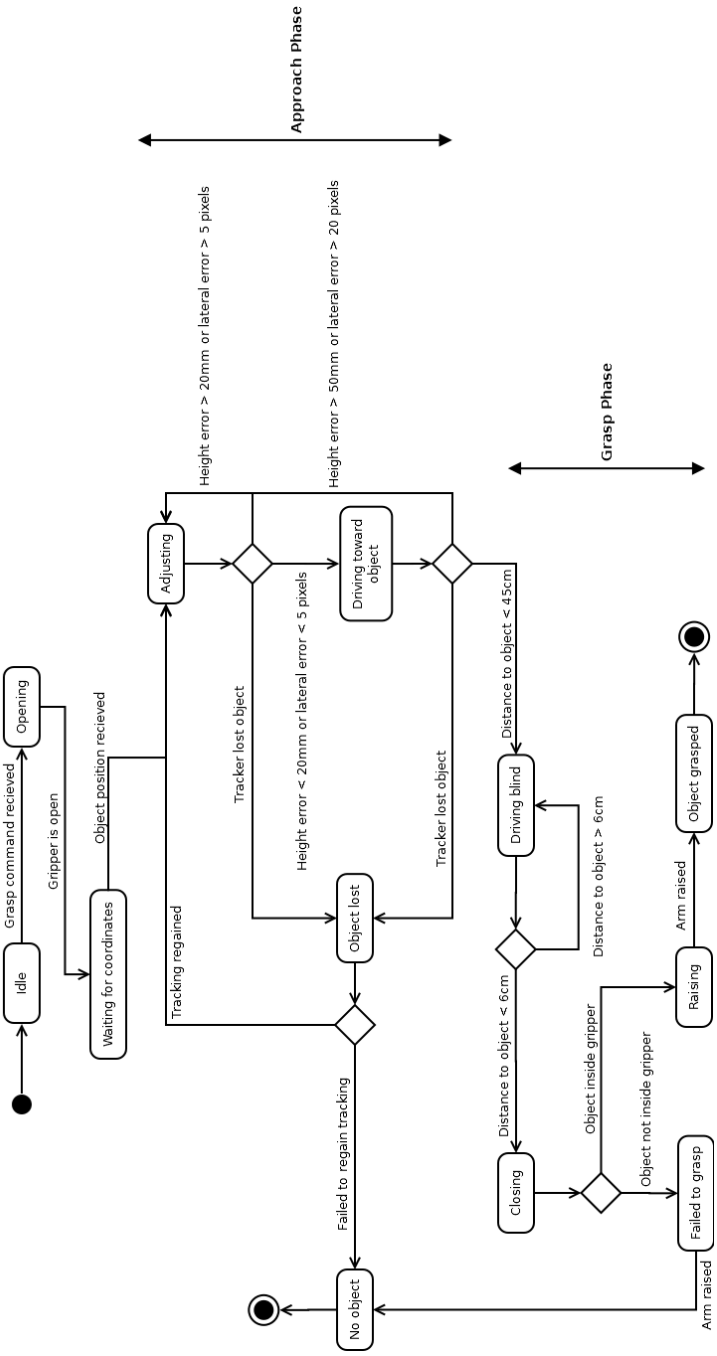


Figure 6.1: A state machine showing the different states of the grasping algorithm.

the presence of only a proportional controller. While an additional derivative controller could be used to decrease this oscillation, it would lead to a longer settling time which is not very much desired for a real-time operation. This can also be easily affected by noise in the visual feedback. Additionally a second order controller can lead to instability in combination with a PID controller already present in the low-level hardware controller of the motors. Hence a simpler and effective solution is using a "region of desired position" instead of a final desired position. Hence (6.3) is modified to (6.5) where the parameters $X_{thr} = 5mm$ and $Y_{thr} = 20mm$ are used.

$$V_x = \begin{cases} -K_x(X_{obj} - X_{req}) & \text{if } |X_{obj} - X_{req}| > X_{thr} \\ 0 & \text{if } |X_{obj} - X_{req}| \leq X_{thr} \end{cases} \tag{6.4}$$

$$V_y = \begin{cases} -K_y(Y_{obj} - Y_{req}) & \text{if } |Y_{obj} - Y_{req}| > Y_{thr} \\ 0 & \text{if } |Y_{obj} - Y_{req}| \leq Y_{thr} \end{cases} \tag{6.5}$$

The horizontal alignment of the gripper with the object has to be more precise than the vertical alignment. Even if the gripper is slightly misaligned horizontally, the object will be off-center within the gripper and there is a greater probability of final gripping failure. This is reflected in the tolerance in X_{thr}. The tolerance in Y_{thr} can be higher as the three gripper phalanges allow successful grasping even slightly above or below the centroid of the object. An ellipse centred at X_{req}, Y_{req} and its principle axes dictated by the threshold parameters is the tolerance zone. The system is actuated until the positional error trajectory enters this zone.

6.3.2 Depth error correction

The $x - y$ correction ensures that any large planar error is within the threshold range and ensures the robot gripper is co-planar with the object plane. In ideal situations the robot has to move straight forward to grasp the object. Instead of blindly moving forward until the object is grasped, additional intelligence has been incorporated. This takes place in two states: driving forward with visual feedback and driving forward with wheel odometry. The lower limit of the depth map from the Kinect is limited to 45 cm [28]. The 45 cm region of depth in front of the Kinect is termed as its blind zone. Due to this, the robot initially drives forward decreasing the distance error based on the control law in the z direction:

$$V_z = -K_z(Z_{obj} - Z_{req}) \tag{6.6}$$

until it reaches the blind zone. A value of $K_z = 1$ was found to be adequate for the direct approach of the object. Once the robot's presence in the blind zone is detected in the tracking feedback, the robot moves forward relying solely on its proprioception. Proprioception is the knowledge of the robots current configuration based on the internal sensory information. For example, the position of the end effector can be predicted if the joint angles are known. This can be seen as analogous to the way humans can reach small distances with their arms in the dark. In this case, the distance moved by the robot is approximated by

using the velocity commands issued to the mobile base.
The last known distance from the object before the blind zone ($Dist_{blind}$) is the distance to be moved before the robot is just in front of the object. The object is grasped when the proximity sensor detects the presence of an object at a distance less than $Dist_{grasp}$ from the gripper. This value was determined to be 6 cm. A higher value causes the gripper to be closed, even when the object is not within it. When a lower value is used, the robot may topple the object. Hence, $6cm$ was found to be optimal for a successful grasp for almost all everyday objects. In order to avoid any stray sensor noise influencing the grasping, an additional check of the wheel odometry is used. Since the velocity commands to the base are known, the distance$[S_{forward}(t)]$ moved forward at time T is determined by:

$$S_{forward}(T) = \int_0^T V_z(t)\,\mathrm{d}t \tag{6.7}$$

Based on this, the robot can grasp the objects based on the proximity sensor information only if the distance moved $S_{forward}(t)$ is within threshold distance (Thr_{odom}) from $Dist_{blind}$. So the final grasp based on the sensor is activated by the state $Grasp_{state}$ which is governed by:

$$Grasp_{state} = \begin{cases} True & \text{if } |S_{forward} - Dist_{blind}| \leq Thr_{odom} \\ False & \text{otherwise} \end{cases} \tag{6.8}$$

Thr_{odom} is used instead of $Dist_{blind}$ to compensate for the inaccuracy in Kinect depth and the wheel odometry information. A value of 5cm for Thr_{odom} was used to obtain reliable grasping in different indoor conditions. Once these conditions are satisfied and the object is grasped, the robot returns to its default neck and hip positions and the object manipulation module returns a success response. As the overall object grasping process is closed loop, the technique can also be used to grasp dynamically moved objects.

6.4 Recovery Mechanisms

The continuous tracking and segmentation for grasping unknown objects as described above assumes that the object position is continuously available from the tracker module. The object grasping process can fail in situations where the tracker module temporarily or permanently loses the location of the object. In order to avoid undesirable behaviour from the robot, certain measures have been taken to recover from these situation. The recovery action is two fold: recovery in the approach phase and recovery in the grasp phase.

The recovery in the approach phase works purely on basis of visual feedback. There are instances where the tracker temporarily loses the object due to sudden viewpoint change. In this situation the robot attempts to regain the tracking by retracing to the last position where the object was still tracked. In most of the situations the tracker regains the object and the robot controller functions

normally again. In case when the object location cannot be regained the robot goes back to the initial configuration at which the grasping action was activated. The robot tries to locate the object to be grasped again and the tracker starts by learning the features afresh.

When the approach was successful and the robot starts a depth correction phase, an infra-red proximity sensor detects the presence of an object, the gripper closes and the grasping is accomplished. When the object is not detected by the proximity sensor even while the robot is in the vicinity of the object, the only possible reason is that the object is not at that location any more. Hence for the recovery in the grasp phase the robot goes back to the original pose in which it located the object and the whole process is repeated again. Repeating the process is restricted to a maximum of three attempts. If the object grasping fails three times the robot stops trying and assumes the object is not graspable.

In the next section detailed experimental setup and results are presented which show that this grasping technique is very generic and simplistic. It can be reliable in many scenarios like simple objects, complex objects, different locations and also externally movable objects. These features of the grasping technique make the grasping process smooth, continuous and a naturally interactive process also leading to better human-robot interaction.

Chapter 7

Experimental Setup and Results

In this section, a detailed experimental setup and experiments performed to test the performance of the software architecture and active grasping method are given. The software architecture experiments do not focus on the performance of individual modules but on the performance of the overall complex task and the fail-safe mechanisms in case of a module failure. The experiments of the grasping method focus mostly on the grasping performance and not on the performance of the object localization and tracking method used. Robot Robby was used for all the experiments in this thesis.

7.1 Software Architecture Testing and Application

The proposed software architecture has been tested in a variety of different service robot task that are specified in the RoboCup@Home rulebook [5]. RoboCup@Home is a competition were teams from around the world come with their service robot. The idea behind this competition is to stimulate research of household tasks and how these can be performed by a robot. An example of a task at RoboCup@Home is "Who is Who" as shown in Appendix C The "Doctor Demo" is another task which was performed at the RoboCup@Home competition as the demo challenge. Both these tasks and many other complex tasks have been tested extensively in different environments, such as a testing arena in Delft, the Dutch Open arena and the RoboCup 2012 @Home arena.

7.1.1 Doctor Demo

The doctor demo was created as a demo challenge for the RoboCup@Home competition and shows application of robot as a secretary in medical settings.

In this demo the robot needs to perform a series of tasks which are required to receive patients in medical center. Scenario is as follows.

First the robot will enter the room and wait for the doctor to appear. Once the robot has recognized that the doctor has entered, it will great the doctor and ask if it can get him a coffee. Next, a patient enters the room and greats the robot. The robot must then learn this patient and guide him to the waiting area. The robot informs the doctor that the patient is waiting and the doctor asks the robot to inform the patient that he may enter. After the robot informs the patient that the doctor is ready, the robot goes to get the coffee for the doctor. When the robot has the coffee, it brings it to the doctor and finally leaves the area.

In this demo multiple modules were used to complete the complex task. These modules include:

- Door Opening Detection
- Face Recognition
- Face Learning
- Speech Recognition
- Speech Synthesis
- Navigation
- Grasping
- Delivering

7.1.2 Who is Who

In the who is who task the same modules as those in the doctor demo have been used to accomplish a totally different complex task. In the who is who the robot enters the arena by detecting the door opening (Door Opening Detection). Next it has to learn three people (Face Learning, Speech Recognition and Speech Synthesis). It must then safely navigate to the living room and ask for a order (Navigation, Speech Recognition and Speech Synthesis). Finally, it manipulates the ordered objects and delivers them (Navigation, Grasping and Delivering). Using the same modules in a different sequence allows for the task to be easily executed. Adjusting of the modules was not necessary to complete the task.

7.1.3 MoSCoW Requirements Analysis

In table 7.1 the requirements of the MoSCoW model have been rated on a +/- scale:

++ : More than required has been done.

\+ : The requirement has been fulfilled very well.

Must Have		+/-
1.1	Must be modular.	0
1.2	Must be able to combine different modules to accomplish a complex task.	+
1.3	Must be applicable on robot Robby.	++
1.4	Must be usable to accomplish all RoboCup@Home tasks.	+
Should Have		
2.1	System modules should be reusable such that they can be part of different complex tasks.	+
2.2	System architecture should be usable on different robotic platforms with a different body morphology.	0
2.3	Should have a fail-safe mechanism, in the case of a module failure.	-
Could Have		
3.1	Use modules in a computational and energy efficient way.	-
3.2	Adjust the action based on the information given.	-
3.3	Optimal division of tasks and sub tasks, such that there are no two task having similar functionalities.	-
Won't Have		
4.1	System architecture will be able to decide automatically which modules are required to complete a complex task.	- -
4.2	Will be able to automatically re-plan its actions by using other available modules, in the case a module fails.	- -

Table 7.1: Software Architecture Requirements Analysis

0 : The requirement has been fulfilled

- : There has been work done on this requirement, more future work may be required.

- - : The requirement has not been fulfilled.

The testing and implementation of different complex tasks such as the Doctor Demo and Who is Who task were used as a criteria to analyse each of the requirements.

All the must have requirements have been fulfilled. Requirement 1.1 has a lowest rating since the system could have been designed more modular by dividing the tasks in more optimal way. For example the find calling person

module, consisting of looking around and finding a person waving, could have been divided into a searching module and a gesture detection module such that both these new modules could have been used for different tasks such as searching for objects or perceiving gestures from a user when the user is already in front of the robot. The architecture showed good performance on the given platform (robot Robby) and was used to complete all the tasks required for the RoboCup@Home competition.

Most of the should have requirements have been implemented. Modules were often reused for different complex tasks and the software architecture will function on a different robot platform with the correct drivers and control for this robot. Requirement 2.3 was implemented in some of the modules but not all the modules created contained a fail-safe mechanism. For the grasping module a fail-safe mechanism was implemented in the case the grasping procedure failed the robot could correctly communicate this with the user directly or move on to the next task. Such fail-safe mechanism are important to keep the actions of the robot efficient and effective. Another example of a fail-safe mechanism was implemented in the learning a new person subcore. In the case the robot does not understand the name correctly, it will assign a random name to avoid the situation of continuously asking the name of the person.

The could have requirements were all implemented but to a very minimal level. Requirement 3.1 was fulfilled in some modules as they were optimized to be computational efficient. Requirement 3.2 was partially implemented as the robot changes the action in the who is who task based on whether it actually got the order or not and apologizes to the user in the case it could not get the drink.

None of the won't have requirements were implemented as these requirements are considered to be developed in future work. Making the system more intelligent by including a adaptive decision making module could have great advantages to robots in the household. Such a decision making module should decide online which modules are best used to complete a certain complex task or to re-plan the actions in case of a module failure.

7.2 Grasping experiments

To test the grasping performance, a series of grasping sequences have been performed. The precision rate is obtained by counting the number of successful grasps performed by the robot. In total 10 different objects were used that varied in size and shape complexity as can be seen in Figure 7.1. Complex objects had different properties, anamorphic shape, slippery surface, shiny texture, etc. This variation in object properties was required to test the efficiency of the grasp phase. Grasping heights were varied and grasping was performed at three different locations the floor, table top and fridge in order to test effectiveness of the approach phase. For every object and each location four grasping attempts where performed, which in total for all experiments counts to 120 grasps with the robot. For all experiments performed, the robot had no prior information

on objects or the environment.

Figure 7.1: *Objects used in grasping experiments: from left to right*: simple objects, complex objects

7.2.1 Experiment 1: Grasping precision in case of grasping objects from a table top

An example of the robot grasping objects from the table top is shown in Figure 7.7. This grasping method is generally very reliable and has been successfully demonstrated in Robocup@Home [8]. In order to show the benefits and limitations of such an approach, the following scenarios were used to examine the influence of different object and environmental conditions on the grasping method.

1. simple vs complex objects to test the efficiency of gripper
2. uniform vs textured background to test the influence of tracker
3. single vs multiple object scenario to test the influence of clutter

	simple objects	complex objects
Single object scenario	100%	100%
Multiple object scenario	85%	70%

Table 7.2: Precision in the case of grasping object from table top with a simple background

	simple objects	complex objects
Single object scenario	100%	95%
Multiple object scenario	80%	65%

Table 7.3: Precision in the case of grasping object from table top with a cluttered background

In Table 7.2 and 7.3 precision results for all scenarios are shown. It can be seen that in case of single object present on the table active grasping is a very

efficient method, with a success rate of above 95 %. Also, complexity of object shape or uniformity of the background have no influence on the grasping process. In case of cluttered background with complex objects, the precision drops by 10% however it is still around 90%. This decrease is due to either objects with slippery surfaces which dropped from the gripper or transparent and non-reflective objects which are difficult to track. In the case of multiple objects, 4-6 objects with similar color and texture to the object being grasped are present and objects are placed close together as seen in Figure 7.2. The grasp precision decreases further by 10 % which can be attributed to the tracker failure due to the effects of viewpoint and scale change combined with the similar appearances around the object. Figures 7.3 and 7.4 show convergence of the planar error, based on the velocities generated by the grasping algorithm in cases of a successful grasp. It can be observed that though the error converges around seven seconds irrespective of the properties of the object or the background, there is more oscillation in presence of multiple objects in a cluttered background. This can also be associated with the tracker's performance to locate the centroid of the refined contour while the object is being grasped.

Figure 7.2: Robot grasping object which is touching another object.

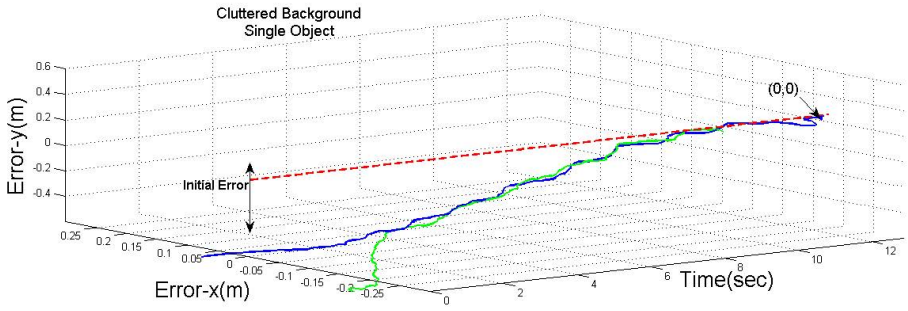


Figure 7.3: Error convergence in a case of single object on a cluttered background. Blue represents simple, while green represents complex objects

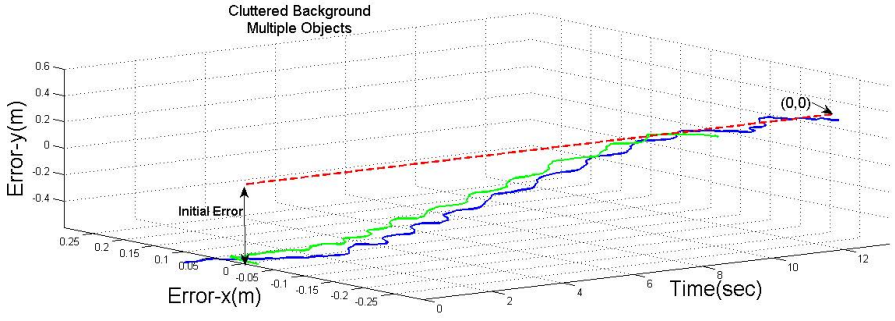


Figure 7.4: Error convergence in a case of multiple objects on a cluttered background. Blue represents simple, while green represents complex objects

	Table Top	Fridge	Floor
Simple Object	100%	80%	75%
Complex Object	95%	80%	70%

Table 7.4: Precision of grasping simple and complex Objects from different locations.

7.2.2 Experiment 2: Grasping precision in case of different heights

To test the effectiveness of the approach phase, objects are placed at different heights. In this experiment only single object scenario were observed, performing grasps from the table top, fridge and floor as is displayed in Figures 7.7, 7.8 and 7.9. In Figure 7.4 influence of the object height on precision is shown. There is a large drop in precision in the case of grasping from locations other then table top, up to 20% when grasping from the fridge. Between the fridge and the floor there is only small difference in precision of 5%. The complexity of objects did not have much influence in this case. Grasping from a fridge door had many challenges due to motion of the door during the grasping process. Also slippery objects gave problem while lifting objects from the fridge door. When grasping from floor, the main challenge was in updating the tracking model, due to the great distance between the object and the camera. This gave a lot of problems in case of shiny objects. Similar result can be seen in Figure 7.5 where the error convergence in case of multiple heights is shown. Due to the tracking challenges in the case of grasping from floor it takes much longer for the robot to reach the grasping position compared to other cases.

7.2.3 Experiment 3: Interactive grasping

The final experiment that was performed was interactive grasping. This experiment was performed to show that using the introduced active grasping method it is possible to perform grasping of object that is constantly moved by the human

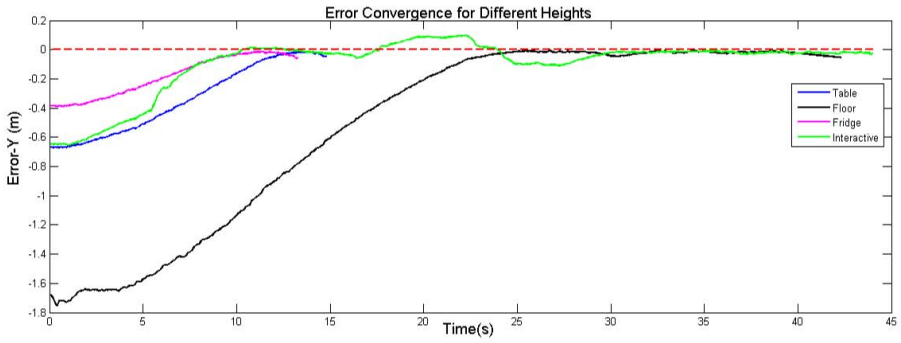


Figure 7.5: Error convergence in a case of grasping single object from multiple heights

user, as is shown in Figure 7.10. The robot managed to successfully perform such grasps and error convergence for both complex and simple object is shown in Figure 7.6. Such approach is very handy for human robot collaboration.

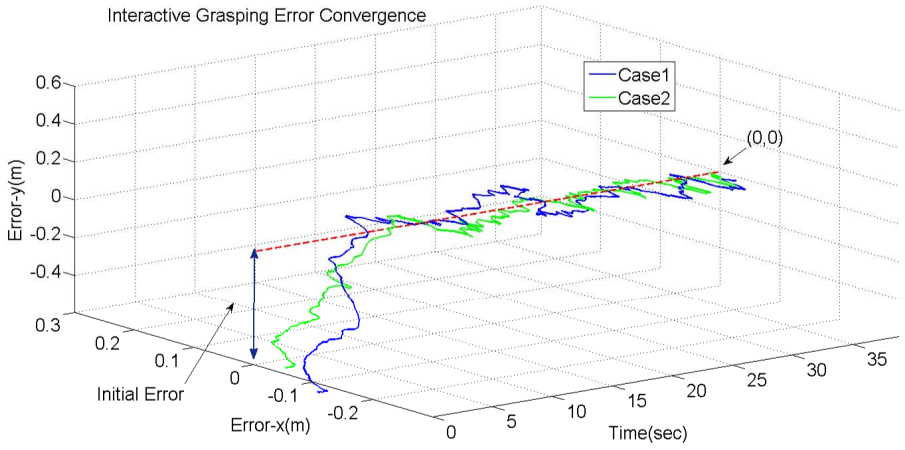


Figure 7.6: Error convergence in a case of interactive grasping. Blue represents simple, while green represents complex objects

7.2.4 MoSCoW Requirements Analysis

In table 7.5 the requirements of the MoSCoW model have been rated using the same scale as used in table 7.1. Together with a rating, each requirement has been linked with one or more experiments performed.

All the must have requirements have been fulfilled since without these re-

Must Have		+/-	Reference
1.1	Robot must locate atleast one object in the scene and find the position of this object in relation to the robot.	++	Exp. 1-3
1.2	Robot must track the object and constantly update the relative position, to assist the grasping process. While the arm is moving, the object should be static.	+	Exp 1-2, Figure 7.3
1.3	If the tracker looses the object it must communicate this correctly to the robot core and the grasping procedure must be stopped.	+	Section 6.4
1.4	Robot must move its arm till the gripper is within proximity of the object and pick up this object.	+	Exp. 1-3
1.5	Robot must be able to grasp from a table top.	++	Exp 1, Table 7.2 and 7.3
1.6	Robot must be able to grasp objects with a simple shape (boxes, cans, etc.).	+	Exp. 1, Table 7.2 and 7.3
Should Have			
2.1	Robot should be able to grasp an unknown object.	+	Exp. 1-3
2.2	Robot should localize multiple objects in the scene and select which object to grasp.	+	Exp. 1-2, Figure 7.4
2.3	If the tracker has temporarily lost the object position, the robot should continue grasping when track of object is regained.	+	Section 6.4
2.4	Robot should correctly grasp a dynamically moving object.	0	Exp. 3, Figure 7.6
2.5	Robot should correctly handle the case that the desired object could not be grasped by communicating this to the core of the robot.	-	Section 6.4
2.6	Robot should be able to grasp objects from different heights.	+	Exp. 2
2.7	Robot should be able to grasp objects with a complex shape.	+	Exp. 1-2
Could Have			
3.1	Be able to grasp the object even if the object is touching an other object.	-	Figure 7.2
3.2	Grasp an object of the same or similar category if the user requested item is not available.	- -	
3.3	Update and learn the object appearance model while manipulating the object.	0	Section 6.1
3.4	Detect whether an object is graspable before executing the grasping algorithm.	- -	
3.5	Grasp objects from constrained surfaces such as an open fridge or a bookshelf.	0	Exp. 2, Figure 7.8
3.6	Avoid all obstacles while executing the grasping algorithm.	- -	
Won't Have			
4.1	Robot will be able to grasp objects that have a vertical grasp orientation.	- -	
4.2	Detect a cabinet door or fridge door and open it before finding and grasping the desired object.	- -	
4.3	Move objects that are obstacles to clear a path to the desired object to be grasped.	- -	

Table 7.5: Active Grasping Requirements Analysis

Figure 7.7: Example of robot grasping object from table top in multiple objects case with cluttered background

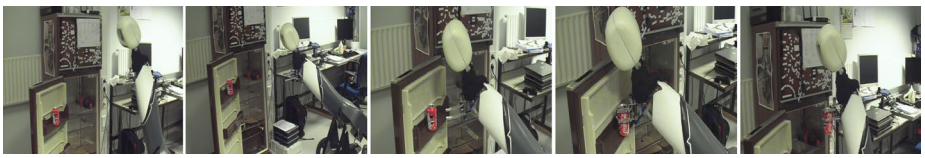


Figure 7.8: Example of robot grasping object from fridge

quirements the robot would not be able to perform the grasping task at all. Requirement 1.1 is rated as "++" since the robot is capable of localizing an object among multiple objects. 1.5 is also rated "++" as the experiments show that the robot is capable of grasping from many different heights.

Most of the should have requirements were fulfilled very well. The experiments show that the robot can robustly grasp various objects from multiple heights and in different environments, this meets most of the requirements described in as should have requirements. Requirement 2.4 has been given a "-" as it did not handle the failure of grasping correctly in all cases.

Requirement 3.2, 3.4 and 3.6 of the could have requirements have not been implemented as these require more research and did not directly effect the functionality of the method. Avoidance of obstacles while grasping is a requirement which can be considered as future work with a high priority as it influences the safety of the system. Requirement 3.1 was implemented and made functional but failed in many cases, and thus requires future work to function correctly. Requirement 3.3 and 3.5 were both implemented and functional but could be improved to achieve additional robustness.

None of the won't have requirements were implemented as these are considered requirements which can be used for future work as extensions of the existing method. Future work includes opening of a fridge or cabinet door to

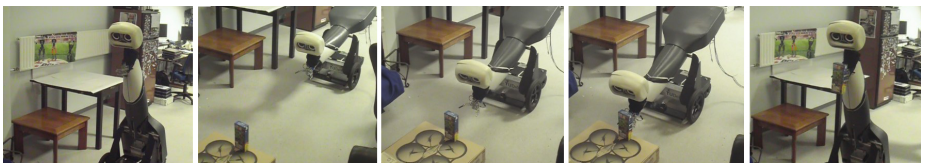


Figure 7.9: Example of robot grasping object from floor

Figure 7.10: Example of robot grasping an object moved by an operator

reach for an object, although this method should be used as an individual module as it is useful for other tasks as well. The method may also be extended for use on a more complex robot capable of grasping objects with a vertical grasp orientation. This will allow handling of small objects and grasping from cupboards often present in households.

Chapter 8

Conclusion

In this thesis methods which shall bring robots closer to their application in the household environment have been proposed. They deal with all the challenges of working in dynamic and novel environments such as reusing previous knowledge to solve new tasks or manipulation of unknown objects. In this thesis a modular software architecture capable of solving complex tasks has been proposed. This architecture allows for reusability of individual modules within a robot and can be applied on different robot platforms. The main benefit is that it enables for updates or additions of functionalities of the robot as well as integrating them with the existing modules to solve complex tasks. It also allows for a fail-safe mechanism in case of a module failure or crash which makes entire system reliable.

One of the most important functionalities of robots performing complex household tasks is object manipulation. Therefore the second method that has been proposed is a novel active grasping method, capable of manipulating different objects from various heights without any prior knowledge. This method depends on an input from the object localization and tracking modules responsible for providing the relative position of the object. A closed loop grasping process has been proposed which uses the input from the tracking module to constantly update the position of the object as the robot approaches it. Constant feedback from the tracker makes grasping of dynamically moving objects possible. After the robot is in close proximity with the object a proximity sensor is used to detect the presence of the object in the robot gripper.

Both software architecture and object manipulation have been applied on a robot platform with a mobile base, a 1-DOF arm and an underactuated gripper which adapts to the shape of an unknown object. The entire system has been tested extensively in a variety of household environments. Several examples of the robot successfully performing complex household tasks have been shown and was used for Rocup@Home competition that deals with challenges of bringing service robots to real world. Furthermore to prove the effectiveness of the introduced active grasping method, extensive experiments have been performed. Experiments showed that by using this method the robot can efficiently and in

real time grasp objects which vary both in appearance and shape complexity, independently from clutter in the scene or the presence of multiple objects. It was also shown that it is possible to grasp objects from various heights as well as to perform interactive grasping when the object is constantly moved by a user. Finally, using experimental results the MoSCoW requirements have been analysed to show the benefits and disadvantages of the complete systems as well as possible future extensions.

For future work a decision making module could be implemented, that allows the system to decide online which modules may be required to complete a complex task. Such a module could also be capable of re-planning a task in the case one module failed to complete its action. Other future work related to the grasping method is object avoidance and more complex grasps. The grasping method should be extended with an object avoidance module to make it safer and more robust. Grasping method could also be extended to work on different robot platforms, with more complex arms, so that more complex grasps such as vertical oriented grasps can be performed.

Appendix A

Teamwork

Although officially I worked alone on this bachelor project I did not do everything by myself. My main supervisor was Dr. Maja Rudinac and I received much guidance from her, which I am very grateful for. I worked on this bachelor project as part of the RoboCup@Home team consisting of bachelor, master and PhD students. Together with the guidance and help from different team members, I researched on possible approaches for the problems of bringing service robots to households and listed the requirements.

When I started my project there was already a basic architecture available designed by a former student from Haagse Hogeschool. After the orientation phase and many discussions with other team members, I decided to take a different approach to the problem and started with a new design. Three students especially, Aswin Chandarr, Floris Gaisser and Mukunda Bharatheesha, took part in these discussions and helped to formulate the requirements of the software architecture. After I had set up the main structure of the software architecture, Floris also helped with implementation of specific cores and subcores.

Together with Maja and Aswin, I developed the active grasping algorithm. My main contribution to the development of this method lays in the robot control algorithm. The object localization and tracking methods used where developed by Maja and Aswin, as my expertise are lacking in the area of computer vision. Without these methods the grasping would not have been functional. Along with providing their expertise in computer vision they also helped with the development, implementation and testing of the final grasping method. The vision processing required for this method was also tested in same experiments but was not covered in this thesis as I brought no contribution to this part of the grasping method.

Besides receiving guidance and help during the project I also took an active role in the team by being available to help others in the lab. I spent a lot of time together with Mukunda testing and adapting the ROS navigation stack on the robot. Also I was involved in supervision and guidance of new members, especially Haagse Hoogschool students developing robot user interface. Towards

the end of the project I joined the team to Mexico for the Robocup competition and helped in the trip organization.

Finally I would like to acknowledge Dr. Maja Rudinac and Dr. Wouter Caarls for the help and guidance they gave me while writing this thesis. I would also like to thank Dr. Joost Broekens and Dr. Martha Larson for taking part in my thesis defense.

Appendix B

Task Description

Title of project proposal:
Software architecture for affordable personal robots.
Keywords:
Task coordination framework, Testing and validation, Object recognition and motion analysis, Safe human robot interaction.
Introduction:
Bringing personal robots to households requires solving many challenges. The robots need to deal with the unstructured and dynamic environment, need to safely interact with humans and to be very affordable for the large market of consumers. In the BioRobotics lab, we have constituted Robocup@Home team of students with a goal to design the first Dutch affordable service robot. From this year we will also participate in the RoboCup@Home league competition, which is the largest competition of service robots, where they need to perform complex household tasks such as: interaction with humans, bringing and preparing food, cleaning the items from floor, shopping in the mall, etc. The team is consisting of both masters and bachelor students from both 3ME and EWI who are designing and implementing all required algorithms and preparing the robot for the competition.

Service robots must be capable of performing activities such as object recognition and grasping, autonomous navigation and safely interact with their respective environment. Each of these activities requires complex algorithms to be executed on a computer and subsequently communicate with the robot hardware to achieve the desired actions. These algorithms constitute the Brain of the robot, You will have the opportunity to work in the multidisciplinary team and to test your algorithms on the real platforms DPR2 and Tuxedo. And of course to help us bring personal robots to the every household.

Research questions:

- Student 1: Implement a task coordination framework for Tuxedo such that it meets the real-time task demands of a service robot. The framework

has to be generic enough to be portable to different service robots.

- Student 2: Design and implement guidelines for the verification and ratification of the Tuxedo software, based on the existing testing frameworks at both a component and functional level. Student 1 and Student 2 will closely collaborate together.
- Student 3: Design of a real time object recognition module for a mobile robot in dynamic and cluttered environment combining 2D and 3D information available from different sensors.
- Student 4: Design of a motion analysis module for a mobile robot including multi sensory calibration, ego motion estimation and compensation and tracking of rigid objects and people using combination of 2D and 3D sensors.

Student 3 and Student 4 will closely collaborate together. All students: All modules will be applied and tested on the low cost personal robot Robby. Requirements:

1. Programming skills: Knowledge of C++ is required
2. Experience on Robot Operating System (ROS) and Linux is a plus
3. Finished minor in robotics is a plus but not required

Remarks:
Students will need to collaborate together and with the multidisciplinary Robocup@Home team. Results of the bachelor theses will be used for the Robocup@home competition in June this year.
Supervisors:
Dr Maja Rudinac and Dr Wouter Caarls
Biorobotics Lab, 3ME
TU Delft

Appendix C

RoboCup@Home Tasks

C.1 Who Is Who

The robot has to learn and recognize previously unknown persons, and deliver drinks.

C.1.1 Focus

This test focuses on human detection and recognition, manipulation, safe navigation and human- robot interaction with unknown persons.

C.1.2 Task

- Entering: The robot has to enter the arena, and stop in the vicinity of the door.
- Memorizing guests: Three persons enter through the door (one after another) and step in front of the robot. The robot has to introduce itself to each person, memorize it, and ask for its name.
- Changing rooms: The three persons are arranged by the referees in another room, two standing and one sitting. There are also two persons in the room which are not known to the robot, one standing and one sitting. When told to do so by a referee (after the arrangement of the persons), the robot may enter the room.
- Getting called: One of the known standing persons (designated by the referees) tries to get the attention of the robot, by lifting his arm and waving, and by calling it. The robot has to approach and face that person. If the robot fails to approach the person within 2 min or if the team decides to do so, the person approaches the robot and stands in front of it. After the robot has recognized the ordering person and announced its name, it asks for a drink order.

- Taking the order: The designated person orders drinks (designated by the referees and from the set of prede

  ned objects) for all three known persons.

- Getting the drinks: The robot has to navigate to a designated location in another room where drinks are stored. The robot may grasp any number of drinks, e.g., all three drinks ordered, or just one, and return to the room where it received the order.

- Delivering the drinks: The robot has to search for persons, recognize found persons, and hand over the correct drink if there is an order for the recognized person.

- Leaving the arena: After delivering all three drinks, the robot has to leave the arena.

C.1.3 Additional rules and remarks

- Repeating names: The robot may ask to repeat the name if it has not understood it.

- Misunderstood names: If the robot misunderstands the name, the understood (wrong) name is used in the remainder of this test.

- Misunderstood order: If the robot does not understand the order, it can continue with an own assignment of drinks to persons or with a wrong, misunderstood assignment.

- Changing places: After giving the order (when the robot is not in the room), the referees may re-arrange all persons including their body posture. That is, a sitting person may change to a standing posture and vice versa.

- Positions and orientations: All persons roughly stay where they are, but they are allowed to move in certain limits (e.g. turn around, make a step aside). They do not need to look at the robot, but are requested to do so, when instructed by the robot.

- Asking for help: If the robot fails to grasp a drink (or if the teams decides to do so), the robot may ask for help and that a referee hands over the object (loosing points for grasping). The robot has to clearly indicate that it has recognized the correct drink, e.g., by facing the drink, naming it and telling its rough position (e.g., leftmost, rightmost etc.) relative to the other drinks on the table.

- Correct delivery: The drinks do not have to be handed over to the user. Putting them on the ground or asking the user to grab them from some kind of tray is allowed. When taking a drink from the robot, a sitting person may stand up in order to get it. However, in case the robot is

carrying more than one object at a time, a delivery is only considered successful when there is an easily comprehensible mapping from grasped objects to recognized persons. When putting all three drinks on a tray, for example, the robot has to name the correct drink and indicate its rough position relative to the others.

- Empty arena: During the test, only the robot, the three learned and the two unknown persons are in the arena. The door opener, the referees and other personnel will be outside the scenario (or be the unknown persons).
- Calling instruction: The team needs to specify before the test which ways of getting the attention of the robot are allowed. This can be waving, calling or both of them. It can also decide to skip this part.
- Announcement of locations: Both the locations of the drinks and the rooms where the test takes place are announced beforehand. Note that there may be more objects at the drink location than the three ordered drinks.

C.1.4 Referee Instructions

The referees need to

- select the people and their names from the list of person names (see Section 3.2.8),
- arrange (and re-arrange) persons in the room,
- select the ordering person and the order to give (write down the understood names and update the order accordingly).

C.1.5 Organizing Committee Instructions

2h before test:

- Specify and announce the location where the drinks are placed.
- Specify and announce the rooms where the test takes place.

C.2 Demo Challange

During the Demo Challenge teams are encouraged to demonstrate recent research results and the best of the robots' abilities. In contrast to the open challenge, it is not a completely open but scoped demonstration. Teams are encouraged to pick up problems within the scope of the challenge, and to demonstrate new abilities and applications. The scope of the demo challenge changes every year.

C.2.1 This year's focus

The scope of this year's demo challenge is health care. That is, the demonstration should focus on people who require care, such as senior citizens and children. Whatever is demonstrated should be within the scope and new, i.e., nothing that has already been demonstrated in the previous tests or in previous competitions. Possible tasks include, but are not limited to:

- eating aid
- rehabilitation aid
- mobility aid
- hair-washing aid
- setting up and/or cleaning up a table
- making the bed
- catering drinks or delivering medicines
- carrying a person
- other elderly-care
- playing with children or baby-sitting
- conversational partner robot

C.2.2 Task

The Demo Challenge is an open demonstration which means that the teams may demonstrate anything they like (within the scope of the focus). The performance of the teams is evaluated by a jury consisting of all members of the technical committee. The procedure for the challenge and the timing of slots is as follows:

- Setup and demonstration: The team has a maximum of ten minutes for setup, presentation and demonstration.
- Interview and cleanup: After the demonstration, there is another thee minutes where the team answers questions by the jury members. During the interview time, the team has to undo its changes to the environment.

C.2.3 Presentation

- Elevator pitch: At the beginning of the demonstration, the team has to briefly (maximum one minute) describe the addressed problem, what the robot is about to do, and the importance of the task with respect to the problem and the scope of the challenge.
- No presentation: The rest of the demonstration should not feature a presentation, and on its own should make sense to the audience and the jury.

C.2.4 Changes to the environment

- Making changes: As in the other open demonstrations, teams are allowed to make modifications to the arena as they like, but under the condition that they are reversible.
- Undoing changes: In the interview and cleanup team, changes need to be made undone by the team. The team has to leave the arena in the very same condition they entered it.

C.2.5 Jury evaluation

- Jury: The jury is constituted of members of the technical committee.
- Evaluation: Both the demonstration of the robot(s), and the elevator pitch are evaluated. The jury can give a maximum of 1500 points for factors such as
  - complexity of the task and performance,
  - marketability/story,
  - safety,
  - human-robot-interaction, and
  - usability/appearance.

The actual scoring is not just normalized over all jury members, but discussed within the technical committee after the Demo Challenge.

C.2.6 Additional rules and remarks

- Abort on request: At any time during the demonstration, the jury may interrupt and abort the demonstration
  - if nothing is shown: in case of longer delays (more than one minute), e.g., when the robot does not start or when it got stuck;
  - if nothing new is shown: the demonstrated abilities were already shown in previous tests (to avoid dull demonstrations and push teams to present novel ideas).
- Team-team-interaction: An extra bonus of up to 500 points can be earned if robots from two teams (4 robots maximum, 2 from each team) successfully collaborate (robot- robot interaction).
  - This bonus is earned for both teams.
  - The robot(s) of the other team must only play a minor role in the total demonstration.
  - It must be made clear that the demonstrations from the two teams are not similar, otherwise the points cannot be awarded.

– In case a team receives two (or more) bonuses, the maximum bonus will be taken.
– The collaboration is possible even if one of the two teams has not reached Stage 2.
– The team which does not participate in Stage 2 receives no points for this test.

Appendix D

Software Improvement Group

This appendix contains details of the evaluation from the Software Improvement Group(SIG). SIG evaluates code implementation base on the maintainability of the code. For the evaluation the implemented code had to be sent to SIG. For the first evaluation the code of the grasping algorithm was evaluated by SIG and scored 3 out of 5 stars. The following is the response to the evaluation of the code:

"[Aanbevelingen] De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size en Unit Complexity.

Wat als eerste opvalt binnen dit systeem is het kleine volume, slechts 410 regels code in 2 files. Door dit volume concentreren de aanbevelingen zich op een enkele methode, simpelweg omdat in deze methode ruim 30

Voor Unit Size meten wij het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langste methode in dit systeem, de 'timerCallback'-methode, zijn aparte stukken functionaliteit te vinden binnen elk case-statement, deze kunnen ge-refactored worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '// Close the gripper and check if there is an object in the hand' en '// stop both the base and arm from moving and stop the timer' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar deze lange methode binnen dit systeem en deze indien mogelijk op te splitsen.

Voor Unit Complexity meten wij het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van de bovenstaande methode in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit

geval komen de meest complexe methode ook naar voren als de langste methode, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

Over het algemeen scoort de code gemiddeld, hopelijk lukt het om dit niveau te behouden of te verbeteren tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen."

Bibliography

[1] [Online]. Available: http://pal-robotics.com/

[2] [Online]. Available: http://www.robocupathome.org/

[3] [Online]. Available: http://en.wikipedia.org/wiki/MoSCoW_Method

[4] [Online]. Available: http://www.ros.org/wiki/

[5] [Online]. Available: http://www.ais.uni-bonn.de/~holz/2012_rulebook.pdf

[6] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, vol. 17, pp. 315–337, 1998.

[7] T. Asfour, K. Regenstein, P. Azad, J. Schrder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann, "Armar-iii: An integrated humanoid platform for sensory-motor control." in *Humanoids*. IEEE, 2006, pp. 169–175.

[8] M. Bharatheesha, M. Rudinac, A. A. B. Chandarr, F. Gaisser, M. Bruinink, S. P. Rueda, B. Kupers, S. Driessen, X. Wang, M. Wisse, and P. Jonker, "Delft robotics robocup@home 2012 team description paper," in *RoboCup 2012 Mexico*, 2012.

[9] J. Bohg, M. Johnson-Roberson, B. Leon, J. Felip, X. Gratal, N. Bergstrom, D. Kragic, A. Morales, and B, "Mind the gap-robotic grasping under incomplete observation," pp. 686–693, May 2011.

[10] J. Bohren, R. B. Rusu, E. G. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Msenlechner, W. Meeussen, and S. Holzer, "Towards autonomous robotic butlers: Lessons learned with the pr2," in *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*. IEEE, 2011, pp. 5568–5575.

[11] R. A. Brooks, "A robust layered control system for a mobile robot," in *Autonomous Mobile Robots: Control, Planning, and Architecture (Vol. 2)*, S. S. Iyengar and A. Elfes, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 152–161.

[12] A. Chandarr, "Towards bootstrapping robotic perception of indoor environments," Master's thesis, Delft University of Technology, 2012.

[13] M. Ciocarlie, K. Hsiao, G. Jones, S. Chitta, R. B. Rusu, and I. Sucan, "Towards Reliable Grasping and Manipulation in Household Environments," in *Proceedings of 12th International Symposium on Experimental Robotics (ISER)*, Delhi, India, December 18-21 2010.

[14] A. Edsinger and C. C. Kemp, "Manipulation in human environments," in *in Intl Conf Humanoid Robots. IEEE*, 2006.

[15] R. J. Firby, "Adaptive execution in complex dynamic worlds," Tech. Rep., 1989.

[16] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots," in *Proceedings of the tenth national conference on Artificial intelligence*, ser. AAAI'92. AAAI Press, 1992, pp. 809–815.

[17] Z. Kalal, J. Matas, and K. Mikolajczyk, "P-n learning: Bootstrapping binary classifiers by structural constraints," in *In IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 49–56.

[18] P. Kannus, H. Sievnen, M. Palvanen, T. Jrvinen, and J. Parkkari, "Prevention of falls and consequent injuries in elderly people," *The Lancet*, vol. 366, pp. 1885 – 1893, 2005.

[19] G. A. Kragten, "Underactuated hands: Fundamentals, performance analysis and design," Ph.D. dissertation, Delt University of Technology, 2011.

[20] J. Kuffner, J.J. and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 2, 2000, pp. 995 –1001 vol.2.

[21] K. Mikolajczyk, Z. Kalal, and J. Matas, "Online learning of robust object detectors during unstable tracking," in *3rd Online Learning for Computer Vision Workshop,Kyoto, Japan, IEEE CS*, 2009.

[22] A. T. Miller and P. K. Allen, "Graspit!: A versatile simulator for robotic grasping," *IEEE Robotics and Automation Magazine*, vol. 11, pp. 110–122, 2004.

[23] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams, "Remote agent: To boldly go where no ai system has gone before," 1998.

[24] N. J. Nilsson, "A mobius automation: an application of artificial intelligence techniques," in *Proceedings of the 1st international joint conference on Artificial intelligence*, ser. IJCAI'69. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1969, pp. 509–520.

[25] M. Rudinac, "Exploration and learning for cognitive robots," Ph.D. dissertation, Delft University of Technology, 2013.

[26] G. N. Saridis, "Architectures for intelligent controls. in: Intelligent control systems: Theory and applications." IEEE Press, 1995.

[27] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," 2008.

[28] Z. Z. Shpunt Alexander, "Three-dimensional sensing using speckle patterns," Patent 20 090 096 783, April, 2009.

[29] J. Stückler, D. Holz, and S. Behnke, "Robocup@home: Demonstrating everyday manipulation skills in robocup@home," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 2, pp. 34–42, June 2012.

[30] I. A. Sucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *WAFR'08*, 2008, pp. 449–464.

[31] N. Vahrenkamp, M. Do, T. Asfour, and R. Dillmann, "Integrated Grasp and motion planning," May 2010, pp. 2883–2888.