

BACHELOR THESIS 2

Biomedical Engineering – Medical and Hospital Engineering

Creation of a robot-user interface for persons with reduced motor capabilities

Executed by: Simon Schmid
Personal Identifier: 1010227051

Advised by: Dipl.-Ing. Christoph Veigl

Vienna, May 13, 2013



Declaration

„I confirm that this paper is entirely my own work. All sources and quotations have been fully acknowledged in the appropriate places with adequate footnotes and citations. Quotations have been properly acknowledged and marked with appropriate punctuation. The works consulted are listed in the bibliography. This paper has not been submitted to another examination panel in the same or a similar form, and has not been published. I declare that the present paper is identical to the version uploaded.”

Place, Date

Signature

Kurzfassung

Im Laufe dieser Bachelorarbeit wird ein Roboter-Modell für Menschen mit Behinderung entwickelt um eine größtmöglich Selbstständigkeit zu gewährleisten. Dabei können verschiedene Sensoren wie Gesichts- oder Augen-Scanner, einer Gehirn- Computer Schnittstelle (BNCI), Joysticks, uvm. zur Steuerung verwendet werden. Durch die Verschmelzung dieser Sensoren mit dem AsTeRICS System (Assistive Technology Rapid Integration and Construction Set) können verschiedene Geräte wie Computer oder „on-screen keyboards“ einfach über eine einzige Plattform gesteuert werden. Durch Verwendung dieser Sensoren und dem AsTeRICS System soll die Steuerung eines Roboters und des darauf montierten Roboterarms ermöglicht werden. Es wird ein Roboter des Typs Pioneer3-AT verwendet. Der Roboterarm wurde von der Firma Neuronics hergestellt und ist ein sogenannter „Katana Arm“. In dieser Bachelorarbeit werden alle nötigen Funktionen, die für die Steuerung notwendig sind, eingefügt: der Roboter wird mit Hilfe eines „on-screen keyboards“ vorwärts, rückwärts, links, rechts fahren, sowie verschiedene Objekte greifen können. Des Weiteren wird eine automatisierte Navigation implementiert, mit welcher der Roboter bis zu fünf verschiedene Positionen selbstständig anfahren kann. Das für die Steuerung verwendete „on-screen keyboard“ wird auf einem Rechner mit dem Betriebssystem Windows betrieben. Der Roboter empfängt den Befehl über das Netzwerk und bewegt sich in die gewünschte Richtung. Das Roboter Modell wurde mit einem Simulationsprogramm sowie mit dem Roboter selbst getestet und es zeigte sich, dass sich der Roboter wie vom Benutzer gewünscht bewegt.

Schlagwörter: Pioneer3-AT, On Screen Keyboard, AsTeRICS, ROS, OSA, Handicapped Person

Abstract

In the course of this Bachelor thesis, a robot interface for handicapped people with the use of different sensor techniques like brain computer interfaces (BCI), face- or eye-tracking, momentary switches, mice or joysticks, puff and sip sensors, and many more will be created. Through merging these sensors within the AsTeRICS system (Assistive Technology Rapid Integration and Construction Set), various devices such as personal computers or on-screen keyboards can be easily controlled via a single platform. With the use of the AsTeRICS framework and the mentioned devices, the manipulation and navigation of a four-wheeled robot platform called 'PioneerP3-at' shall be established. In this bachelor thesis, the basic movements of the robot and a mounted katana arm, as also an intelligent collision query have been implemented by utilizing and extending functions and modules of the Robot Operating System (ROS) which is a widely used open source collection of algorithms for robot control. For this purpose, a suitable grid with selectable cells for directions has been designed for the OSKA on-screen keyboard application. This on-screen keyboard runs on any computer with Windows operating system installed and communicates with the AsTeRICS system via TCP/IP sockets. In AsTeRICS models, developed in this thesis, the robot and katana arm can be controlled by various input devices. The mobile platform receives the command via the network and moves as desired. The model was successfully tested with a simulation tool and then executed on the real robot platform, a PioneerP3-at system. The tests showed that the robot moves as desired.

Keywords: PioneerP3-at, On Screen Keyboard, AsTeRICS, ROS, OSKA, Handicapped Person

Table of content

1	Introduction	7
1.1	Potential Target Group.....	10
2	Material and Methods	11
2.1	AsTeRICS system overview.....	11
2.1.1	AsTeRICS Runtime Environment (ARE)	12
2.1.2	The AsTeRICS Configuration Suite (ACS)	13
2.2	ROS overview	13
2.2.1	ROS basics.....	14
2.2.2	ROS services.....	15
2.2.3	ROS actionlib package.....	15
2.2.4	ROS launch files	16
2.2.5	Transformations and parent-child dependencies.....	17
2.3	Sonar	17
2.4	Point cloud	18
2.5	Infrared distance measurement.....	19
2.6	Kinematics	20
2.6.1	Forward kinematics.....	20
2.6.2	Inverse kinematics	20
2.7	TCP/IP connection	21
2.8	Robot 'PioneerP3-at' overview	21
2.9	Use Cases	22
2.9.1	Navigation via “one button” switches.....	22
2.9.2	Navigation via Face tracking	23
3	Implementation	24
3.1	The final version of the AsTeRICS-Pioneer model	24
3.1.1	The graphical user interface (GUI)	26
3.2	The final version of the ROS program	28
3.2.1	P2OS driver	29
3.2.2	Katana package.....	30
3.2.3	Point cloud to laser scan	31
3.2.4	OpenNI camera.....	33
3.2.5	Static transform.....	34
3.2.6	Map server and slam mapping	34
3.2.6.1	Slam gmapping	34
3.2.6.2	Map server	35
3.2.7	Costmap	35

3.2.8	AMCL.....	36
3.2.9	Move base	37
3.2.10	Asterics Pioneer Robot	37
3.2.11	Asterics Pioneer Network.....	38
4	Results.....	38
4.1	Testing the robot-model by simulation.....	38
4.2	Testing the robot-model with the Pioneer-p3at.....	40
5	Discussion	40
6	Conclusion	40
7	Further Challenges	40

1 Introduction

People with reduced motor capabilities often have problems using standard input methods for computers and other information technologies. The AsTeRICS (“Assistive Technology Rapid Integration and Construction Set” [1]) therefore provides software for “Assistive Technologies” which provides handicapped persons an easy access to human-machine-interfaces (HMI). There are various devices on the market, but they do not allow a full access to or an easy modification of the device capabilities. The therapist or handicapped person therefore cannot adapt the software by him or herself easily but has to call a specialist. Through the use of an open source software it is possible that the therapist also can modify or adapt the software via the Internet through the usage of a library where different models are stored and uploaded. So the replacement or adaption of different functions can be easily done. Also the ROS framework (Robot Operating System) which is used for programming various robot platforms is an open source software which allows programmers to easily extend the software and the robot itself with special functions or additional manipulators like for example a robot arm.

Ambient Assisted Living (AAL) is nowadays omnipresent. Many people with disabilities are already using a variety of devices which help them to cope with their everyday life easier or to do various things that were previously impossible. Examples are electric wheelchairs, automatic door openers and “accessibility supports” for the computer or even whole “smart home managements”. On-screen keyboards allow selection of letters, icons or symbols with single switches or sensors via so-called “scanning” methods, where rows and columns are highlighted and the person uses a desired sensor to select the respective cell on the graphical keyboard. The AsTeRICS system provides a configurable on-screen keyboard application (“OSKA” [2]), where fully graphical keyboard grids can be designed with a dedicated editor. A cell selection by the user can trigger events in the AsTeRICS Runtime Environment (ARE), where they can be linked with desired actions. By the use of a “face tracking-“ or “eye tracking-system” via “point and click”- operations, the cursor can be controlled through head- or eye movements and cells of an on-screen keyboard can be efficiently selected [3]. As on-screen keyboards are widely used, there are a lot of researches on different scanning processes to find out which is the best way to select a single button. This is important when handicapped person only can use a binary input device (like a puff and sip sensor). An additional and novel type of support is the use of robots, which are equipped with a manipulator arm and can recognize different objects, grab and carry them. These robots are designed especially to help elderly and/or people with reduced motor capabilities. One example is the recent introduction of a household robot, developed by the company Toyota [4]. Also the “Technische Universität Wien” (TU Wien) is developing a robot

arm which can identify and grab a specific object [5]. Therefore the idea came up to merge the AsTeRICS software with ROS to develop a support robot. With the use of an adaptable input interface for people with special needs via AsTeRICS and OSKA and the use of a robot that can easily be extended, several assistive functions can be established. The system software for the input devices is programmed via the AsTeRICS software and the robot control is implemented via the Robot Operating System. For the movement of the robot, the AsTeRICS software calls up different functions of a robot plugin, as soon as a direction-button on the on-screen keyboard is pressed. This plugin then sends the command to move in the desired direction to the robot via the wireless network (TCP/IP). The AsTeRICS plugins are programmed in the program language JAVA. ROS is based on the programming languages C++ and Python.

Summarized, the goal of this thesis is to develop a robot-user interface including an on-screen keyboard and a suitable scanning method for the buttons. Also a robot-model will be created in the ROS framework, which can execute a command received from the network.

In the course of this Bachelor thesis, the mobile platform will be able to move around by following commands:

- Move forward
- Move backward
- Turn right
- Turn left
- Stop
- Turn around
- Turn 90° left
- Turn 90° right
- Move forward in a left curve
- Move backward in a left curve
- Move forward in a right curve
- Move forward in a right curve

Additionally to the manual navigation, an automatic navigation function is added. That means the robot will be able to move to four different locations by its own as also the current position can be saved and recalled at any time.

The user will also be able to use a katana arm to interact with the environment. Therefore following commands for the katana movements were implemented:

- Move forward
- Move backward
- Move left
- Move right
- Move up
- Move down
- Grab
- Release
- Save the current position
- Move to the saved position
- Move to the initial position
- Raise the movement width
- Lower the movement width

As there are three different movement cases, there will be three different on screen keyboards available. The user can switch between those anytime in runtime.

In the following chapter (2.1 AsTeRICS system overview) the AsTeRICS framework is described and the possibilities for navigating the robot are outlined. Subsequently in chapter 2.2 the Robot Operating System ROS and for this thesis necessary ROS functions are described. Chapter 2.4 describes what a point cloud is. Chapter 2.5 describes how an infrared distance measurement works and chapter 2.6 describes the term kinematics. In chapter 2.7 a TCP/IP connection is described and in chapter 2.8, the utilized robot platform and its modules and functions are presented. Chapter 2.9 describes two use cases. In chapter 3, the whole implementation process and all implemented functions are described. Chapter 4 presents the results. Chapter 5 and 6 concludes with a discussed and some future prospects are mentioned in chapter 7.

It has to be mentioned that all terms by whatever means, either male or female, that they have to be interpreted gender-free and that both sexes are meant, although it is not as such mentioned.

1.1 Potential Target Group

As the number of people with reduced motor capabilities has not reduced in the last years the assistive technologies market and its products have developed very fast. More than three million people worldwide are affected by paraplegia. Of these three million people around 52% have hemiplegia and around 46% have tetraplegia [6]. People with extremely reduced motor capabilities can often only use single buttons like momentary switches or puff- and sip-sensors and therefore depend on assistive devices where the selection of the functions is adapted to the physical condition of the user. With the AsTeRICS system it is possible to use an input device designed specifically for one particular user, and with the implementation of ROS it is possible to link different robot modules with AsTeRICS. So this thesis provides a robot-user interface for persons with such physical conditions in particular, and in general for all people with motor impairments. So it addresses a wide ranged group. But not only handicapped persons can benefit from this idea, also elderly people who have reduced fine motor skills could use a service robot system for simple tasks like fetching things or carrying objects with the use of a robot arm.

2 Material and Methods

2.1 AsTeRICS system overview

AsTeRICS consists of different components: the hardware modules and the software framework. A great variety of the hardware modules are available. For this thesis, only standard mouse input and a webcam for face detection are used. As there are a lot of hardware modules, the system can easily be personalized to special needs of a person. The actually useable sensors for navigating the robot are described in Chapter 2.4. The full list of available Sensors and Actuators can be looked up in the User Manual which can be found the download section of the official AsTeRICS website [7]. Figure 1 shows the concept of the AsTeRICS system.

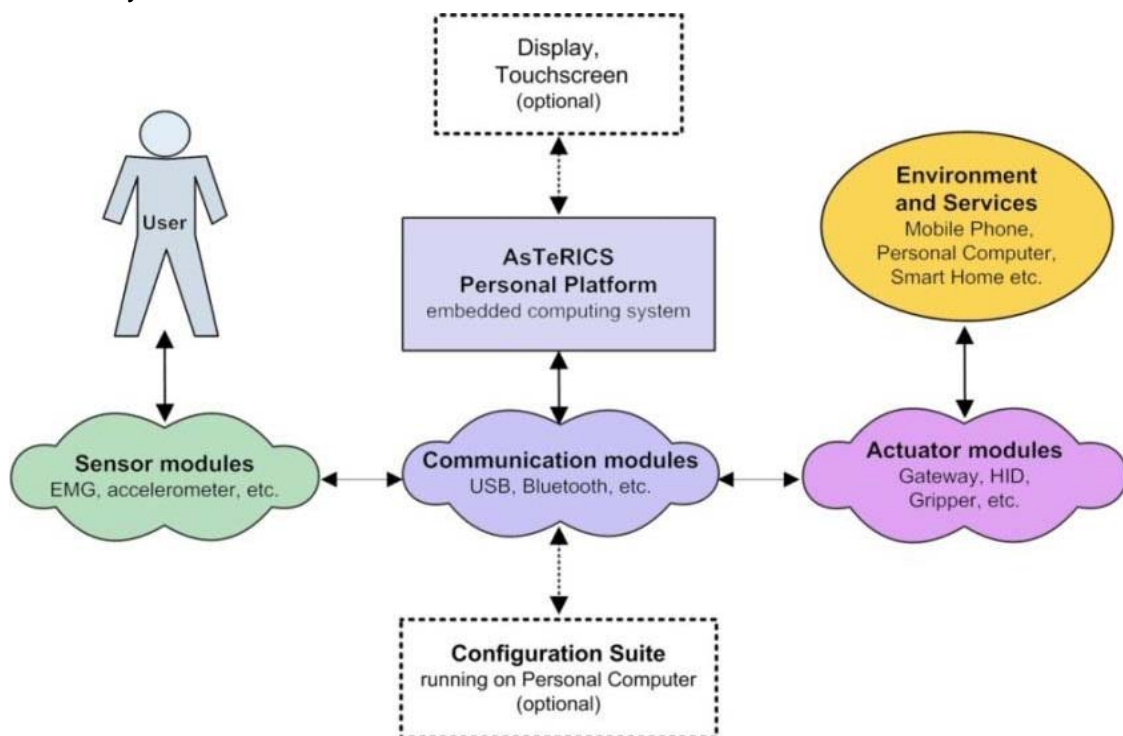


Figure 1: Concept of the AsTeRICS system (Source [1], S. 6)

The hardware components include all sensors- and actuator modules, the Communication Interface Modules (CIMs) and a computing platform like a laptop, tablet, etc. The sensors and actuators are used to establish an interaction between the user and his environment. The CIM then provides an interface and connects the sensors and actuators to the computing platform where the ARE (AsTeRICS Runtime Environment) runs. As computing platform any device with Windows operating system can be used. The ARE provides all the functions which are included in the currently loaded model or application. The application or

models have functions for signal processing of the actuators and sensors. These applications are programmed with the AsTeRICS Configuration Suite and can be loaded in the ARE through a TCP/IP connection. Thus, models can be adapted and sent to the user's ARE via an internet connection.

The following two chapters describe the AsTeRICS Configuration Suite and the AsTeRICS Runtime Environment for further understanding of this Bachelor thesis.

2.1.1 AsTeRICS Runtime Environment (ARE)

The ARE hosts applications which contain the plugins. These plugins provide all the functionalities to the application. AsTeRICS applications are also called models or configurations. In the ARE, the user can start, pause, stop or deploy a model. The ARE can be run on a computer with the operating system Windows or on the AsTeRICS personal platform. The AsTeRICS personal platform is an embedded computing platform where input devices can be connected and models can be run. The models are configured and designed with the AsTeRICS Configuration Suite which is described in the next chapter. Figure 2 shows how the ARE graphical user interface can look like. The “Main Panel” can contain desired GUI-elements for graphical feedback or manipulation of model parameters which can be arranged in the ACS GUI designer window. On the “Control Panel” there are four buttons for loading, starting, pause or stop a model. The “Main Menu” provides further options and models can also be loaded via the “Main Menu”.

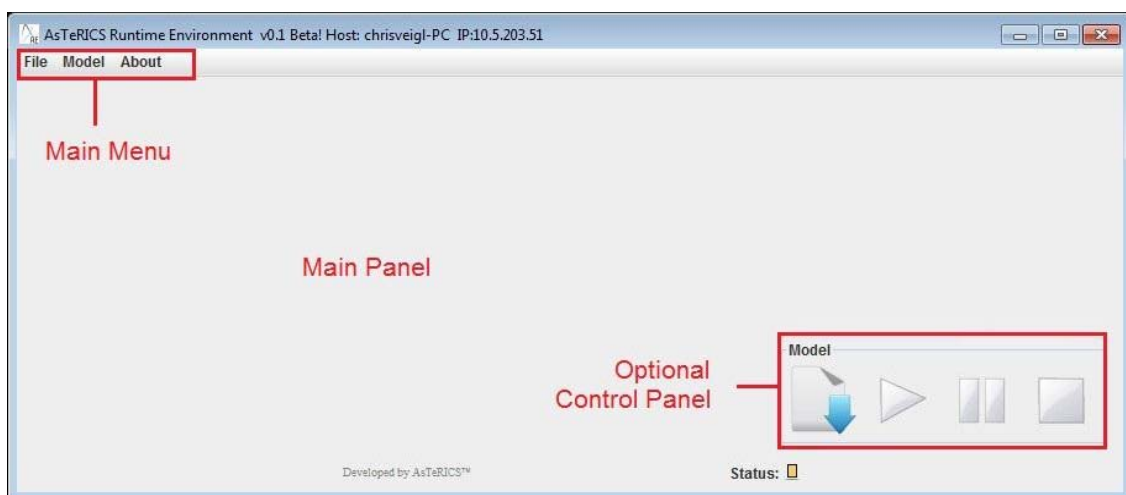


Figure 2: Graphical User Interface of the ARE (Source [1], S. 14)

2.1.2 The AsTeRICS Configuration Suite (ACS)

With the AsTeRICS Configuration Suite, models can be developed and edited. They also can be uploaded or downloaded from/to the AsTeRICS Runtime Environment. It is also possible to start, pause and stop a model through the ACS and it does not have to reside on the same system as the ARE. The ACS offers all the sensor-, processor- and actuator plugins which can be used. They easily can be placed and connected via drag and drop.

Figure 3 shows how the ACS looks like. The plugins can be placed and connected in the deployment area (2), which actually constitutes the runnable AsTeRICS model. This model can then be uploaded and executed on the ARE. Using the main menu, the user can save, open or create a model. In the main menu toolbar (1) the user can interact with the ARE or select the components and place them in the deployment area. The ACS includes also a plugin creation wizard which allows the creation of a JAVA skeleton for a new plugin by specifying the plugin's input-, output- and event-ports.

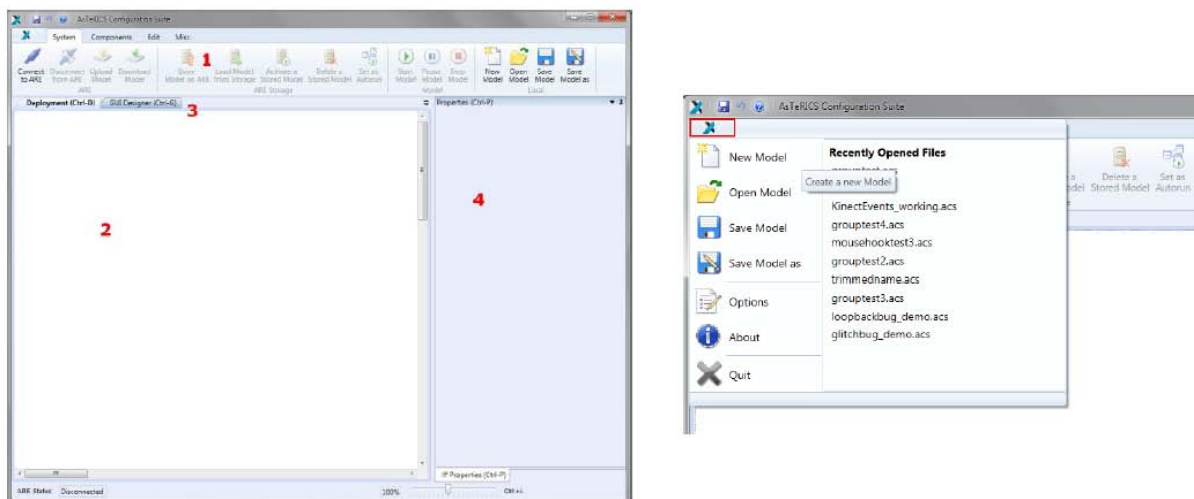


Figure 3: Graphical User Interface of the AsTeRICS Configuration Suite (Source [1], S. 16)

In the figure above, the menu area (1), the deployment area (2), the GUI area (3) and the properties (4) are shown. The main menu appears when clicking the AsTeRICS button (seen on the right).

2.2 ROS overview

The Robot Operating System is an open source framework for robot software development. It was developed by the Stanford Artificial Intelligence Laboratory. Nowadays it is wide spread and commonly used in commercial and educational projects. It is designed in a way

that it is easy to access and exchange newly programmed packages. A Package (also called “stack”) contains all the executables and processes which the robot will use.

According to the developers of the ROS framework the philosophical goals can be summarized as:

- Peer-to-peer
- Tools-based
- Multi-lingual
- Thin
- Free and Open-Source [8]

ROS has many components, tasks and services which provide hardware abstraction, device control, re-use of functionalities, message exchange between programs or program parts, package management and exchange of libraries for usage on multiple computers. Also a big variety of different robot types are compatible with ROS. For the full list of usable Robots, visit www.ros.org/wiki/Robots. In this thesis, the “Pioneer P3-at” robot platform is used (described in chapter 2.8). In addition, a full list of the compatible sensors exists at www.ros.org/wiki/Sensors.

In the following section, the basics and some functions of the ROS framework are described in detail for further understanding of this Bachelor thesis.

2.2.1 ROS basics

The Robot Operating System bases on a topic system where various messages can be subscribed to. These messages could contain for example a command for moving the robot to the right. The topics and messages are defined in a node. Such nodes can communicate with each other through sending messages to a specific topic. This system is described in detail with the help of the following picture:

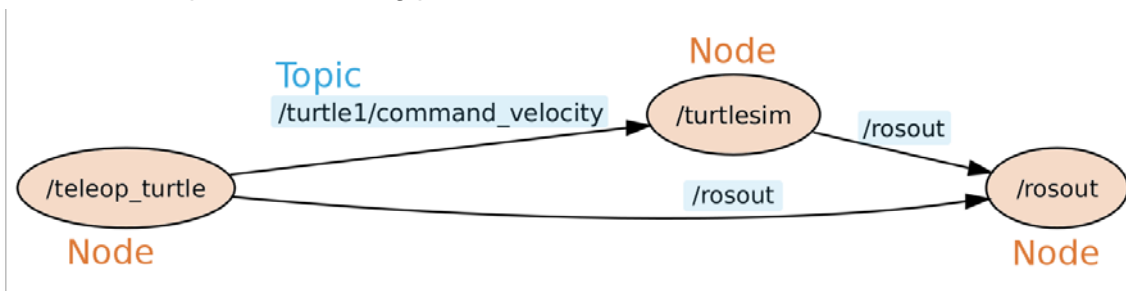


Figure 4: Example of a ROS program with different nodes and topics (Source [9])

In this example a little turtle can be moved on the screen using the arrow keys of the keyboard. The node `/turtlesim` contains the topic `/turtle1/command_velocity` which is responsible for moving the robot in the desired direction. The node `/teleop_turtle` publishes a message to this topic. The message contains the values for the linear and angular movement. The node `/turtlesim` also defines the graphical interface seen on the screen. All two nodes are subscribing to the `/rosout` node which can be referred to as the endpoint or output of the program.

The most important part of ROS is the Master. Its task is to act as a nameservice and to store topic- and service- registration information for the nodes. Without the master, nodes would not be able to exchange information or even find each other. Nodes communicate with the Master to get information about other nodes. This enables the nodes to create connections among them. That means nodes do communicate directly, the Master only provides lookup information.

2.2.2 ROS services

For some functions, a request / reply interaction is needed. But the standard ROS publish / subscribe model is not really suitable for such interactions. Therefore ROS uses services which are defined by a pair of message structure where one is for the request and one for the reply. A node can use the service by sending the request message to the node where the service is defined and then waits for the reply.

2.2.3 ROS actionlib package

As described in chapter 2.2.2, with ROS services it is possible to send a request to a node to perform a task and then receive a reply to the request. But if the service request takes too long, the user may want to cancel the request during execution or get a feedback about the request's progress. Therefore the actionlib package provides a server-client model for such preemptable tasks. The so called "ActionServer" and "ActionClient" communicate via a "ROS Action Protocol". This protocol is built on top of ROS messages. On the Client side the user can request a goal which then gets executed on the server side. The following figure shows the scheme of the "ActionServer" and "ActionClient".

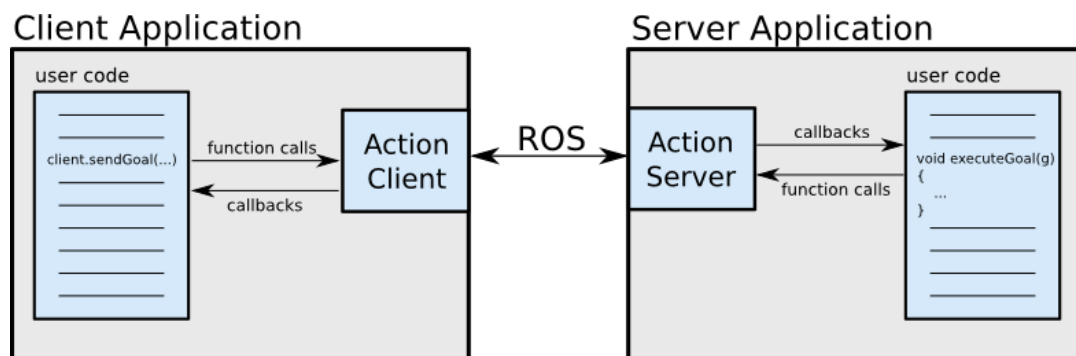


Figure 5: ActionServer and ActionClient communication scheme (Source [12])

The client and server communicate via “Goal”, “Feedback” and “Result” messages which are described below.

Goal

The “ActionClient” can send a goal message to the “ActionServer” to accomplish a task. For example if the user wants to move a robot, the goal message would contain information about where the robot should move to.

Feedback

The feedback message is send periodically from the “ActionServer” to the “ActionClient” and tells the client about the progress of the goal. For example in the case of moving the robot, the feedback message would contain information about the robots current position.

Result

The “ActionServer” sends a result to the “ActionClient”. The result contains information about the completion of the goal. The result can either be that the goal was achieved or not. The result message is sent only once to the “ActionClient”. In the example of moving the robot, the result would be that the desired position was reached.

2.2.4 ROS launch files

For launching multiple ROS nodes locally or remotely and for setting various parameters, ROS offers the launch files. In these launch file, every node that should get started and every parameter which should get set is written down. Also launch files for other nodes can be invoked. The commands in the launch file are invoked consecutively. The following figure shows an example of a launch file. In the first line, the launch file for the node “p2os_driver” gets executed. In the second part, the nodes for converting a point cloud into a laser scan

are started and in the third section the nodes for the front and back camera drivers are stated.

```
<launch>

  <!-- ##### START P2OS ROBOT DRIVER ##### -->

  <include file="$(find p2os_launch)/p2os_driver.launch" />

  <!-- ##### CONVERT POINTCLOUD TO LASERSCAN ##### -->

  <node pkg="pointcloud2laser_front" type="pointcloud2laser_front" name="pointcloud2laser_front" output="screen">
    <!--<remap from="Laser_Camera_Front" to="Laser_Camera">-->
  </ node>

  <node pkg="pointcloud2laser_back" type="pointcloud2laser_back" name="pointcloud2laser_back" output="screen">
    <!--<remap from="Laser_Camera_Back" to="Laser_Camera">-->
  </ node>

  <!-- ##### START ASUS FRONT AND BACK CAMERA ##### -->

  <include file="$(find openni_launch)/launch/openni.launch">
    <arg name="camera" value="asus_front" />
    <arg name="device_id" value="#2"/>
  </include>

  <include file="$(find openni_launch)/launch/openni.launch">
    <arg name="camera" value="asus_back" />
    <arg name="device_id" value="#1"/>
  </include>

</ launch>
```

Figure 6: Example of a launch file for converting a point cloud into a laser scan

2.2.5 Transformations and parent-child dependencies

With the help of transformations the user can keep track of multiple coordinate frames over time. The relationships between every coordinate frame bases on a tree structure which is buffered in time. Any robotic system normally has a lot of 3D coordinate frames which change over time. When relationships are defined between every part of the robotic system the user can recall every frame's position in the world at any time. As transformations base on a tree structure, one frame can only have a single "parent" but one "parent" can have multiple "children".

Examples for such dependencies and transformations can be seen in chapter 3.2.2 and chapter 3.2.3.

2.3 Sonar

Active sonar is used to measure the distance to an object. Therefore it uses a sound transmitter and a receiver. There are three operation modes:

- **Monostatic operation mode** – the transmitter and receiver are localized in the same place
- **Bistatic operation mode** – the transmitter and receiver are separated
- **Multistatic operation mode** – more than one transmitters or receivers are used which are spatially separated

A pulse of sound (called “ping”) is created through the transmitter. The receiver then listens for reflections of the radiated pulse. For distance measurements, the time from transmission of the sound wave to reception is measured. By knowing the speed of sound, the measured time can be converted into a range. The pulse of sound is normally generated electronically.

In figure 7 is shown how a monostatic sonar system works. The red waves represent the radiated pulse of sound which are then reflected by an object and sent back to the receiver (green waves). By knowing the duration between sending and receiving the distance to the object can be calculated.

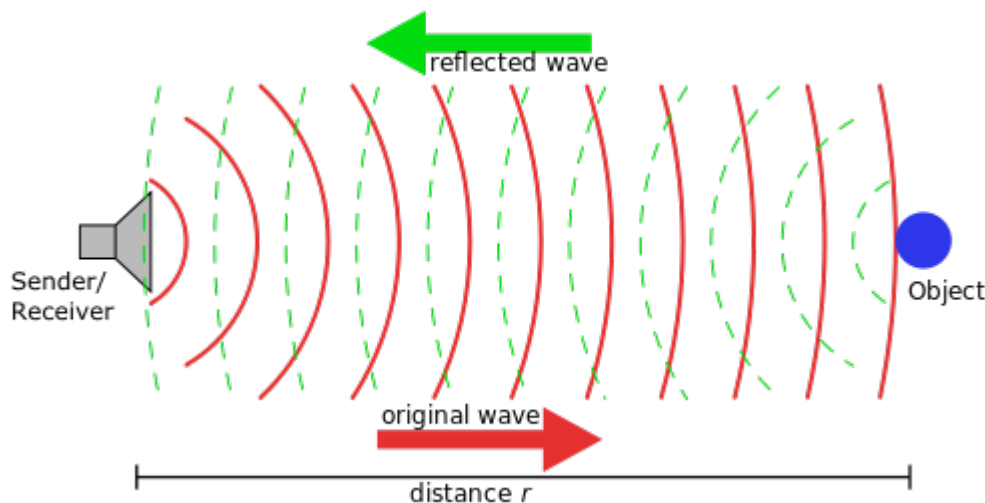


Figure 7: Principle of how sonar works (Source [13])

2.4 Point cloud

A set of vertices in a 3D coordinate system is called point cloud. These vertices typically represent an external surface of an object and are defined by X, Y and Z coordinates. That means every single point of the cloud represents a specific measured point on the recorded object's surface. When color information is added to the point cloud, it becomes 4D.

For the creation of a point cloud normally 3D scanners are used which measure a set of points representing the point cloud. In this bachelor thesis an infrared camera was used. The

infrared camera does nothing else as measuring the distance of a set of points. With the collected data, a point cloud can be created which is done by software.

The following picture shows a typical point cloud of different objects generated from the camera data. As the point cloud also includes the color information of the objects, it is four dimensional.



Figure 8: Example of a recorded point cloud (Source [14])

2.5 Infrared distance measurement

In infrared distance measurement devices a light beam is sent out from an infrared LED. This light beam gets reflected by an object of which the distance should be measured. The reflected light beam comes back and hits a position sensor within the device. Depending on where the light beam hits the position sensor a different voltage gets emitted. This voltage now can be converted into a distance. The following figure shows two light beams reflected from different distances (P1 and P2) resulting in two different voltages U1 and U2.

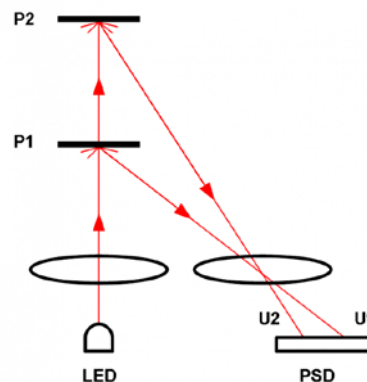


Figure 9: Principle of infrared distance measurement (Source [15])

2.6 Kinematics

In classical mechanics the motion of points, objects and group of objects is described through kinematics. The cause of motion is not included in kinematics. It is also called the geometry of motion. It studies the trajectories and movements of geometric objects, lines and points and their velocity and acceleration. In robotics it is used to describe the motion of joined parts such as robotic arms or engines.

For the use of kinematics with robotic arms there are two approaches for computing the necessary joint movements to reach a desired endeffector position, forward kinematics and inverse kinematics. These two approaches are described in detail in the following two sub chapters.

2.6.1 Forward kinematics

In forward kinematics the endeffector position gets computed from the specific joint parameters. The endeffector position is not known at the beginning, only the angle of each joint is specified. To move the endeffector to a specific position each angle gets altered till the desired position is reached beginning with the hierarchic uppermost joint. With the help of the following picture forward kinematics can be easily described. To move the endeffector to a desired position the first joint (joint1), which is also the hierarchic uppermost point and therefore the start joint, gets moved. Then the remaining joints are also moved till the end position is reached.

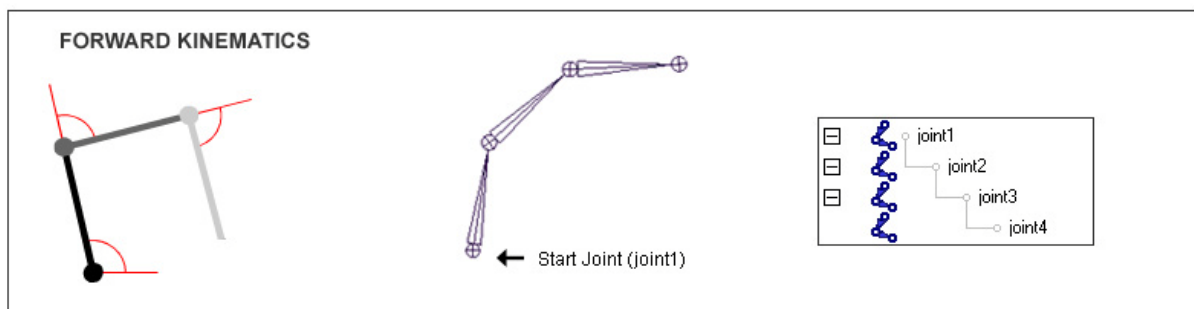


Figure 10: Forward kinematic scheme (Source [16])

2.6.2 Inverse kinematics

Inverse kinematics can be referred to as the reverse process of forward kinematics. At the beginning only the endeffector position is known. When moving this endeffector to a defined point, all joint parameters get calculated with the help of a mathematical expression. For

example when the endeffector (in the picture bellow called ikHandle) should be moved to a specific position, all joints (joint 1 to joint 4) move according to the calculated angle.

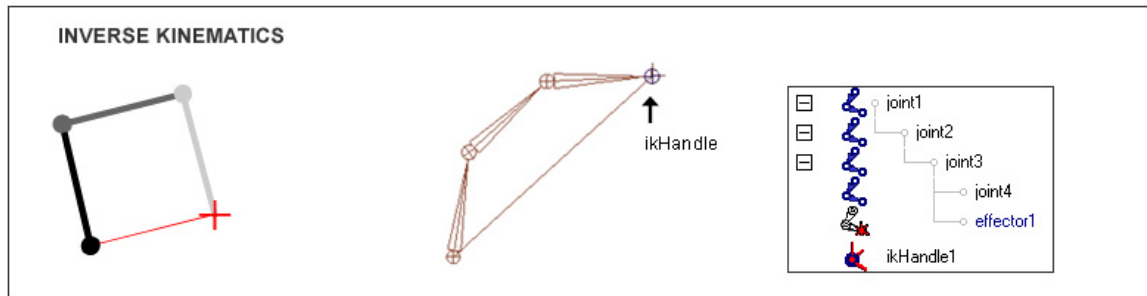


Figure 11: Inverse kinematic scheme (Source [16])

2.7 TCP/IP connection

TCP stands for “Transmission Control Protocol” and IP for “Internet Protocol”. It acts as protocol for communication between a computer and the internet. It defines how the computer should be connected to the internet and how data should be transmitted. TCP is responsible for the communication between the software and IP is responsible for communication between computers. Therefore TCP breaks the data to be sent into packages and passes them on to the IP which is responsible for sending them to the correct destination. TCP can also assemble incoming IP packages. TCP/IP connection is the most common used communication between software and software or computer and computer.

2.8 Robot 'Pioneer3-at' overview

The Pioneer3-at is a four wheel drive robotic platform. It has an aluminum body, four-wheel skid-steer, reversible DC motors, motor-control and drive electronics, high-resolution motion encoders and battery power. These components are all managed by mobile-robot server software which resides on the onboard computer. Because of the onboard computer it is possible to run the client software without any additional PC. The Pioneer3-at can be equipped with additional accessories like manipulator arms and grippers, bumper switches speech and audio systems, laser mapping systems and many more. So with this robot a high customization factor is given and later new features can be easily adapted. As only the robot main functions are implemented in this thesis the equipped sensors will not be used. For more information about the Pioneer3-at visit the official website (www.mobilerobots.com/researchrobots/p3at.aspx).

The following figure (Figure 12) shows how the robot currently looks like. It is equipped with the hardware components (1), a katana arm (2) for grabbing different objects, a front and back bumper switch (3) as also a built in sonar device (4) for collision query and three cameras. Two of these are an Asus Xtion Pro (5) with which a distance measurement is possible and the other is a normal webcam from Logitech (6) for normal image recognition. In the picture below the Logitech webcam is not included but the arrow points to the location is should be mounted.

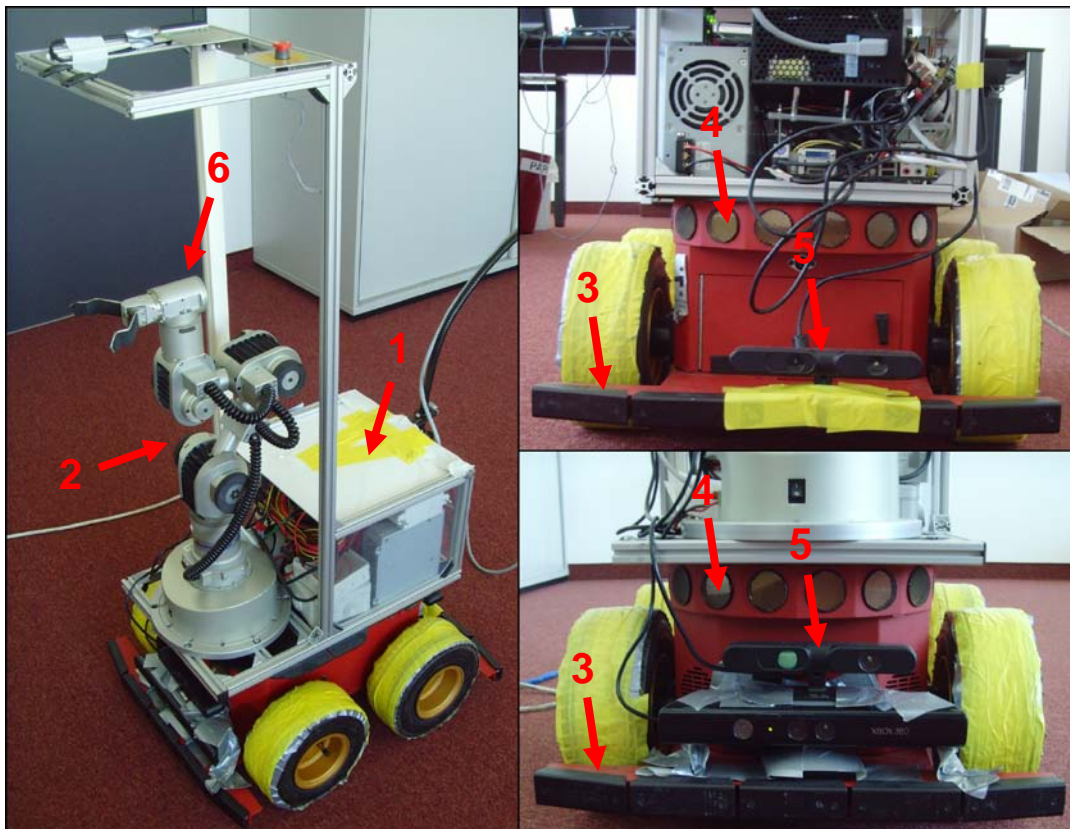


Figure 12: Appearance and current setup of the PioneerP3-at robot

2.9 Use Cases

2.9.1 Navigation via “one button” switches

As OSKA offers different scanning processes which can easily be defined in the ACS various one button switches can be used. For example a puff and sip sensor can be used for the row and cell selection. A use case therefore could be that the user can control the row selection by blowing into the sensor and for cell selection the user has to blow for a certain amount of time. In the case of one button switches any device which can send out a signal by activation can be used.

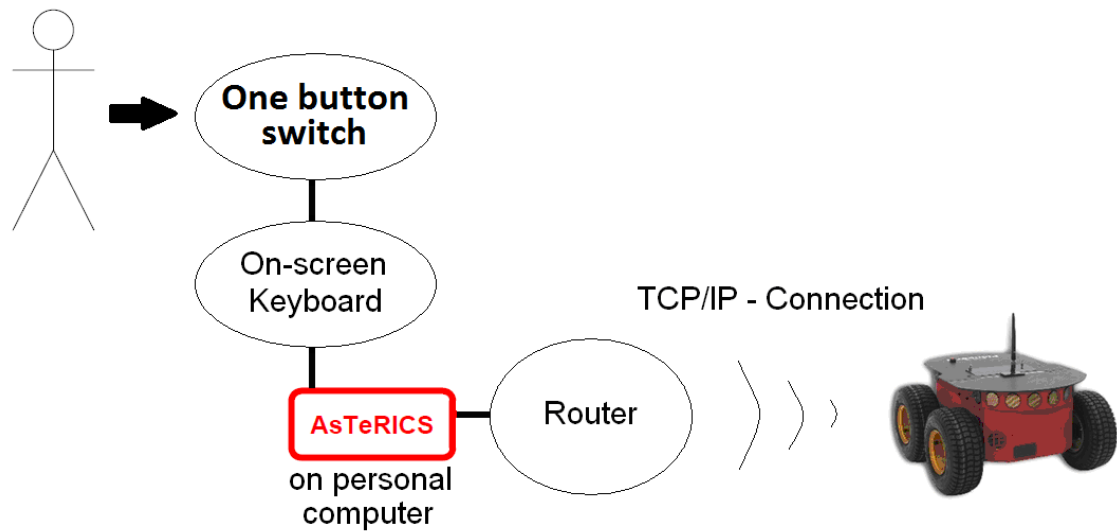


Figure 13: Use-case scenario using one button switches for navigation the robot

2.9.2 Navigation via Face tracking

The head movements are tracked via a normal webcam. The software converts the head movements into a movement of the mouse cursor with which several options can be selected on an on screen keyboard. Then a plugin converts the command in a command the robot understands and sends it to the robot via a TCP/IP connection. The model which does the previous described task is located on a personal computer with ARE running on it. The following Figure shows a scheme of a possible use-case scenario.

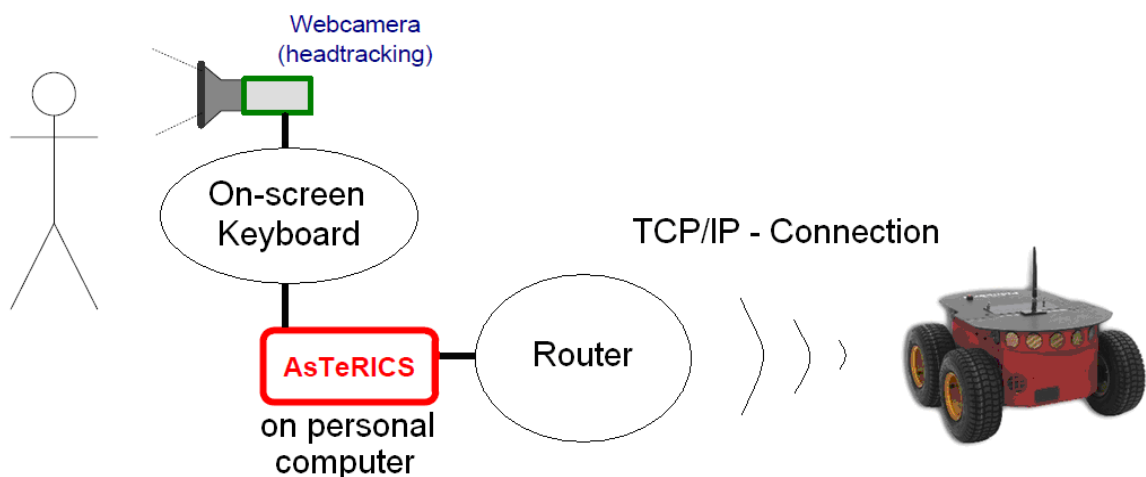


Figure 14: Use-case scenario using a webcam for navigating the robot

3 Implementation

3.1 The final version of the AsTeRICS-Pioneer model

At first, an empty JAVA-skeleton for the pioneer plugin was created with the help of the AsTeRICS plugin creation wizard. Then the specific functions of the plugin were programmed within the Eclipse IDE, which is a universal toolset for development and programming [10]. As AsTeRICS plugins are written in JAVA, Eclipse was configured for this framework. The pioneer plugin code opens up a TCP/IP connection (as a client) to a specified host (the server part is implemented in the ROS framework) and sends a command which is received via a string input port of the plugin to the defined host (robot). For example if the button for the forward movement is pressed, the program converts the specific command into a special format (called "byte array") which then is sent via the network to the IP-address specified via the plugin properties. The IP-address can be edited with the ACS. The developer must ensure that the IP-address of the robot is not changing and is identical with the target IP-address of the AsTeRICS-Pioneer plugin. If the server side is not started yet, the client can still be started because the plugin tries to connect every few seconds and establishes a connection as soon as the server side comes up. When the connection is established the plugin is ready for sending and also receiving messages.

In figure 15 the OSKA-, robot- and text display- plugin inserted in the ACS can be seen. Via the output of the OSKA plugin the desired command is sent to the robot plugin as string. The robot plugin forwards this command to the mobile platform. The plugin also checks every few milliseconds if there is any data to receive and if needed forwards the received message to the text display plugin. On the right side of the figure the editable properties of the robot platform can be seen.

Figure 16 shows how the model looks like in runtime. The upper window shows the text display where the received message gets displayed. The GUI for manual movement of the robot can be seen in the bottom left window. In the bottom right corner of the picture the ARE- console can be seen.

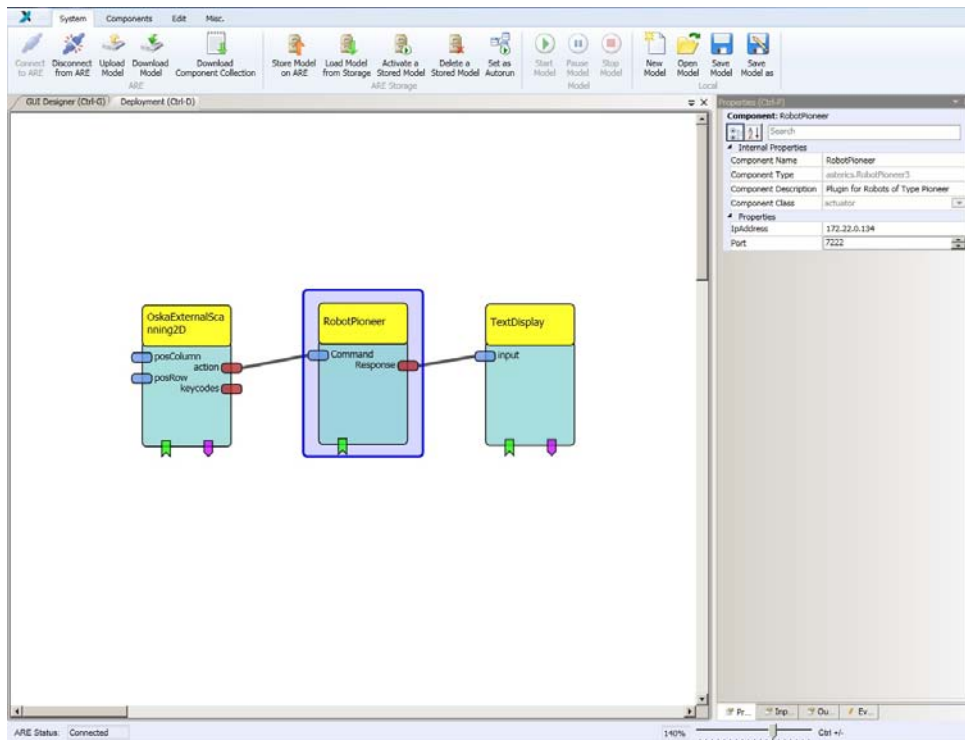


Figure 15: Model setup of controlling the pioneer with the OSKA plugin

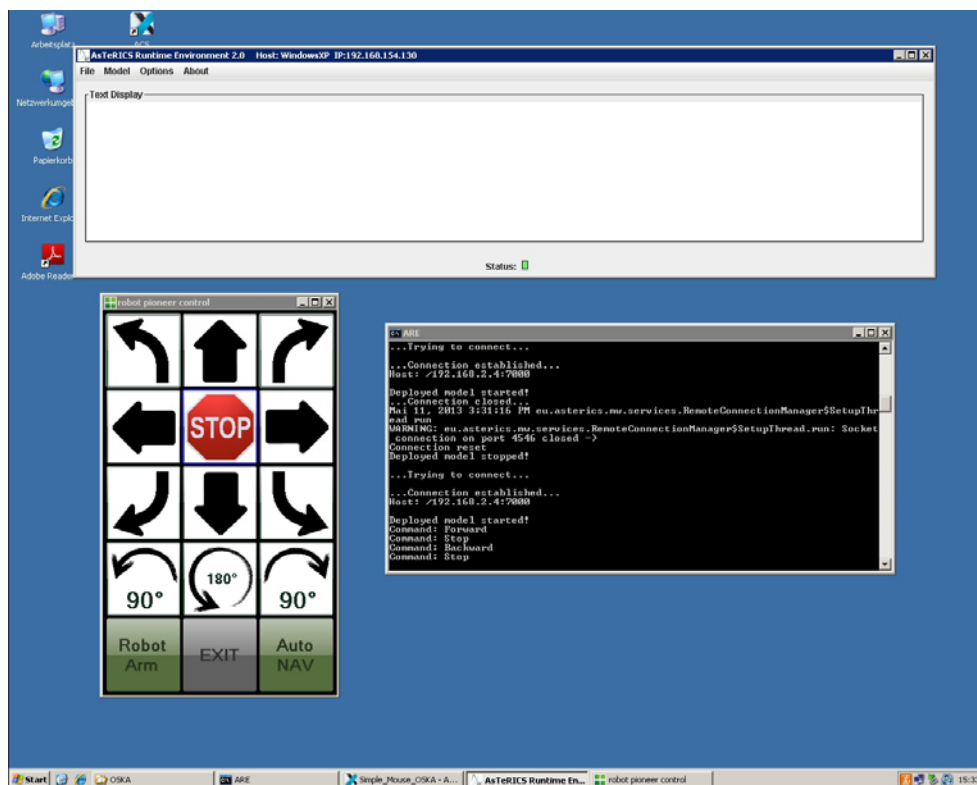


Figure 16: Runtime example of the robot model

3.1.1 The graphical user interface (GUI)

The interaction of an OSKA on-screen keyboard as the primary graphical user interface to the end user depicts the following advantages:

- Easier ACS model due to less components
- OSKA features built-in automatic scanning methods (row/column, single key, ...)
- User friendly design of the GUI with icons is possible in the OSKA grid editor

Therefore the GUI is based on an OSKA keyboard. In the scanning process, the selection of the button will be switched every second from the left to the right. At first the upper left button is selected, then after one second the middle button of the first row is selected and after another second passes the right button will be selected and so on. The user then has the possibility to press a button by simply pressing an external momentary switch, or by pressing the “Enter” key on a special keyboard. For changing the row, the user only has to press the button for a certain amount of time. This type of scanning process is only one method and can easily be changed in the ACS by connecting different signals to the cell selection input ports of the OSKA plugin.

The following figure shows how the GUI for the manual navigation, automatic navigation and for katana navigation looks like. With the on-screen keyboard for manual navigation (figure 17) the user can drive the robot around directly by pressing the desired button. With the keyboard for automatic navigation (figure 18), the robot moves to the desired position like the fridge, shelter, desk or the battery station automatically. Therefore it plans a path and also avoids suddenly appearing obstacles. The current position of the robot can also be saved and recalled later if desired. With the on-screen keyboard for the katana arm (figure 19), the user can control the robot arm simply by pressing a button. The main movements like moving left, right, up, down, forward and backward are implemented. There is also a button for moving the arm back to its initial pose or for saving the current position. The saved position then can be recalled any time.

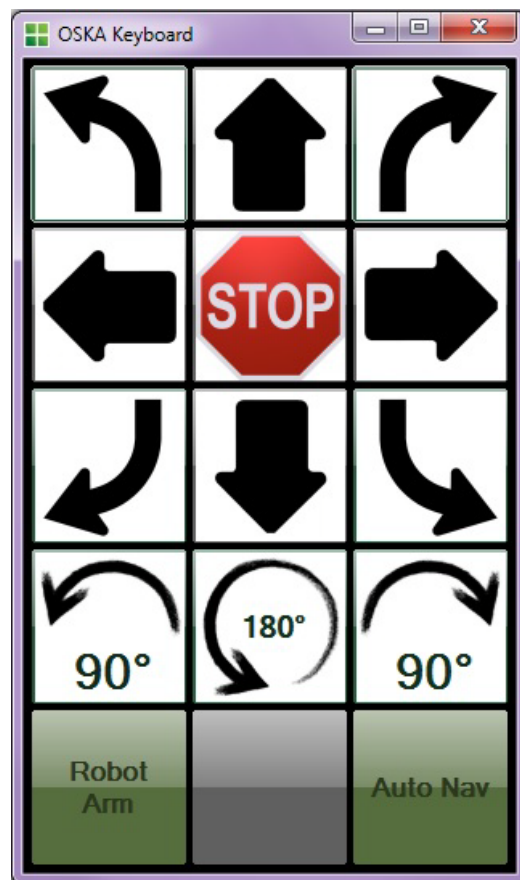


Figure 17: Appearance of the OSKA keyboard for manual navigation



Figure 18: Appearance of the OSKA keyboard for automatic navigation



Figure 19: Appearance of the OSKA keyboard for the katana arm

3.2 The final version of the ROS program

The whole ROS program can be started by executing the “main.launch” file in the package “Asterics_Pioneer_Robot”. By executing this file, the following packages and drivers are activated successively.

- P2OS driver
- Katana arm navigation
- Pointcloud2laser for front camera
- Pointcloud2laser for back camera
- Openni for front camera
- Openni for back camera
- Static transform
- Map server
- Costmap 2D
- AMCL

- Move base
- Asterics Pioneer Robot
- Asterics Pioneer Network

The following chapters explain what each individual stack is doing and how they work.

3.2.1 P2OS driver

This node provides a driver for robots which use a pioneer P2OS/ARCOS interface like the P3-AT Robot. The user can send velocity commands for left, right, front and back movements to this node which then forwards the command to the robot. With the P2OS node the motor state and sonar can also be controlled.

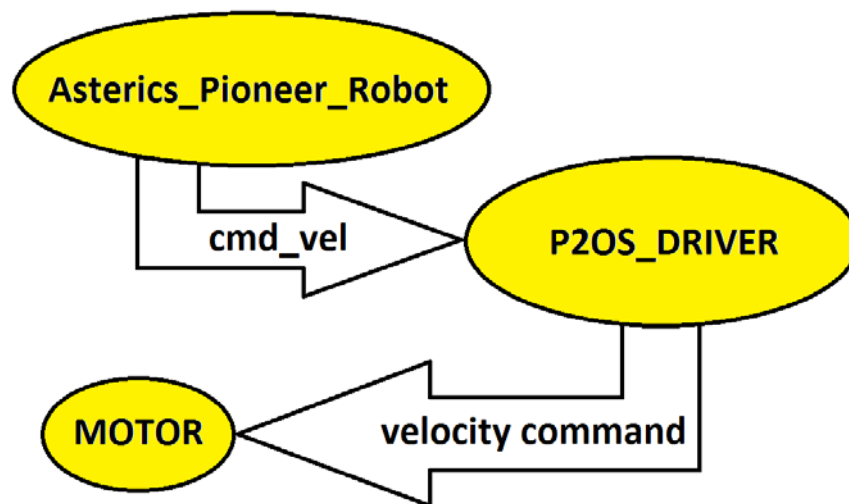


Figure 20: Example of how the P2OS driver works

In figure 20 a “cmd_vel” message gets published from the “Asterics_Pioneer_Robot” to the “p2os_driver” node. This message contains information about the direction the robot should move to. It then forwards the velocity command to the robot’s motor. The node also publishes the topic “pose” (here renamed into “odom”) which contains information about the robot’s current position and movements. The node evaluates the current position through summarizing all movement events so far. But with this information only the robots movements can be tracked, not the actual position on for example a known map. Therefore other nodes like AMCL and map server are needed.

The p2os driver is an open source package which is updated permanently and can be downloaded from the ROS website ^[20]. Nothing of the original package was altered in this bachelor thesis.

3.2.2 Katana package

For interaction and manipulation of objects there are various open source packages available. As on the robot a Neuronics katana arm is mounted, the package “katana_arm_navigation” and “katana_driver” stack were used. The “katana_driver” stack provides the main driver for the katana arm. With the help of the “katana” node the user can control and drive every single motor. The node “katana_arm_navigation” provides the kinematics (inverse and forward) for the arm. This node bases on an actionlib, so the user can send goal commands to this node. That means if a goal where the endeffector should move to is sent to the “katana_arm_navigation” node via a service call, it calculates how each motor has to move to reach the goal. The therefore necessary motor movements are then sent to the “katana” node which triggers the katana arm to move in the desired position.

Figure 22 shows all transforms from the katana arm. The “katana_gripper_link” refers to the endeffector of the arm. The previous links refer to each motor. Figure 21 shows how the katana arm looks like and also the name of each joint can be seen.

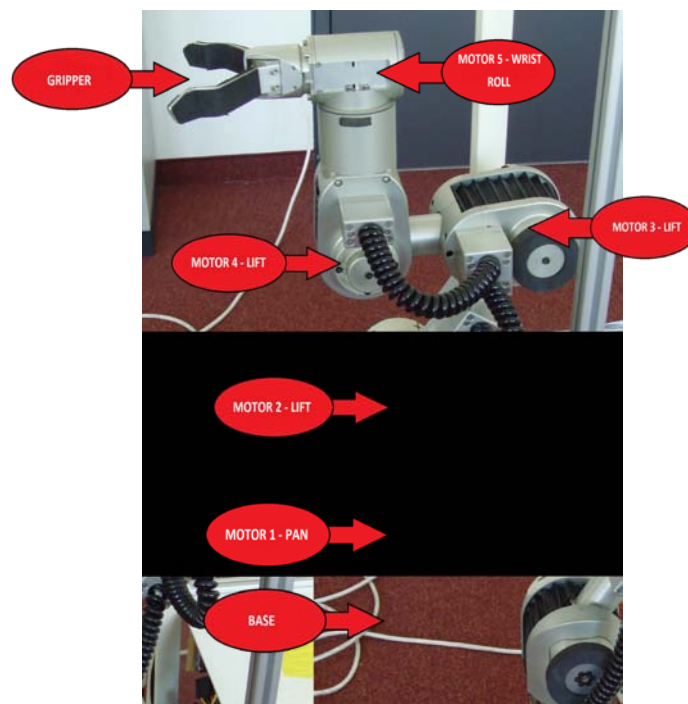


Figure 21: Appearance of the katana arm

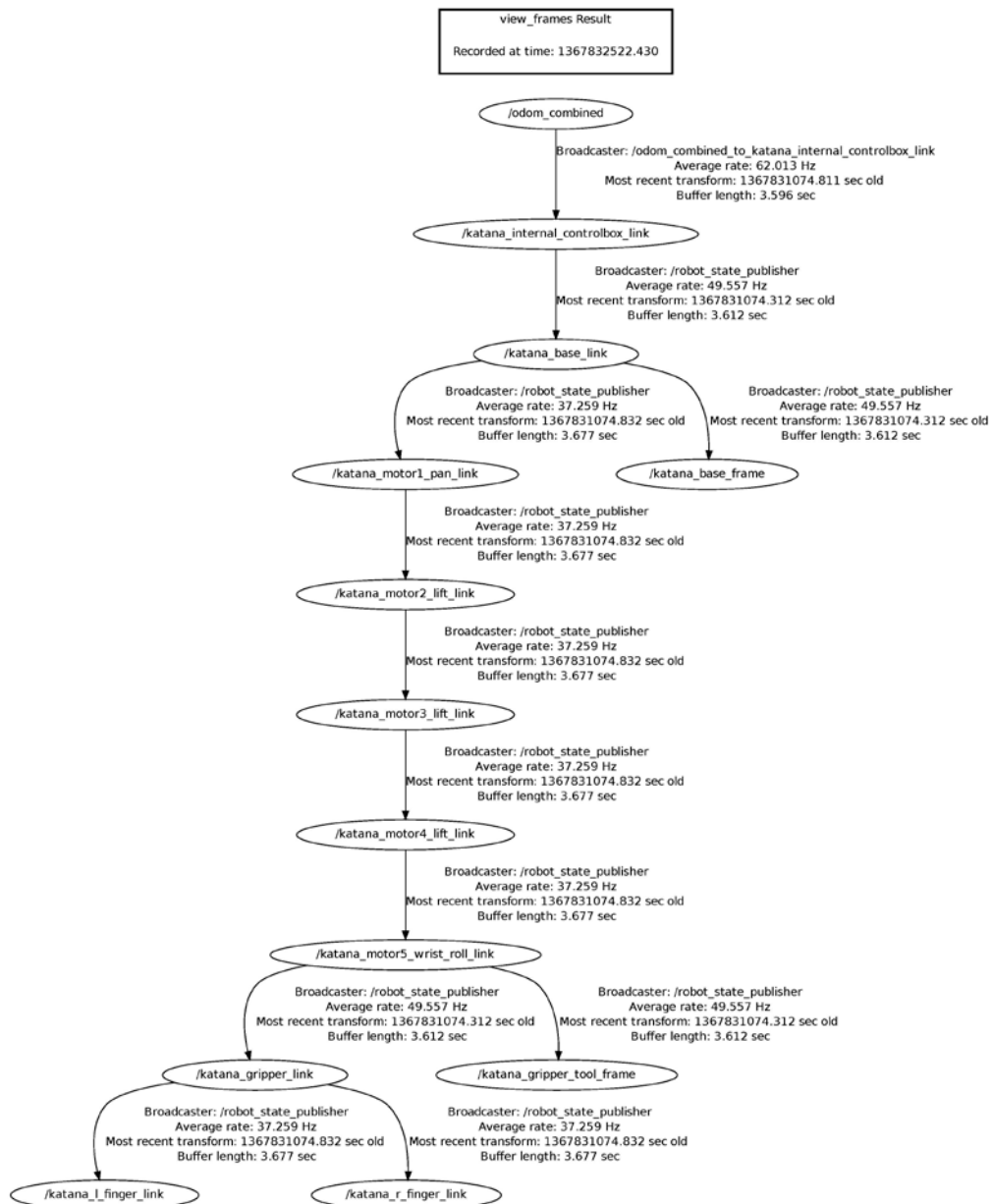


Figure 22: Hierarchic tree of all links of the katana arm parts

3.2.3 Point cloud to laser scan

The “Pointcloud2laser” uses a generated 3D point cloud and converts it into a 2D laser scan. As there are two cameras mounted on the robot (seen and described in chapter 2.8), also two “Pointcloud2laser” nodes were created. One publishes the laser scan for the front and the other for the back.

For the laser scan the following parameters have to be set:

- Minimum height to sample in the point cloud in meters = -0.025
- Maximum height to sample in the point cloud in meters = 0.025
- Minimum scan angle = $-\pi/2$
- Maximum scan angle = $\pi/2$
- Scan rate in seconds = 1/30
- Minimum range in meters = 0.45
- Maximum range in meters = 10
- The frame id of the laser scan = “camera_depth_frame_front”

The node takes the ranges of every point from the point cloud in a user specified plain and saves them in the laser scan message. The plain is defined by the parameters minimum height and maximum height. The point of origin for this plane is defined by the static transform between the base frame, camera frame and the laser scan frame (here the laser scan frames are named “camera_depth_frame_front” and “camera_depth_frame_back”). This parent-child relationship is pictured in figure 23. The frame “base_link” represents the center of the robot. This center is now the so called parent of the frames “asus_front_link” and “asus_back_link”. These two frames represent the point of origin from the two camera devices. The frames “asus_front_rgb_frame” and “asus_back_rgb_frame” belong to the RGB-vision of the cameras. The frames “asus_front_depth_frame” and “asus_back_depth_frame” belong to the infrared-vision of the cameras. This chapter does not elaborate on the frames “odom” and “map” because this is done in chapter 3.2.1 and chapter 3.2.6.

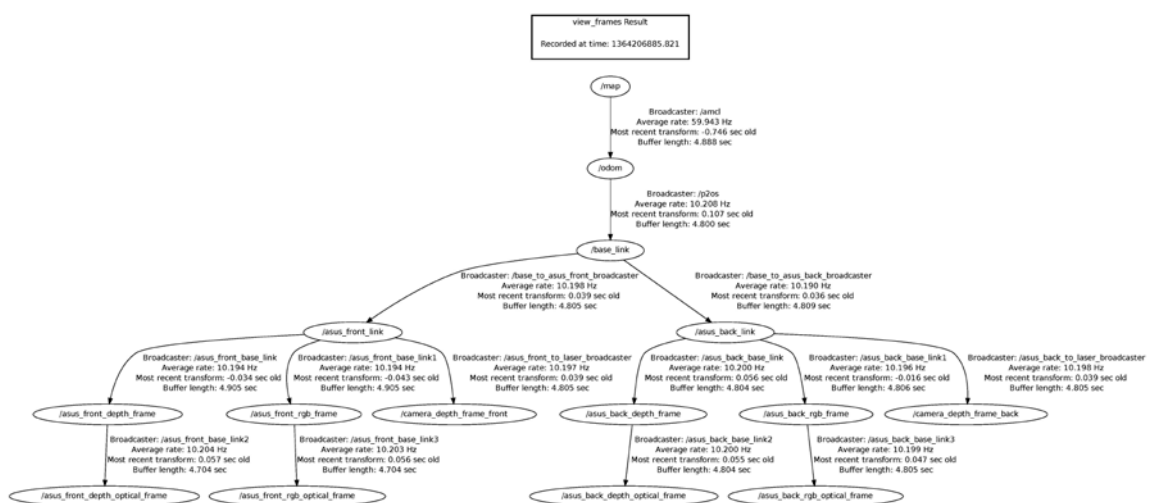


Figure 23: Hierarchic tree of the camera and laser scan link

For ROS it is necessary to define such dependencies, because they are needed for the correct computation of every device's origin of ordinates. In this chapter the parent-child relationships and how the dependencies to each other are defined are not described any further, because this is done in chapter 2.2.5. and chapter 3.2.5.

The following figure shows how the point cloud and the laser scan from both cameras look like. In the bottom left corner the video feedback of the infrared camera can be seen. The point cloud (red) was generated from this image. The computed laser scan (black) can also be seen in the picture.

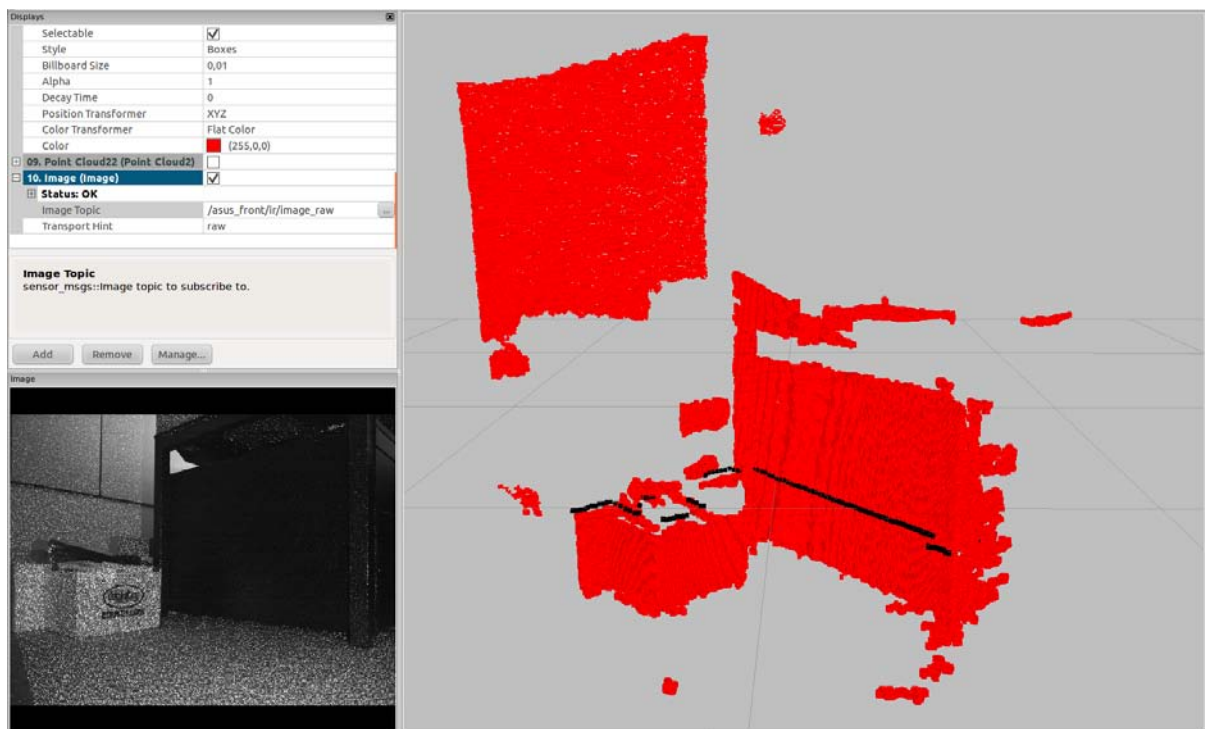


Figure 24: Example of a recorded point cloud (red) and the thereof created laser scan (black)

3.2.4 OpenNI camera

The OpenNI camera node uses the open source framework OpenNI [21] which acts as driver for the used camera. As cameras, two Asus Xtion Pros were used (figure 25). They provide an infrared and RGB image, as also the depth information of the recorded images. With the help of the depth information a point cloud can be generated. The images and the point cloud then can be used with ROS as they are published topics.



Figure 25: Appearance of the used camera for distance measurement (Source [17])

The OpenNI camera node gets called up twice because of the two cameras pointing forward and backward. The device id and their names are defined in the “main.launch” file when calling up the OpenNI launch file. From the original package nothing was altered and can be downloaded from the ROS website [22].

3.2.5 Static transform

As in chapter 2.2.5 is described that transforms are used to define parent-child relationships between the different components, here a static transform is used to define the relationship between the base frame, the two camera frames and the laser scan frames. The front camera frame is located 20 centimeters and the back camera frame is located -20 centimeters on the x axis but rotated 180 degrees (corresponds to roughly 3.1415 radian). As the point cloud of the camera is used to generate a laser scan, the frames of each laser scan is located at the same place as the camera frames. In the transformation tutorial on the ROS website it is explained in detail how to use a static transform in a launch [23]. The transformation tree for the cameras can also be seen in figure 23.

3.2.6 Map server and slam mapping

The map server offers map data as a ROS service. With this node a map can be saved or loaded. Before saving or loading a map, it has to be created with the help of the package slam gmapping.

3.2.6.1 Slam gmapping

Slam gmapping uses pose (here it is named odom) and laser data collected by the robot to create a 2D map like a floorplan. For the process of collecting the data, the user has to drive the robot manually through the room and scan everything. To get a reliable and accurate map the robot should not be moved around very fast but slow and smooth. The map created and used for this bachelor thesis can be seen in figure 26.

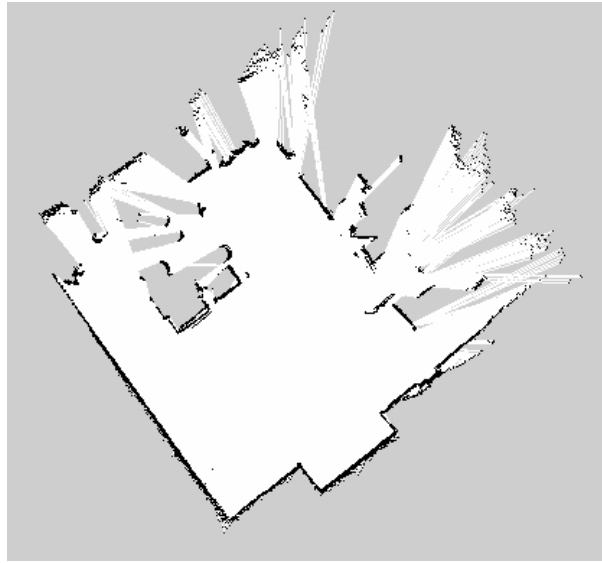


Figure 26: Appearance of the recorded saved and used map

3.2.6.2 Map server

After the map was created with the help of slam gmapping, it can be saved by a command line order. With this order, two files are generated. One contains the map itself as occupancy grid and the other the basic information like size and name of the map. When the map was saved successfully, the slam gmapping node is no longer needed and can be closed. To load a map, the map server node must be started, which is done in the “main.launch” file where also the location and name of the map has to be defined.

3.2.7 Costmap

This package implements a 2D costmap that takes in sensor data from the robot and builds an occupancy grid from the data. With the help of this occupancy grid, the robot now knows where to move and how close it can move towards a wall. Also suddenly appearing obstacles are registered and recorded. The user has to define some parameters like the robot’s safety radius or update rate. In this thesis all needed parameters are loaded when calling the move base node. For the full list of editable parameters used by this node visit the ROS website [24].

The following picture shows a generated costmap2D where the red cells represent obstacles, the blue cells represent obstacles inflated by the inscribed radius of the robot and the red polygon represents the footprint of the robot. To avoid any collision, the footprint of the robot should never intersect a red cell and the center of the robot should never touch a blue cell.

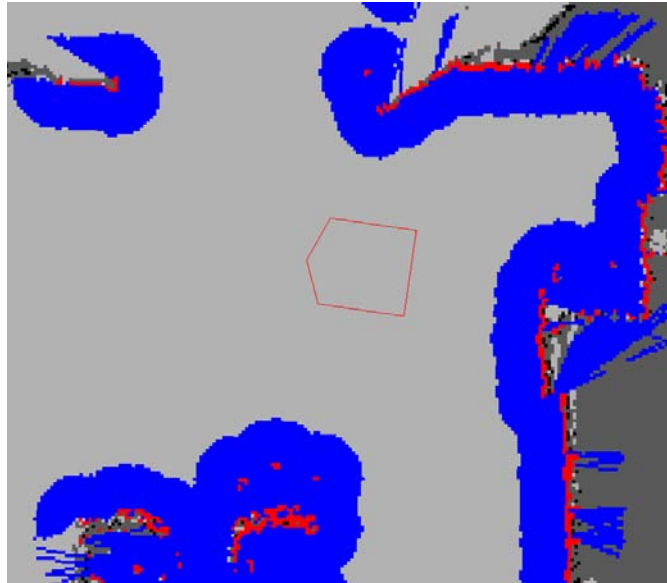
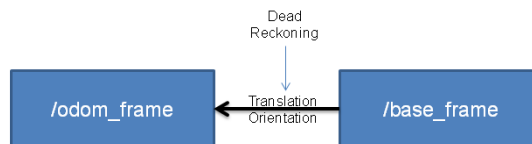


Figure 27: Example of the appearance of a map including the costmap package (Source [18])

3.2.8 AMCL

The AMCL package provides a localization system for robots moving in 2D. It computes the robot's current position in the map with the help of the laser scan and odometry data and creates a relationship between the base frame and the map frame. Without this package the current position of the robot on the map would not be known. The figure below shows the difference between localization using odometry and AMCL.

Odometry Localization



AMCL Map Localization

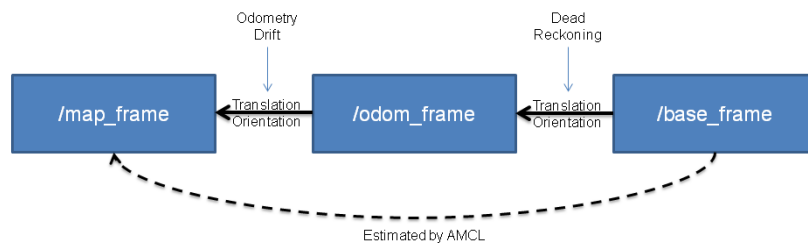


Figure 28: Example of the function of the AMCL package (Source [19])

For this bachelor thesis the following parameters were modified:

- Minimum allowed number of particles = 500
- Maximum error between the true and estimated distribution = 0.05
- Transform tolerance = 0.9

For the full list of editable parameters used by this node visit the ROS website [25].

3.2.9 Move base

This package provides action commands for sending, pausing or canceling a goal which the robot will attempt to reach. It uses the map, laser scan and odometry data to plan a path to the goal and sends then the corresponding velocity commands to the robot. With the help of costmap2D it also tries to avoid appearing obstacles and alters the calculated path in a way that the mobile platform drives around this obstacle and does not hit it. For correct path planning and obstacle avoidance a lot of parameters have to be adjusted accordingly. All these parameters are defined in a few different files, which are loaded in the “main.launch” file when calling the move base node. In these files also the parameters for the costmap2D are defined. For the full list of editable parameters used by this node visit the ROS website [26].

3.2.10 Asterics Pioneer Robot

This package is responsible for the movements of the robot itself and the katana arm. It gets the commands from the “Asterics_Pioneer_Network” node and processes them. Before starting this node the robot must stand on its initial position which is here defined as the battery station. Now when starting the node, the robot's position gets set correctly in the map and the localization, obstacle avoidance through costmap2D and the path planning should work correctly. After the position is set, the katana arm will move into its initial pose. When these two processes are completed the node waits for incoming commands. If a command arrives, the node checks if it is meant for the katana arm or the mobile platform and acts accordingly.

For example, when the user presses the button for the forward movement of the robot, the node “Asterics_Pioneer_Network” gets this command and forwards it to the “Asterics_Pioneer_Robot” node. This node then checks if it is meant for the mobile platform or for the katana arm. It jumps right in the mobile platform class and checks which command should be performed. As it is a command for forward movement, the node sends an

appropriate velocity command to the node “p2os_driver” which then drives the robot in the desired direction. If a katana movement command is coming in, the node sends the appropriate velocity command to the nodes responsible for the katana arm (explained in detail in chapter 3.2.2).

If desirable, an additional obstacle avoidance function with the use of the built in sonar device can be started. But therefore the user has to alter the program code a little bit so that the additional collision query gets also started.

3.2.11 Asterics Pioneer Network

This package is responsible to establish a network connection to the computer where the GUI is running. It acts as server and the other computer acts as client. When starting this node, it waits until a client connects. It has to be ensured that the client uses the right port to connect to, because on the server side, the port on which the client should connect to, is defined in the program code. When the client connects and no error occurred, the node is ready to receive commands. This command is then published as ROS topic to the node “Asterics_Pioneer_Robot” where it gets reprocessed. The “Asterics_Pioneer_Network” node is also able to send such a message to the client where it gets displayed in a text field. Therefore it checks every few milliseconds if there is something to send. For example if an obstacle occurs in front of the robot, the node can send a situation related message to the computer of the user where it gets displayed on the screen. The client side of the robot model is described in detail in chapter 3.1.

4 Results

4.1 Testing the robot-model by simulation

For simulation the program gazebo was used. It is a simulation tool with which every function of the robot can be tested, just like using a real robot. Figure 29 and figure 30 show the graphical interface of the simulation tool. With Gazebo it is possible to generate and edit complex simulation scenarios by simply dragging and dropping walls or objects. Such elements also can be inserted by writing a command into the console or terminal. It is also possible to record the movements for further analysis of the systems behavior

For executing the simulation, the ARE running the AsTeRICS-Pioneer model was started on a computer running the operating system Windows 7. The Gazebo simulator and the ROS program which receives the commands sent by the ARE were started on a laptop running the operating system Ubuntu. These two computers were connected over the network using a

WLAN-Router. When testing the whole program and every function, the robot and the katana arm moved as desired. For example when the button for closing the gripper was pressed on the on screen keyboard the gripper in gazebo closed in an instant.

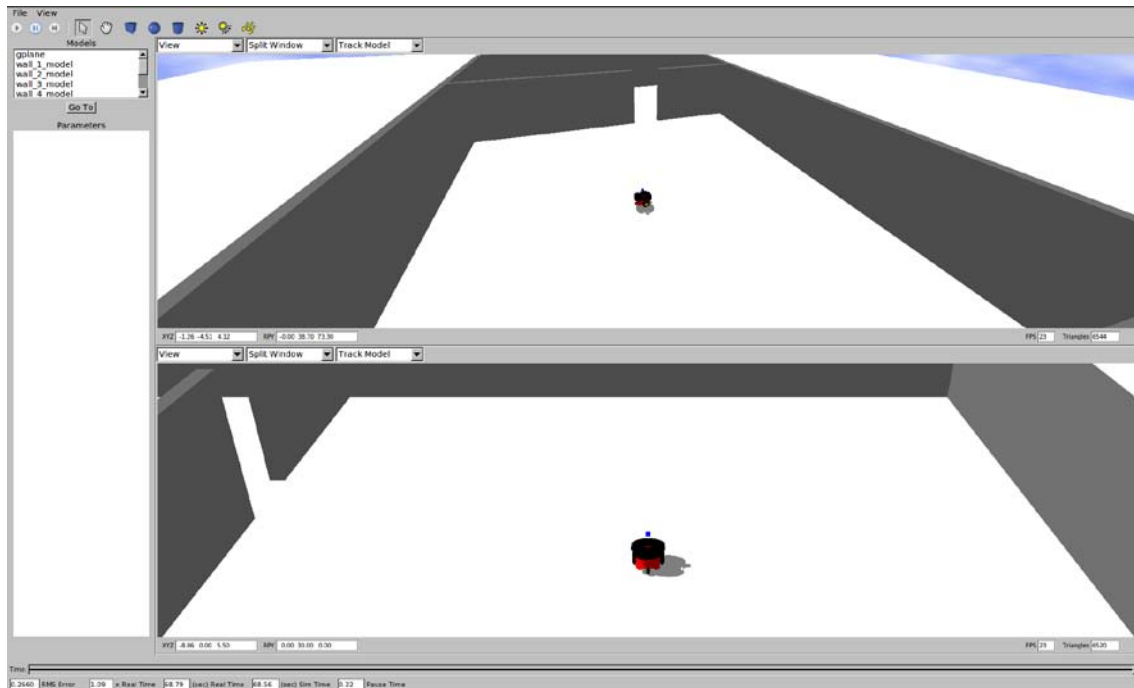


Figure 29: Graphical interface of the Gazebo simulation tool for the mobile platform

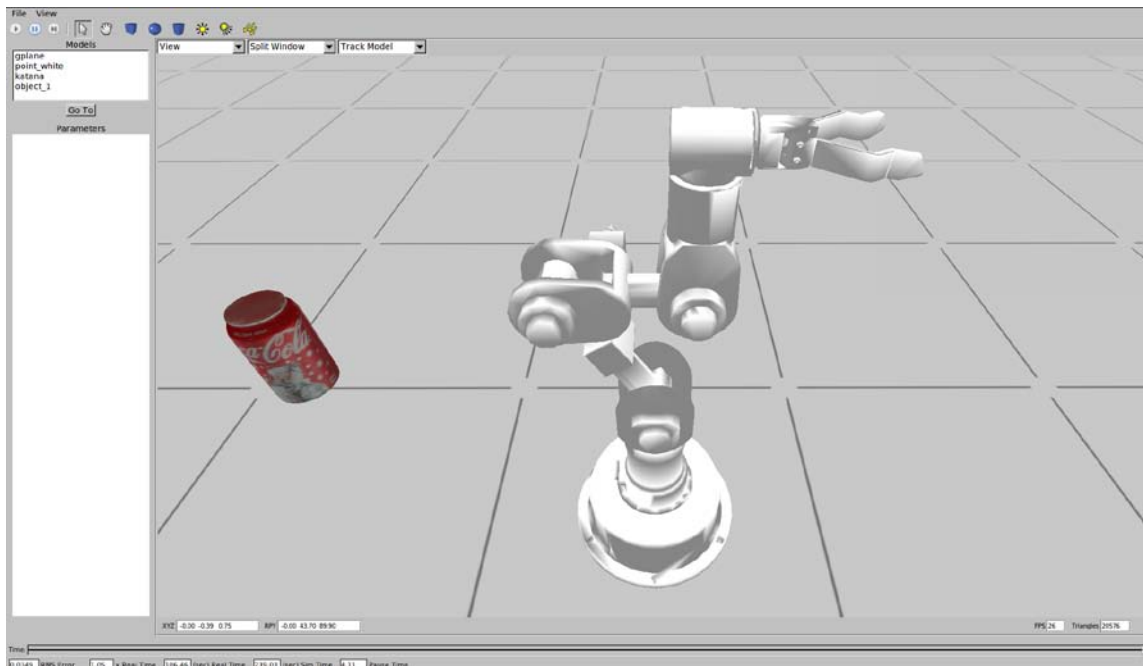


Figure 30: Graphical interface of the Gazebo simulation tool for the katana arm

4.2 Testing the robot-model with the Pioneer-p3at

Every function of the OSKA on-screen keyboard was tested and the robot and katana moved as desired. As the robots function were tested on the real platform, the katana functions were only tested in simulation because of technical difficulties with the katana arm. The computer running the GUI is connected to the robot via WLAN. The results showed that every function for moving the robot base works as desired. The only problem occurred when sending a katana movement task to the robot. The execution was delayed for a few seconds. Not only the katana movement was delayed, also the video feedback was delayed for about one second.

5 Discussion

As mentioned in section 4 (results), the robot moved according to the input given by the user. Also the automatic navigation worked as desired. The robot reached the desired destination without problems but when an obstacle is put in the path, it is avoided very scarce. The problem therefore could be that the parameters for obstacle avoidance and path planning are not defined that well. But a reconfiguration of these parameters should eliminate this problem. As also mentioned a big delay between pressing a katana movement button and the execution of this command is occurring. The problem therefore could be an insufficient ROS program where a mistake in the program code was overseen. Another big problem is the lag of the video feedback. But this problem cannot be solved as this lag occurs due to the analog digital and digital analog conversion. The lag could only be minimized or erased by sending the video data analogously.

6 Conclusion

This thesis made the first step for building a bridge from AsTeRICS to ROS by creating a robot user interface for people with severe motor disabilities. The model was extended step by step and so an easy to use graphical user interface was created. The complete robot platform can now be navigated easily by a person with reduced motor capabilities.

7 Further Challenges

As all functions are working separately without problems, the next step would be to observe and optimize the program and every parameter to get better performance of the different functions and when everything is run at the same time. Another challenge will be to find a better solution for the visual feedback. Also the control of the katana arm could be optimized

as the current solution is not that ideal for grabbing objects. After solving all these challenges, the robot model should work without problems.

List of Literature

- [1] Kompetenznetzwerk KI-I, Project AsTeRICS, Online at Internet URL: http://www.asterics.eu/fileadmin/user_upload/Documentation/UserManual.pdf (last accessed 12.1.2013)
- [2] Claro Interfaces, <http://www.clarointerfaces.com/category/oska-for-pc-access.php> (last accessed 12.1.2013)
- [3] <http://www.cogain.org/home> (last accessed 12.1.2013)
- [4] Agence France- Presse, IndustryWeek, Online at Internet URL: <http://www.industryweek.com/robotics/toyota-unveils-robot-designed-perform-household-chores> (last accessed 4.12.2012)
- [5] Der Standard, Online at Internet URL: <http://derstandard.at/1345166255878/Im-Umgang-mit-uns-muessen-Roboter-weich-werden> (last accessed 4.12.2012)
- [6] <http://www.wingsforlife.com/de-at/querschnittslaehmung/ursachen-folgen/> (last accessed 12.1.2013)
- [7] Kompetenznetzwerk KI-I, Project AsTeRICS, <http://www.asterics.eu/index.php?id=26> (last accessed 12.1.2013)
- [8] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng. - ROS: An open-source Robot Operating System, Stanford University, Willow Garage, University of Southern California
- [9] <http://www.ros.org/wiki/ROS/Tutorials/UnderstandingTopics> (last accessed 4.12.2012)
- [10] Eclipse Foundation, <http://www.eclipse.org/> (last accessed 12.1.2013)
- [11] <http://www.kdevelop.org/> (last accessed 12.1.2013)
- [12] www.ros.org/wiki/actionlib (last accessed 12.5.2013)
- [13] <http://mainland.cctt.org/istf2006/sonar.asp> (last accessed 12.5.2013)
- [14] <http://www.willowgarage.com/blog/2011/02/28/nist-and-willow-garage-solutions-perception-challenge> (last accessed 12.5.2013)
- [15] http://home.roboticlab.eu/de/examples/sensor/ir_distance (last accessed 12.5.2013)
- [16] <http://www.dma.ufg.ac.at/app/link/Grundlagen%3A3D-Grafik/module/14174> (last accessed 12.5.2013)
- [17] http://www.asus.de/Multimedia/Motion_Sensor/Xtion_PRO/ (last accessed 12.5.2013)
- [18] http://www.ros.org/wiki/costmap_2d?action=AttachFile&do=get&target=costmap_rviz.png (last accessed 12.5.2013)
- [19] <http://www.ros.org/wiki/amcl> (last accessed 12.5.2013)
- [20] http://www.ros.org/wiki/p2os_driver (last accessed 12.5.2013)
- [21] <http://www.openni.org/> (last accessed 12.5.2013)
- [22] http://www.ros.org/wiki/openni_kinect (last accessed 12.5.2013)

- [23]** http://www.ros.org/wiki/tf#static_transform_publisher (*last accessed 12.5.2013*)
- [24]** http://www.ros.org/wiki/costmap_2d (*last accessed 12.5.2013*)
- [25]** <http://www.ros.org/wiki/amcl> (*last accessed 12.5.2013*)
- [26]** http://www.ros.org/wiki/move_base (*last accessed 12.5.2013*)

Table of figures

Figure 1: Concept of the AsTeRICS system (Source [1], S. 6)	11
Figure 2: Graphical User Interface of the ARE (Source [1], S. 14).....	12
Figure 3: Graphical User Interface of the AsTeRICS Configuration Suite (Source [1], S. 16)	13
Figure 4: Example of a ROS program with different nodes and topics (Source [9])	14
Figure 5: ActionServer and ActionClient communication scheme (Source [12])	16
Figure 6: Example of a launch file for converting a point cloud into a laser scan	17
Figure 7: Principle of how sonar works (Source [13]).....	18
Figure 8: Example of a recorded point cloud (Source [14])	19
Figure 9: Principle of infrared distance measurement (Source [15]).....	19
Figure 10: Forward kinematic scheme (Source [16])	20
Figure 11: Inverse kinematic scheme (Source [16])	21
Figure 12: Appearance and current setup of the Pioneer3-at robot	22
Figure 13: Use-case scenario using one button switches for navigation the robot.....	23
Figure 14: Use-case scenario using a webcam for navigating the robot	23
Figure 15: Model setup of controlling the pioneer with the OSKA plugin.....	25
Figure 16: Runtime example of the robot model.....	25
Figure 20: Example of how the P2OS driver works.....	29
Figure 21: Appearance of the katana arm.....	30
Figure 22: Hierarchic tree of all links of the katana arm parts	31
Figure 23: Hierarchic tree of the camera and laser scan link	32
Figure 24: Example of a recorded point cloud (red) and the thereof created laser scan (black)	33
Figure 25: Appearance of the used camera for distance measurement (Source [17])	34
Figure 26: Appearance of the recorded saved and used map	35
Figure 27: Example of the appearance of a map including the costmap package (Source [18])	36
Figure 28: Example of the function of the AMCL package (Source [19]).....	36
Figure 29: Graphical interface of the Gazebo simulation tool for the mobile platform	39
Figure 30: Graphical interface of the Gazebo simulation tool for the katana arm	39

List of abbreviations

AAL	Ambient Assisted Living
ACS	AsTeRICS Configuration Suite
ARE	AsTeRICS Runtime Environment
GUI	Graphical User Interface
HMI	Human Machine Interface
IP	Internet Protocol
OSKA	On Screen Keyboard Application
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
TCP	Transmission Control Protocol
WLAN	Wireless Local Area Network
WWW	World Wide Web