Comenius University in Bratislava Faculty of Mathematics, Physics and Informatics

Robotour with Laser Range Sensor

Diploma Thesis

Bratislava, 2017

Bc. Jozef Dúc

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Robotour with Laser Range Sensor

Diploma Thesis

Study Programme:	Applied Informatics
Field of Study:	2511 Applied Informatics
Department:	Department of Applied Informatics
Supervisor:	Mgr. Pavel Petrovič, PhD.

Bratislava, 2017

Bc. Jozef Dúc

16208072

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Jozef Dúc
Study programme:	Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	9.2.9. Applied Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Robotour with Laser Range Sensor

Aim: The thesis builds on two previous diploma theses dedicated to experiments with navigation of a mobile robot based on visual input processed by an artificial neural network. The outcome was a complete system capable of participation in an international robotics competition Robotour. The aim is to continue the project with several useful improvements including integration of spatial data from a laser range sensor. Student will investigate various algorithms, perform real-world experiments, and create a system with improved performance capable of sensor fusion of image data, gps, compass, map information and the laser range sensor.

Literature: Nadhajsky M., Petrovic P. (2010) Robotour solution as a learned behavior based on Artificial Neural Networks, RIE 2010, Bratislava.
Moravčík M. (2015): Autonómny mobilný robot pre súťaž Robotour, diploma thesis, Faculty of Mathematics, Physics and Informatics, Bratislava.
Nadhajský M. (2010): Robotour, diploma thesis, Faculty of Mathematics, Physics and Informatics, Bratislava.

Keywords: mobile robot, robotour, navigation, laser range sensor

Supervisor:	Mgr. Pavel Petrovič, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	29.09.2015
Approved:	09.12.2015

prof. RNDr. Roman Ďurikovič, PhD.
Guarantor of Study Programme

[illegible]
Student

Pavel Petrovič
Supervisor

16208072

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Jozef Dúc
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.9. aplikovaná informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Robotour with Laser Range Sensor
Robotour s laserovým senzorom na meranie vzdialenosti

Cieľ: Diplomová práca nadväzuje na dve predchádzajúce, ktoré boli zamerané na experimenty s navigáciou mobilného robota založenej na vizuálnom vstupe spracovávanom umelou neurónovou sieťou. Výsledkom bol úplný systém, ktorý bol schopný zúčastniť sa medzinárodnej robotickej súťaže Robotour. Cieľom práce je pokračovať v projekte pridaním niekoľkých užitočných rozšírení vrátane zaintegrovania modulu na spracovanie priestorových dát získaných z laserového senzora na meranie vzdialenosti. Študent preskúma rozličné algoritmy, uskutoční experimenty v reálnom prostredí a vytvorí systém s lepšími vlastnosťami, ktorý bude schopný kombinovať obrazové údaje s gps, kompasom, mapou a informáciami z laserového senzora na meranie vzdialenosti.

Literatúra: Nadhajsky M., Petrovic P. (2010) Robotour solution as a learned behavior based on Artificial Neural Networks, RIE 2010, Bratislava.
Moravčík M. (2015): Autonómny mobilný robot pre súťaž Robotour, diplomová práca, Fakulta matematiky, fyziky a informatiky, Bratislava.
Nadhajský M. (2010): Robotour, diplomová práca, Fakulta matematiky, fyziky a informatiky, Bratislava.

Kľúčové slová: mobile robot, robotour, navigation, laser range sensor

Vedúci:	Mgr. Pavel Petrovič, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, Dr.
Dátum zadania:	29.09.2015

Dátum schválenia: 09.12.2015

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

[illegible]
študent

Pavel Petrovič
vedúci práce

I confirm that this Diploma thesis is my own work and I have documented all sources and material used.

. . . .

I would like to thank my advisor, Mgr. Pavel Petrovič, PhD. for his guidance and valuable pieces of advice.

Abstract

In this diploma thesis, we work with an autonomous robot Smely Zajko. The robot was built in previous works at Comenius University; we focus on integrating a new sensor - laser range sensor into the set of existing sensors.

Based on our analysis of original state of the robot, we change the architecture of its system. We reimplement original code into ROS framework and create a more flexible system that allows running multiple independent experiments, record and replay data from log files and it is easy portable to another robot as well.

We propose a new algorithm for reactive obstacle detection and avoidance using the laser range sensor. We tested our implementation several times in indoor and outdoor environment. We also took part in the international robotic contest Robotour where we won the third place. The results of our testing show that the robot is now able to detect a more suitable path with our algorithm mainly because of the probabilistic approach.

Key words: mobile robot, robotour, navigation, laser range sensor

Abstrakt

V tejto diplomovej práci pracujeme s autonómnym robotom Smelý Zajko. Robot bol postavený v rámci predchádzajúcich prác na Univerzite Komenského; my sa sústredíme na integrovanie nového laserového senzora medzi ostatné už existujúce senzory.

Na základe analýzy pôvodného stavu robota meníme architektúru systému. Implementujeme pôvodný kód do frameworku ROS a vytvárame flexibilnejší systém, ktorý nám umožňuje spúšťať niekoľko nezávislých experimentov, nahrávať a prehrávať dáta zo senzorov a taktiež je ho možné jednoducho preniesť na iného robota.

Navrhujeme nový algoritmus na reaktívnu detekciu prekážok s využitím laserového senzora. Našu implementáciu sme niekoľkokrát testovali v interiéri aj exteriéri. Taktiež sme sa zúčastnili medzinárodnej robotickej súťaže Robotour, kde sme získali tretie miesto. Výsledky testovania ukazujú, že robot je teraz lepšie schopný detekovať vhodnejšie cesty a to hlavne vďaka pravdepodobnostnému prístupu, ktorý sme zvolili.

Kľúčové slová: mobilný robot, robotour, navigácia, laser range senzor

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

An autonomous car or self-driving car is a vehicle that is capable of sensing its environment and navigating without human input. Companies like Google, Uber, Tesla have big projects in this area that attract attention of media and society. Many people believe that self-driving cars are the future of car industry and that they completely change the way how we travel.

Though many such vehicles are being developed, automated cars permitted on public roads are not yet fully autonomous. They all require a human driver at the wheel who is ready at a moment's notice to take control of the vehicle.

Research in this area is of the big importance. Many people try to find new approaches and algorithms how to face new challenges and improve self-driving cars.

In this thesis, we use autonomous robot called Smely Zajko (Brave Bunny). Smely Zajko was built in 2011 by Miroslav Nadhájsky [MN10] as a part of his diploma thesis. Further improvements were made by Michal Moravčík in 2015 [Mor15].

1.2 Goal

Both, Nadhájsky and Moravčík took part in a Czech competition Robotour. In this competition, autonomous robots compete in delivering a load in a park. They are allowed use only the paths in the park.

Even though Smely Zajko regularly attends robotic competitions like Robotour, there is still a large space for improvements. The main goal of this work is to integrate a new sensor: laser range sensor, change the architecture of the software and try new algorithms in order to obtain better results.

Chapter 2

Preliminaries

The main aim of this chapter is to introduce a reader basic terms which are used in this thesis. At first, we provide a general overview of artificial intelligence, neural networks and robotics. Later, we present a laser sensor which we use in our thesis and we introduce a robotic contest Robotour.

2.1 Artificial Intelligence

Artificial Intelligence is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

Nowadays, there are many branches of artificial intelligence:

- **Logical** - what a program knows about the world in general, the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language.
- **Search** - artificial intelligence often examine a large number of possibilities, e.g. moves in a chess game or inferences by a theorem proving program.
- **Pattern recognition** - when a program makes an observation of something, it is often created to compare what it sees with some pattern. There are many types of pattern recognition in computer vision for example detection of people or specific type of objects in images.

This list surely miss some branches, because no-one has identified every single branch of artificial intelligence yet.

2.2 Neural Networks

Artificial neural networks are structures, which are inspired by biological neural networks. Neural networks are based on the mesh of *neurons*. Thanks to that we can solve various number of complicated tasks e.g. image recognition, classification of data, approximation of functions or in artificial intelligence.

2.2.1 Neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network [Hay98]. Perceptron is a machine learning algorithm that helps provide classified outcomes for computing. It dates back to the 1950s and represents a fundamental example of how machine learning algorithms work to develop data. Perceptron is getting n inputs x_i and each input has own weight w_i. The result of perceptrons is showed in equation 2.1.

$$v = \sum_{i=1}^{n} x_i * w_i$$

Figure 2.1: Equation for perceptron

For better result, we can add another not zero fixed input parameter which is called *bias*. Without bias result of our equation is influenced by activation function a lot. After adding bias our equation looks like in Figure 2.2

$$v = \sum_{i=1}^{n} x_i * w_i + b_k$$

Figure 2.2: Equation for perceptron with bias

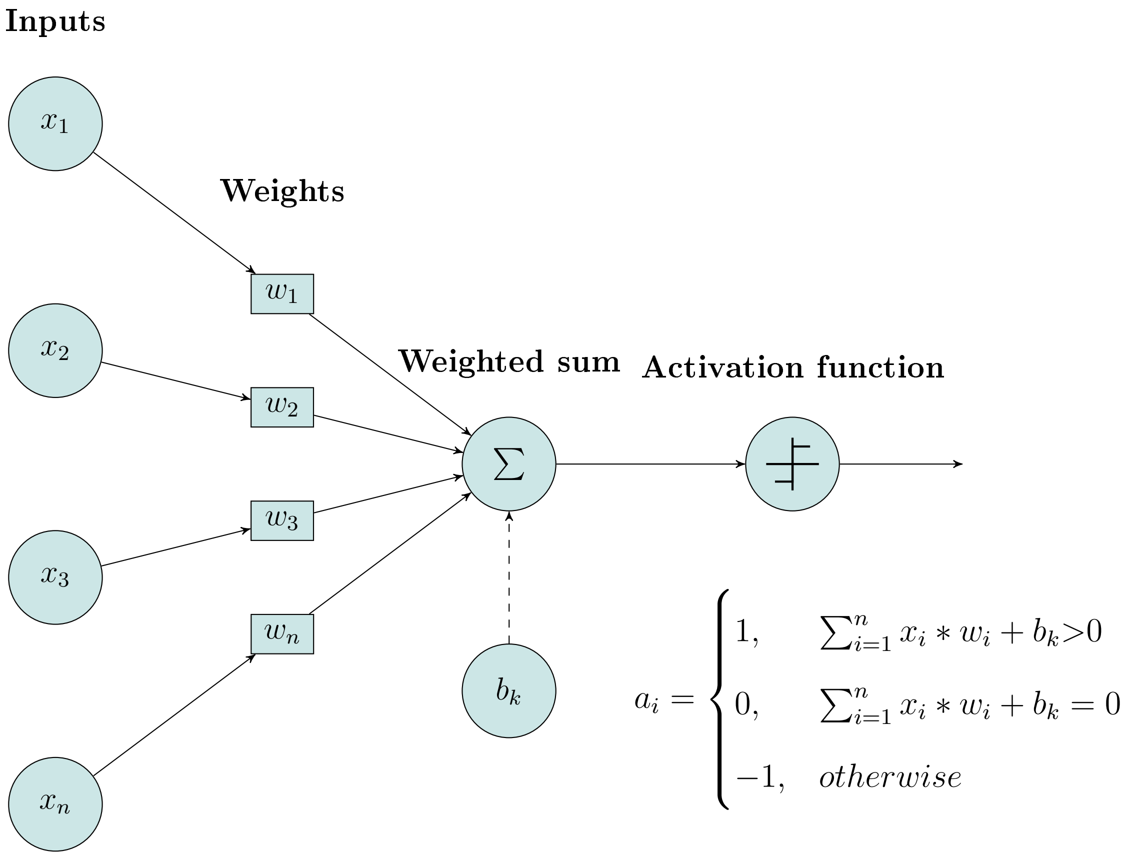
Now we must apply activation function to our sum with bias. Activation function is mathematical formalism to approximate the influence of an extracellular field on an axon or neurons [Rat86]. After applying activation function to the equation we get 2.3.

The basic flow of using previous steps is in the next figure 2.4. As an activation function, we use signum function [Hay98]. For the corresponding form of a sigmoid

$$a_i = \varphi(\sum_{i=1}^{n} x_i * w_i + b_k)$$

Figure 2.3: Equation after applying activation function

function, we may use the hyperbolic tangent function.



$$a_i = \begin{cases} 1, & \sum_{i=1}^{n} x_i * w_i + b_k > 0 \\ 0, & \sum_{i=1}^{n} x_i * w_i + b_k = 0 \\ -1, & otherwise \end{cases}$$

Figure 2.4: Basic flow of perceptron

2.2.2 Multilayer perceptron

Multilayer perceptron (MLP) is a feedforward multi-layered neural network consisting of input neurons that receive the input signal, which is propagated forward through the network to produce the activations of the output layer. The hidden neurons are organised in disjoints layers that are propagated one at a time [Hay98]. Multilayer perceptrons are commonly used with the error backpropagation algorithm introduced by [Rum86]. It is based on the concept of propagation of the signal through the network in order to find the activation of the output layer. Basic signal-flow graph of an MLP is in the next figure 2.5.

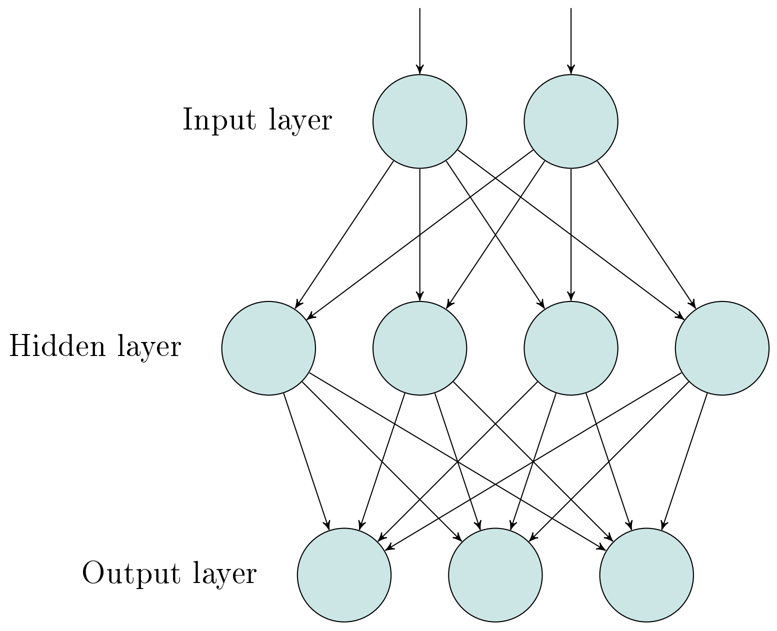


Figure 2.5: Signal-flow graph of an MLP

2.2.3 Learning

Learning can be broadly defined as a computational method using the experience to improve the performance or to make accurate predictions. It is not good to "hardcode" agent behaviour sometimes when we do not know how robot or program should react. If the environment is not known, learning is the only way to "program" him. We know few types of learning.

- Supervised learning
- Unsupervised learning
- Reinforcement learning

2.2.3.1 Supervised learning

The learner receives a set of labelled examples as training data and makes predictions for all unseen points. This is the most common scenario associated with classification, regression, and ranking problems. [MM12]

2.2.3.2 Unsupervised learning

The learner exclusively receives unlabelled training data and makes predictions for all unseen points. Since in general no labelled example is available in that setting, it can be difficult to quantitatively evaluate the performance of a learner. Clustering and dimensionality reduction are examples of unsupervised learning problems.

2.2.3.3 Reinforcement learning

The training and testing phases are also intermixed in reinforcement learning. To collect information, the learner actively interacts with the environment and in some cases affects the environment, and receives an immediate reward for each action. The object of the learner is to maximize his reward over a course of actions and iterations with the environment.

2.2.4 Backpropagation

Backpropagation is an algorithm which is propagating an error through the whole neural network [Hay98]. It is usually considered to be a supervised learning.

2.2.5 Rprop

Rprop algorithm [MR93] is based on traditional backpropagation algorithm. Size of weight change is not based on derivation of error but it changes by the previous result in last iteration.

2.3 Robotics

Robotics is a branch of engineering that involves the conception, design, manufacture, and operation of robots. This field overlaps with electronics, computer science, artificial intelligence, mechatronics, nanotechnology and bioengineering.

Science-fiction author Isaac Asimov is often given credit for being the first person to use the term robotics in a short story composed in the 1940s. In the story, Asimov suggested three principles to guide the behaviour of robots and smart machines. Asimov's Three Laws of Robotics [Asi41], as they are called, have survived to the present.

1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Nowadays robots are used more and more in our lives. We can use robots in medicine for performing low-invasive surgery, industry for lifting heavy loads or even in the sport for entertainment. Many robots are developed to do jobs that are hazardous to people for example finding survivors in ruins, defusing bombs, exploring dangerous locations.

2.4 LiDAR

LiDAR is device which is used for making depth scans of targets. Laser introduces a laser beam into the environments and measures the time of flight of the signal to return. The same principle is used as in ultrasonics sensors, unlike a sonar bean has very wide range, a laser produces an almost infinitesimal field of view. If the laser is directed in a scan, it can produce depth values. There are several major components to a lidar system:

- Laser
- Scanner and optics
- Photodetector and receiver electronics
- Position and navigation systems

Lidar has a wide range of applications which can be divided into two types.

- Airborne type involves the acquisition of three-dimensional ground point data by utilising a laser scanning device operating at frequencies in the near infra-red. The scan direction is orthogonal to the flight path. The unit is mounted in a fixed or rotary winged aircraft flying at altitudes of up to 1000 m.

- Terrestrial type is almost the same as airborne type, except that it is ground based. Locating the scanner on the ground gives some distinct advantages for capturing discrete objects from multiple angles [PR06].

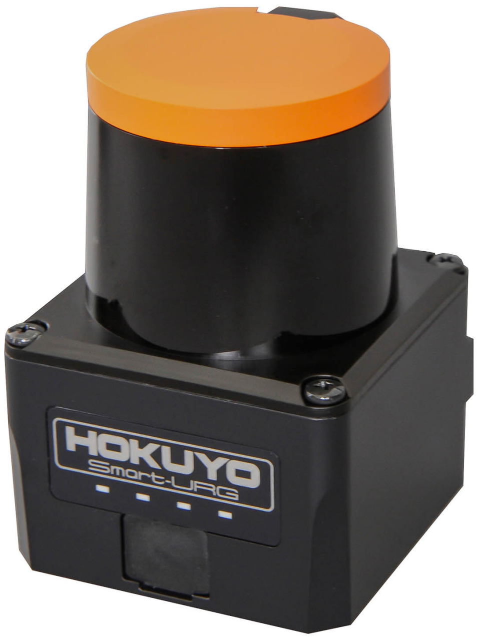


Figure 2.6: Hokuyo LiDAR sensor which is used in our work.

2.5 Robotour Contest

Robotour is an international robotic competition, which is held every year since 2006. In this competition, autonomous robots compete in delivering the load, usually a beer can. They are allowed only to use paths in the park and they have to avoid obstacles. All the competition rules are summarised below.

Competition rules

- Bring load to the finish. Usually 5 litres beer can.
- The robot can move only in marked lines in the park.
- The robot can not collide with an obstacle.
- The robot has to have a big red button to immediate stop.

- The robot can have maps only from one common server - Open Street Maps.
- The robot can move only on road.

We note that we took part in this competition in September 2016 in Deggendorf, Germany and won the third place.

Chapter 3

Smely Zajko

In this chapter, we explain and describe an integration of hardware and software of robot *Smely Zajko (Brave Bunny).* Our work is based on previous works by Miroslav Nadhajský [MN10] (2010) and Michal Moravčík [Mor15] (2015). In figure 3.1, we can see the actual built of the robot as it was used in the competition.

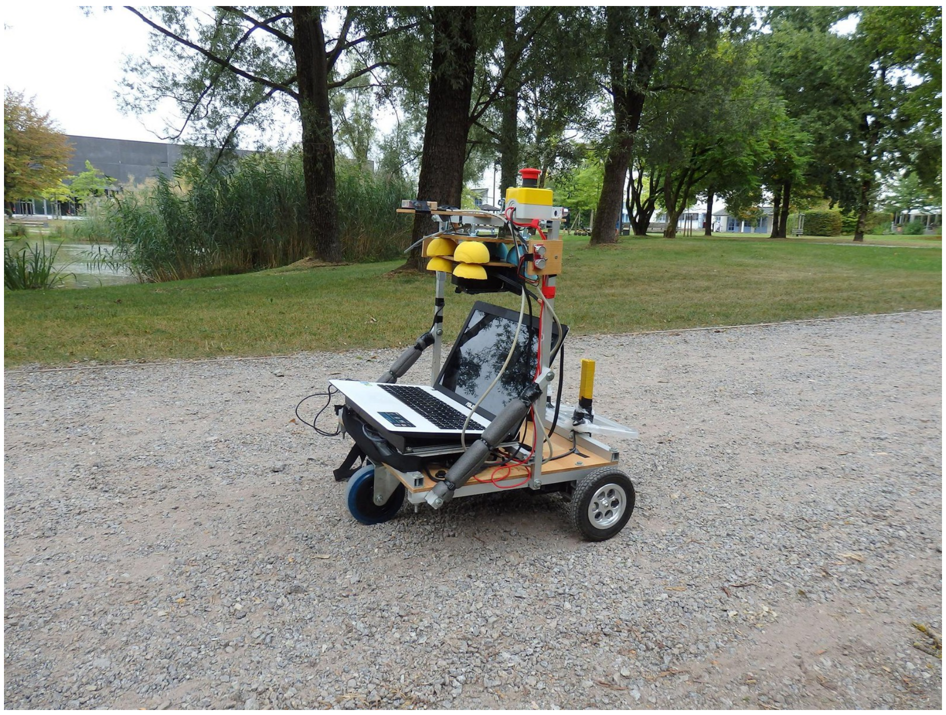


Figure 3.1: Robot *Smely Zajko* as we used it in Robotour 2016 competition.

At first, we describe the hardware parts of the robot and implemented software. At the end of the chapter, we analyse this solution so that we can better define the part we need to improve.

3.1 Hardware

We a use robot which was built in previous works from Miroslav Nadhajský [MN10] and Michal Moravčík [Mor15]. The core of the robot is almost the same, but we added a new laser range sensor which is the crucial part and the main topic of this thesis.

3.1.1 Sensors

The robot has some sensors which are placed on the top and on the sides of the robot: Laser range sensor, GPS, Ultrasonic Rangefinder Sensor, Compass and Camera. These sensors allow the robot to "see" world around it.

3.1.2 Laser range Sensor

As we mentioned in previous section 2.4, we use *Hokuyo* laser range sensor in our robot. This sensor allows us to identify obstacles on the road and also helps us to find the correct path. We can see basic sensor structure in Figure 3.2.

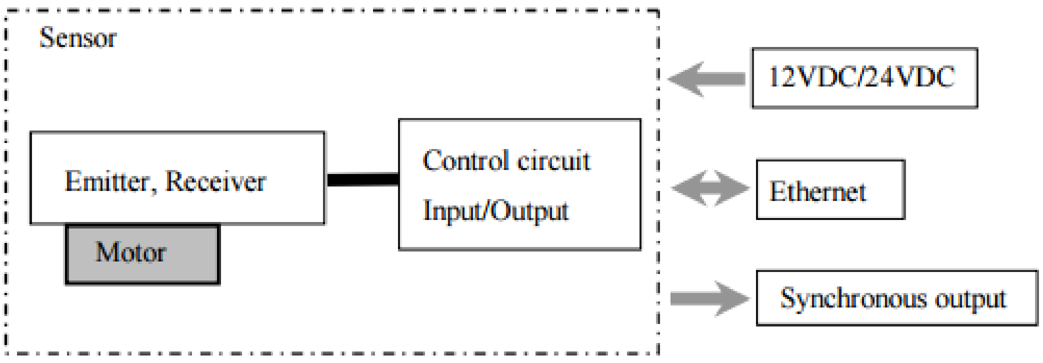


Figure 3.2: Structure diagram of Hokuyo LiDAR sensor.

The sensor is communicating directly with a computer attached to the robot via ethernet cable. This sensor use a laser to scan 270° field of view (we filter input to only 180° field of view, because we are not interested in backward directions). Positions of the obstacles in the range are calculated with step angle and distance. We can see, how the laser scanning image works in Figure 3.3.

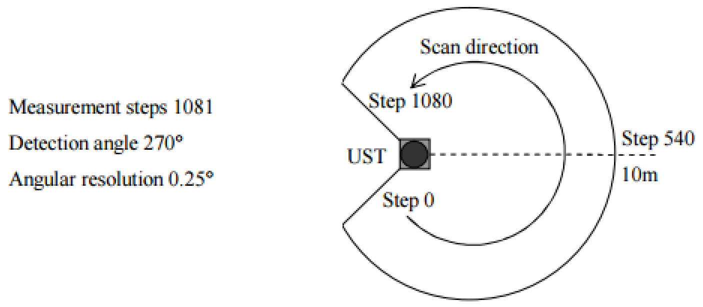


Figure 3.3: Laser scanning diagram.

3.1.3 GPS

We use *Navilock NL-302U* for GPS data. The device has RS-232 port as output, which is converted to USB through FTDI chip.

3.1.4 Compass

For orientation, we use module *SparkFun HMC6343* which has 3 axis gyroscope and accelerometer. We want to use a gyroscope and an accelerometer for later work with more precious road mapping and detection.

3.1.5 Ultrasonic Rangefinder Sensor

We use five ultrasonic sensors *Devantech SRF08* which are used to detect obstacles with 3cm - 6cm range and force robot to stop immediately. Communication with the ultrasonic rangefinder sensors is done via the I2C bus.

Ultrasonic sensors emit acoustic energy into environment and measure time of flight of the signal to return. The beam pattern of our ultrasonic sensors is conical with the width of the beam being a function of the surface area of the transducers and is fixed. The beam pattern of sensors is shown in 3.4.

3.1.6 Camera

For road recognition, we use an ordinary camera. We take as many images as possible from the camera, push them into the neural network and try to recognize road. We use old trained neural network from previous work from Moravcik [Mor15]. One of our

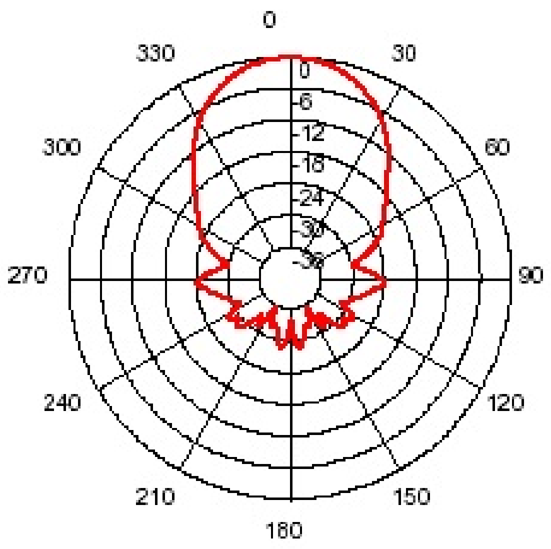


Figure 3.4: Beam which is emmited by ultrasonic sensor.

goals is to verify results from his network. This network has some problems because it identifies grass and road, but when robot reaches some road without grass on sides, it has problems with road following.

3.2 Software

In this section, we introduce the software part of robot that was done before and describe the technologies used in this project.

3.2.1 Code on SBot board

The code runs on SBot board which is based on AVR ATmega128 Microcontroller. Code communicates with our main code core base via buffers which are sent from board to main computer.

This code takes care for obstacles, steering and for status reporting also. It is useful to have this code on the low level because we do not have to care about it on the high level.

3.2.2 Custom made core

The core software in robot was built in C++. In our work, we want to generalize and rewrite this core to the *Robot Operating System* 4.1. Rewriting robot core to the specialized framework allow us to modularize code base, avoid bugs and provides more flexible components management in the future.

3.2.3 FANN

For path recognition, we use FANN library [Nis03]. This library uses the *iRprop* algorithm for learning. The library is very suitable for out purpose because it is really quickly process training data. FANN adds biases to every layer automatically so usage of the library is easier.

3.3 Previous work analysis

In this section, we analyse the previous state of the whole robot. We discuss hardware and software parts of the robot and some bottlenecks in implementation. We describe the main problems we had with the original state of the robot and suggest further improvements.

3.3.1 Hardware

When we tested the robot for the first time, there were not so many problems with the hardware part. The only thing we decided to redesign was *Hokuyo sensor*. The original position of the sensor was on the top of the robot as you can see in the Figure 3.5.

The top position was not so good because of a negative detection of obstacle and curb. As we can see in the picture 3.7 the third situation, robot sometimes can create a negative obstacle. We had the similar experiences as Murphy exaplains in his book [Mur00]. The laser range sensor produced similar negative obstacles.

That is why we decided to move *Hokuyo sensor* from the top to the bottom of the robot. This relocation seemed to be a good idea because now we can recognize obstacles which are smaller and located further.

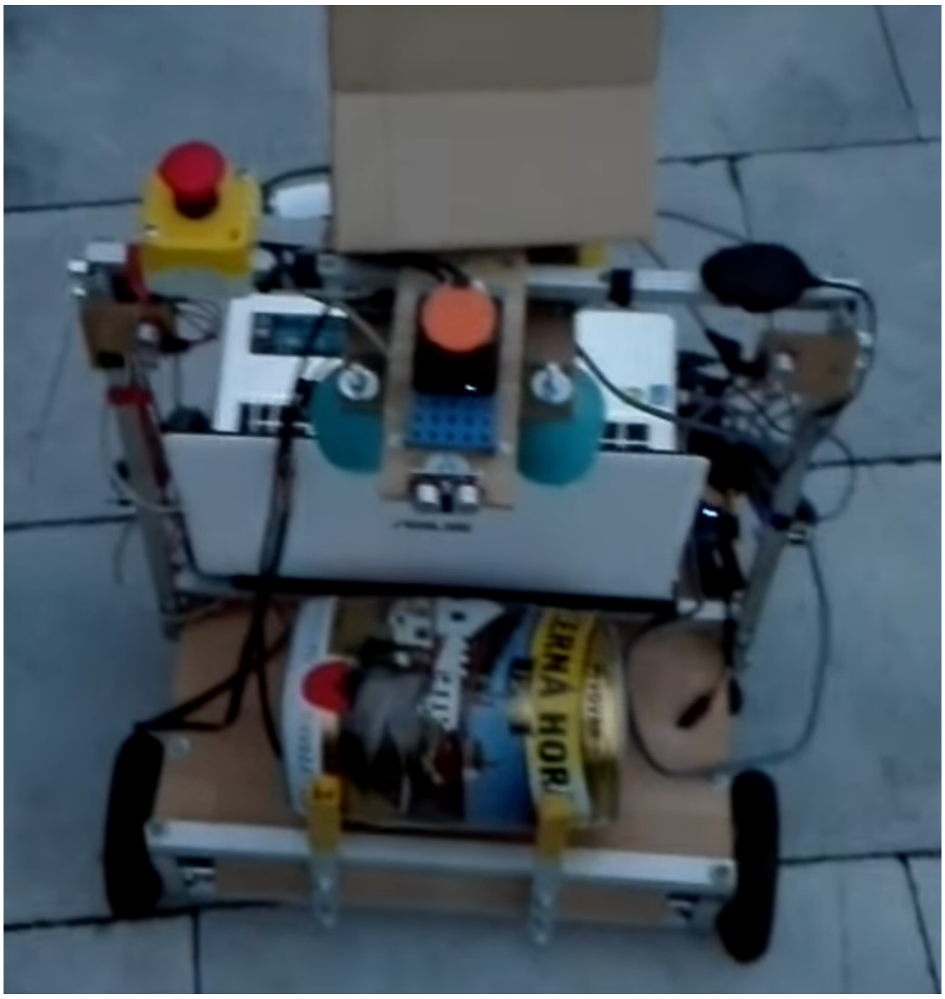


Figure 3.5: Hokuyo sensor on the top of the robot

However, during the competition in 2016, there was heavy rain and the sensor was influenced by it. The sensor recognised rain drops, which were bounced off the ground, as obstacles.

So we concluded that we should create some kind of cover from the bottom to protect *Hokuyo* in next competition.

3.3.2 Software

The software part of *Smely Zajac* was in very poor quality. There were many parts which had low cohesion (a class did a lot of jobs that do not have much in common) and very high coupling (a class has a lot of dependencies) that caused some performance problems. We point out some of these problems:

- Due to synchronous architecture in the main function, there was a problem with

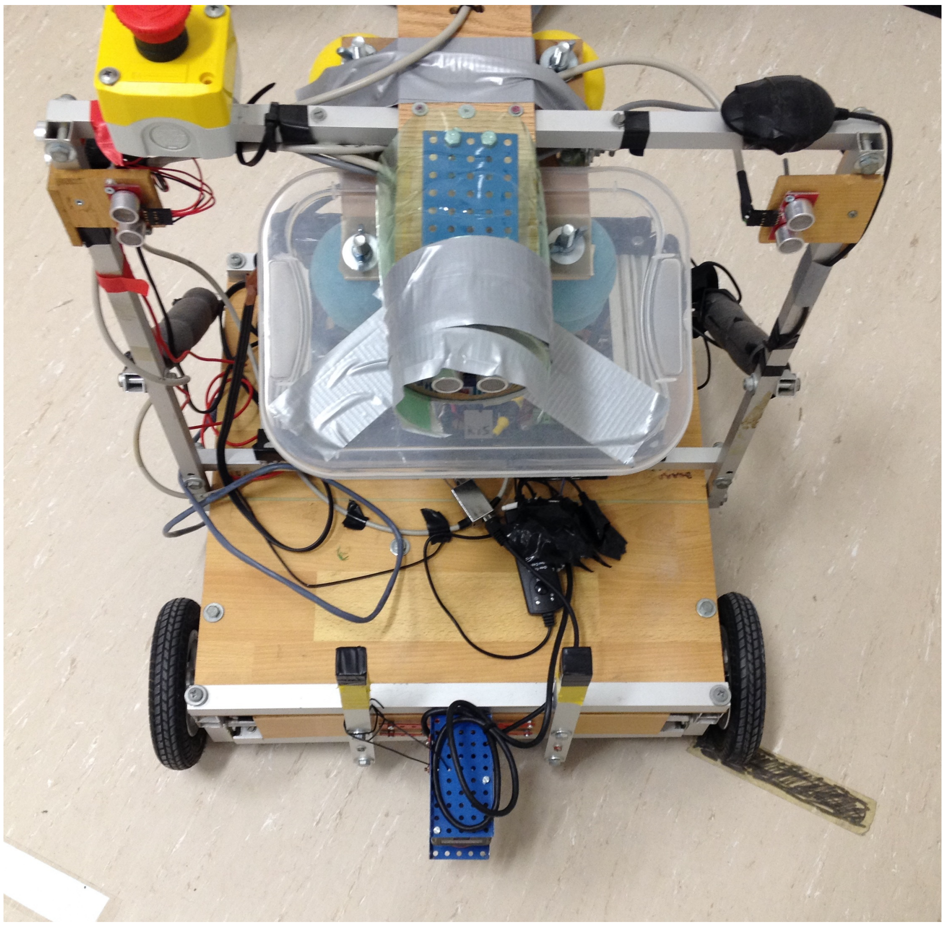


Figure 3.6: Hokuyo sensor on the bottom of the robot

evaluating of data. The program waited for data from camera but also skipped two frames of three (which had significant impact on performance and logging data from other sensors), then evaluated everything (neural network, direction, localization and planning algorithm), render windows with data and finally sent calculated direction to the robot. We provide pseudo-code of this synchronous architecture in algorithm 1.

- We wanted to implement new obstacle avoidance algorithm into the existing set of algorithms. It was almost impossible to implement it due to many different constants which combined data from algorithms, neural network and sensors in some strange way.

- Classes for reading data from sensors had low cohesion for example class for reading data from Hokuyo also rendered window.

That is why we decided to completely redesign software architecture including the

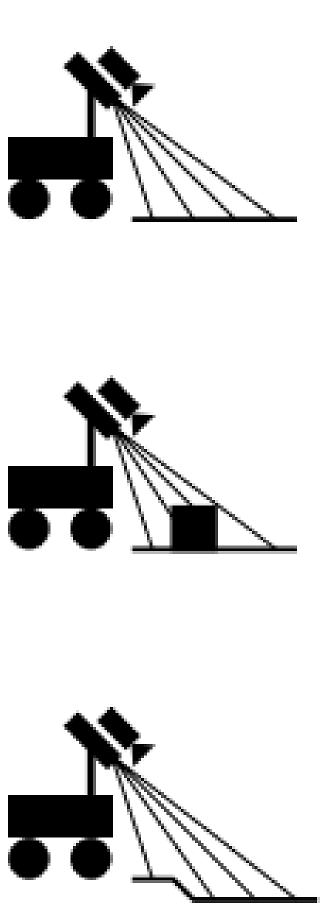


Figure 3.7: Situations and resulting images when sensor is located on the top

used technologies. We describe it in the next chapter in details.

```
Function main()
begin
    while 1 do
        if every third frame from camera then
            write data to log files;
            evaluate direction;
            render windows;
        end
    end
end
```

Algorithm 1: Previous synchronous architecure

Chapter 4

New approaches

After the competition, we decided to rewrite *Smely Zajko* software part to *ROS* framework due to reasons, which we mentioned in the previous chapter. Some parts of this chapter are based on two sources - official ROS documentation [ros] and the book [AM13]. At first, we describe this framework, further, we mention obstacle avoidance algorithms which can be used with our sensor.

4.1 Robot Operating System

ROS is the Robot Operating System framework, which is used nowadays by hundreds of research groups and companies in the robotics industry. But it is also the painless entry point to robotics for nonprofessional people. ROS was originally developed in 2007 in Stanford Artificial Intelligence laboratory in support of AI Robot STAIR project [QBN07].

4.1.1 ROS architecture

The ROS architecture has been designed and divided into three sections or level of concept. We discuss more about each of concept in the next sections.

- The file system level
- The Computation Graph level
- The Community level

4.1.1.1 The file system level

ROS program is divided into folders like in operating system. These folders have files that describe their functionalities. Abstraction of ROS file system can be found in figure 4.1.

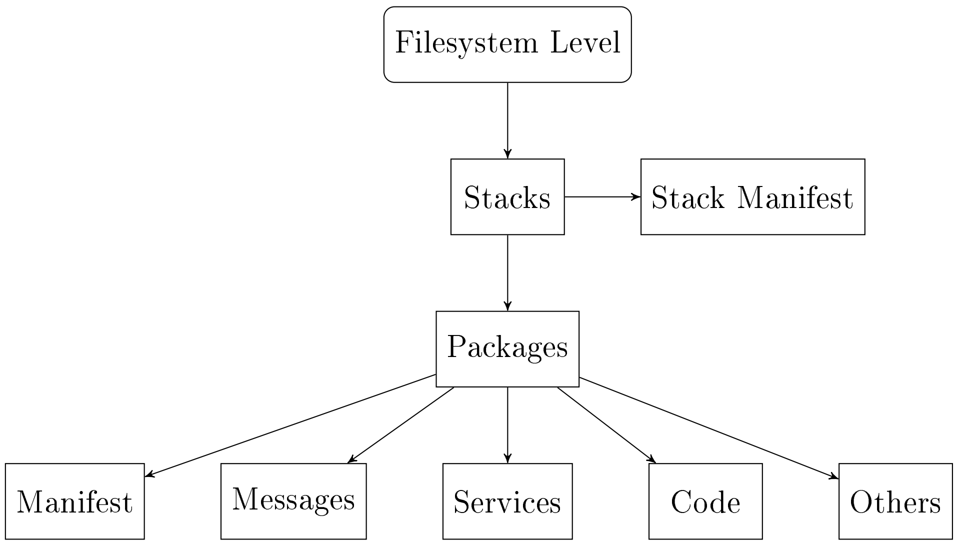


Figure 4.1: Abstraction of ROS Filesystem level.

- **Packages** form the atomic level of ROS. A package has the minimum structure and content to create a program within ROS. The goal of packages is to create minimal collections of code that are easy to reuse.

- **Manifests** provide information about a package, license information, dependencies, compiler flags. Manifests are managed with a file called *manifest.xml.*

- **Stacks** are obtained when you gather several packages with some functionality. The main goal of stacks is to simplify the process of code sharing. Each stack has an associated version and can declare dependencies on other stacks. In ROS, there exists a lot of these stacks with different uses, for example, the navigation stack or manipulation stack.

- **Stack manifests** (*stack.xml*) provide data about a stack, including its license information and its dependencies on other stacks. The bare minimum for *stack.xml* file is much like a readme file.

- **Message** is the information that a process sends to other processes. ROS has a lot of standard types of messages such as geodata, laser range sensor or arrays messages. ROS uses a simplified messages description language for describing the data values that ROS nodes publish.

- **Service** enables request, response communication between nodes. Services behave like classic web servers - one node sends the request and another node sends the response to this request.

4.1.1.2 The Computation Graph level

ROS creates a network where all the processes are connected. Any node in the system can access this network, interact with other nodes, see the information that they send, and transmit data to the network.

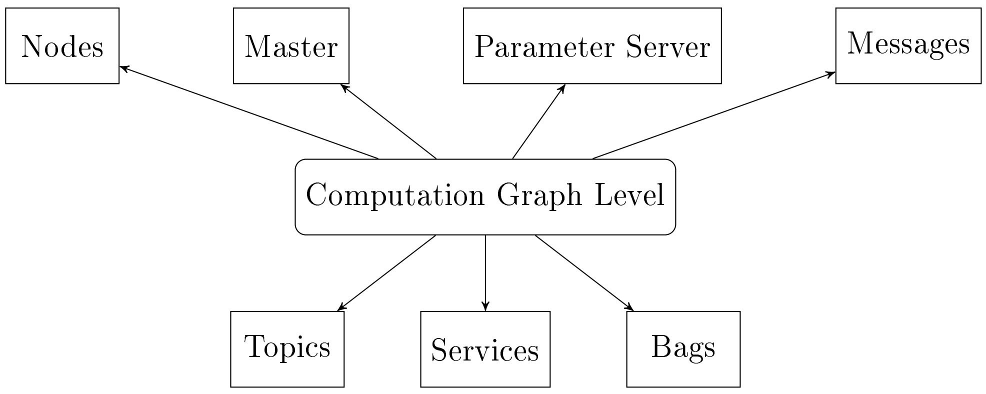


Figure 4.2: Abstraction of ROS Computation level.

- **Nodes** are processes where computation is done. If we want to have a process that can interact with other nodes, we need to create a node with this process to connect it to the ROS network. Usually, a system will have many nodes to control different functions.

- **Master** provides name registration and lookup for the rest of the nodes. If we don't have it in our system, we can not communicate with nodes, services, messages, and others.

- **Parameter Server** gives us the possibility to have data stored using keys in a central location. With this parameter, it is possible to configure nodes while it is running.

- **Messages** allow nodes to communicate with each other. A message contains data that sends information to other nodes. We can use standard messages or we can develop our own type of messages.
- **Topics**: Each message must have a name to be routed by the ROS network. When a node is sending data, we say that the node is publishing a topic. Nodes can receive messages from other nodes simply by subscribing to the topic. A node can subscribe to a topic, and it is not necessary that publishing node exists. This permits us to decouple the production of the consumption.
- **Services**: When we publish topics, we are sending data in a many-to-many fashion, but when we need a request or an answer from a node, we can not do it with topics. The services give us the possibility to interact with nodes. Also, services must have a unique name. When a node has a service, all the nodes can communicate with it, thanks to ROS client libraries.
- **Bags** are a format to save and play back the ROS message data. Bags are an important mechanism for storing data, in our case sensors data. We can use this data later in many ways, for example in visualization or running different types of algorithms.

4.1.1.3 The Community level

The ROS Community level concepts are ROS resources that enable separate communities to exchange software and knowledge. We will not talk about this level a lot because it does not provide us any important technical background.

4.2 Caffe

Caffe [JSD+14] is a Deep Neural Framework created by the Berkeley Vision and Learning Center, UC Berkeley. It reduces the work of the user by allowing the user to define complex deep neural networks in a simple CSS-like language. However, internally, it uses the BLAS libraries, OpenCV library as well as nVidia's CUDA to generate highly optimized code in C++ with a possibility to use GPU to run it. It supports a variety of input types including raw image lists, LMDB, LevelDB, HDF5 multi dimensional data.

Caffe provides a complete toolkit for training, testing, finetuning, and deploying models, with well-documented examples for all of these tasks. As such, it is an ideal starting point for researchers and other developers looking to jump into state-of-the-art machine learning.

We suggest to switch to Caffe or other modern similar Neural Network Framework in the future due these reasons

- Caffe is already implemented into ROS as a package.
- FANN is an old library which is no longer maintained.
- We can not duplicate our old model in FANN, because we do not have the exact training set from which it was created and actual code base is not working correctly with new models.

4.3 Obstacle Avoidance Algorithms

In our work, we studied previous works which propose new obstacle avoidance algorithms using only laser range sensor. Normally obstacle avoidance is considered to be distinct from path planning in that one is usually implemented as a reactive control law while the other involves the pre-computation of an obstacle-free path which a controller will then guide a robot along. In the next sections, we explain few basic algorithms that we studied.

4.3.1 Vector Field Histogram

Vector Field Histogram was proposed in work [BK91] as a reactive obstacle algorithm. The *Vector Field Diagram* uses a two-dimensional Cartesian grid, called the histogram grid, to represent data from laser range sensors [KBM98]. Each cell in the histogram grid holds a certain value that represents the confidence of algorithm in the existence of an obstacle at that location.

A high-level description of the algorithm follows

- Collect current laser range sensor readings.

- Determine the cell in the histogram in which obstacle lies. If the sensor did not detect an obstacle, determine the cell lies at the maximum range.
- Calculate an active window surrounding the center of the robot.
- Set a threshold such that a polar histogram value bellow means that direction is free and above the threshold is occupied.
- Find the closest free direction to the goal.

In picture 4.3, we can see an example of a robot situation and histogram.

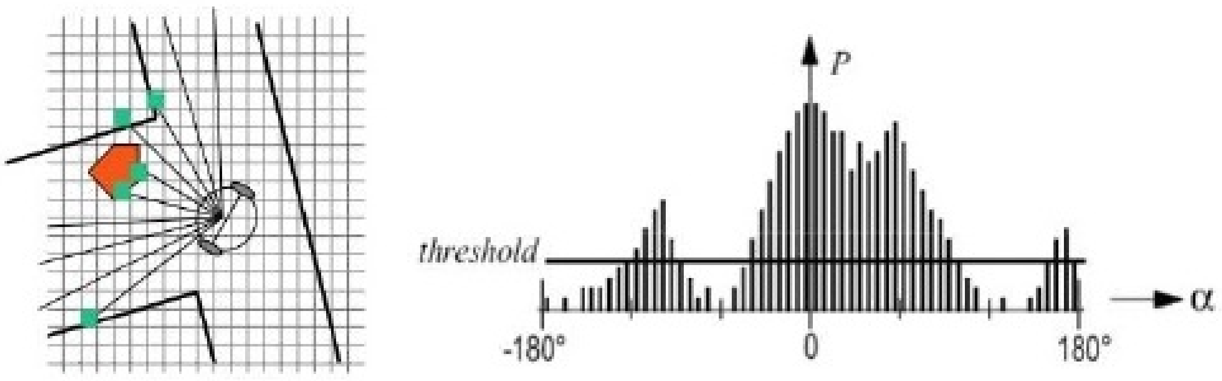


Figure 4.3: Example of VFH algorithm histogram grid

4.3.2 Nearness Diagram Navigation

Nearness Diagram was proposed in work [MM04] as reactive collision avoidance for mobile robots. Nearness Diagram Navigation is reactive based obstacle avoidance algorithm. It uses a *divide and conquer* strategy based on situations to simplify the difficulty of navigation.

Nearness Diagram Navigation carries out the environmental information extraction in three steps. Firstly, from the information available, two nearness diagrams are constructed (the PND and the RND). Secondly, the PND is analysed to identify regions and to select one of them. Thirdly, the RND is analysed to evaluate the robot safety situation. Subsequently, this information is used to identify one of the five general situations.

- The **PND** represents the nearness of the obstacle to the central point. The topology of the environment does not vary in this diagram, so it is used to extract information of environmental characteristics. The PND analysis is performed in three stages. Firstly, gaps in the environment are searched for. From these gaps, regions of the free space are obtained, and finally one of them is chosen by one criteria.

- The **RND** represents information about the nearness of the obstacles from the robot. RND evaluates the robot safety situation based on minimum tolerable distance.

There are five general situations as we mentioned before. They cover all the possibilities among the robot location, goal location and obstacle configuration, but they have to be checked in rigorous order.

- **Low Safety 1**, if there is at least one sector that exceeds the security nearness in the RND, only on one side of the rising discontinuity, which is closer to the goal sector, of the selected valley.
- **Low Safety 2**, if there is at least one sector that exceeds the security nearness in the RND, on both sides of the rising discontinuity, which is closer to the goal sector, of the selected valley.
- **High Safety Goal in Valley**, if the goal sector belongs to the selected valley.
- **High Safety Wide Valley**, if the selected valley is wide.
- **High Safety Narrow Valley**, if the selected valley is narrow.

4.3.3 Smooth Nearness Diagram

Smooth Nearness Diagram proposed by [DB08] is an evolution of *ND+*. As compared with the ND+ navigation scheme, the key difference in approach is that a single motion law is proposed that is applicable to all possible configurations of surrounding obstacles.

The SND method works as follows: first, the laser range data is analysed to determine the structure of obstacles surrounding the robot. The best heading which makes progress towards the goal direction set by the global planner is then selected. This process of determining a safe trajectory is repeated for each sensor update.

In the picture 4.4, we can see one SND algorithm step. The robot detects four gaps, indicated by dashed lines, in the depth measurements around it. There are two types of gaps

(a) is created by neighbouring depth measurements differing by more than the robot diameter

(b) one depth measurement returns no obstacle in range

The four gaps define regions and valleys around the robot, some of which are labelled. The robot choose V_{best}, θ_{rg}, and θ_{og} based on the goal direction θ_{goal}.

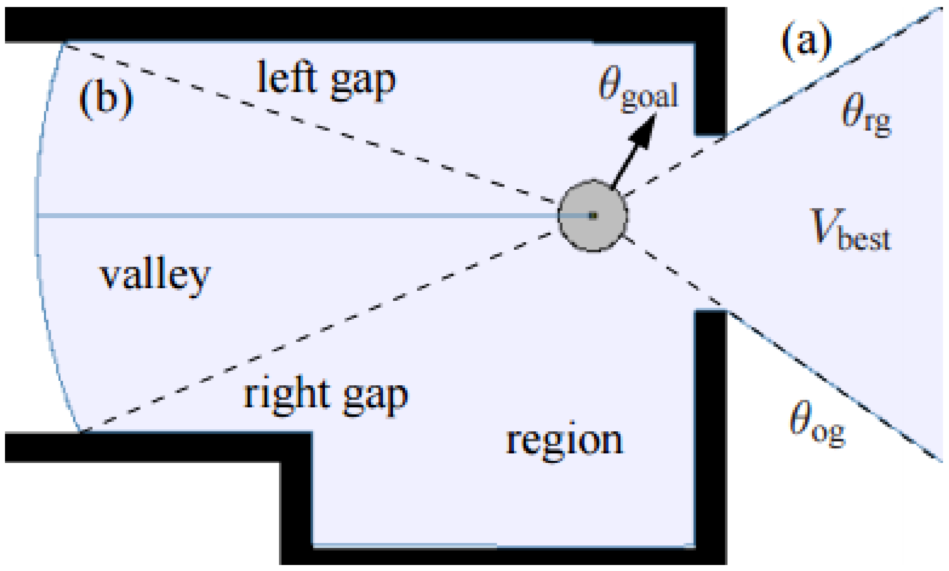


Figure 4.4: One SND algorithm step

Chapter 5

Implementation

In this chapter, we explain some important parts of software implementation. We have legacy code from previous works which we reimplement into *Robot Operating System* framework. We note that we have completely changed the architecture of the system. We have also designed a new probabilistic obstacle avoidance algorithm that uses data from laser range sensor. Details of our implementation including the pseudo-codes can be found in following sections.

5.1 Hokuyo sensor algorithms

5.1.1 Implementation from previous works

The first naive algorithm in the previous implementation of evaluation obstacles from Hokuyo was based on detecting the shortest beam from raw Hokuyo data. This had some shortcomings, because there could be artifacts in data which we incorrectly evaluated in algorithm or results from Hokuyo scans were not correctly synchronized with evaluated direction from the camera. We provide the algorithm pseudo-code in *Algorithm* 2.

5.1.2 Proposed probabilistic algorithm for obstacle avoidance

We decided to rewrite this algorithm before the competition. We agreed, that single boolean variables for each direction is not enough and we need some kind of probability for that. So we proposed and implemented a new algorithm which returns float values

```
input  : Direction index, raw data from Hokuyo sensor
output: Boolean flag indicates the obstacle

C is constant size of half of scanning area width for one direction index;
min ← INFINITY;
foreach beam b of beams in range(direction - C, direction + C) do
    if b < min then min ← b;
end
if min < maximum allowed distance for obstacle then return false;
else
    return true;
end
```

Algorithm 2: Previous naive implementation for obstacle avoidance using only Hokuyo laser range sensor

with probabilities, instead of boolean values. We explain more details about the new algorithm in next sections.

5.1.2.1 Synchronizing directions with camera

Evaluating direction from the camera is divided into eleven parts (it is possible to use more than eleven parts, but for our purpose eleven is enough), every part results into the probability of good direction. We needed from our implementation to be backwards fully compatible with this *API* so we also split laser range data into eleven same equal parts. There were problems with synchronization, which were not connected to software part, but hardware. The reason is that, Hokuyo has detection angle 270°, angular resolution 0.25° and 1081 measurement steps. In the figure 3.3, we can see that index 540 is in the middle of scanning. But in reality, this middle is displaced more to left or right side as we can see in figure 5.1, the green dot is expected centre and the red dot is displaced centre.

So we needed to introduce variable *directions*, which holds corrected indices of beams against the camera, which we *empirically measured.* For simplicity in next parts, we provide the exact mathematical model, which does not count with these anomalies.

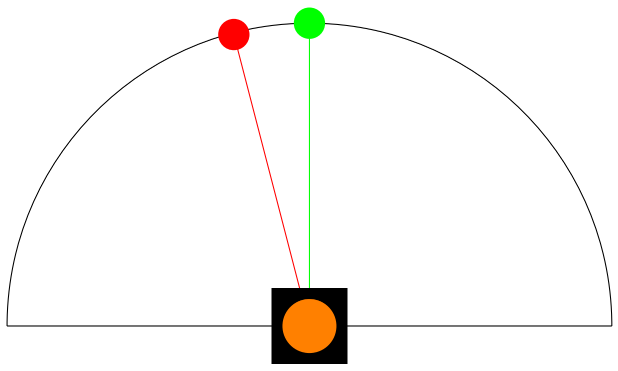


Figure 5.1: Displacement of lidar centre

5.1.2.2 Algorithm initialization

During the initialization phase of our algorithm, we want to calculate maximum distance which has probability $p_i = 1$. The sketch of this idea is in the figure 5.3. Blue circle means the maximum distance of sensor and red ellipse is distance with probability $p_i = 1$. We can see that the largest distance with probability $p_i = 1$ is in the middle because we want prioritize straight direction.

$$b_i = -\frac{-\sin(\alpha) + \sqrt{\sin(\alpha)^2 - 4ac + \cos(\alpha)^2)}}{2a\cos(\alpha)^2} \tag{5.1}$$

α = beam angle in degrees

a = major axis

c = minor axis

Figure 5.2: Equation for calculating maximum distance with weight 1

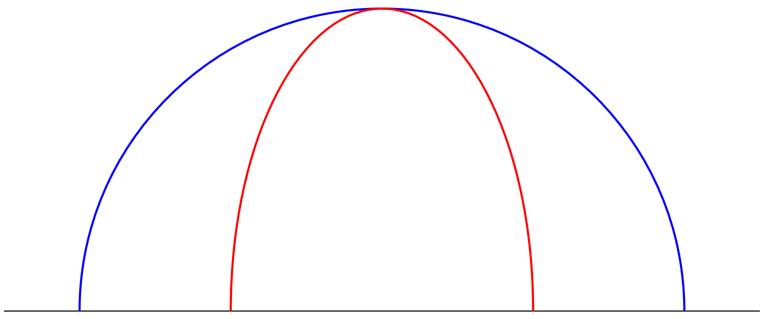


Figure 5.3: Sketch of algorithm initialization

5.1.2.3 Evaluation phase

During the evaluation phase, we want to evaluate probability for each direction. As we mentioned in the beginning of this section, we divide laser range data into eleven equal parts.

We take in mind all neighbour data from the sensor for one direction, not only a few, in comparison with the previous implementation.

$$p_i = \frac{1}{\sum\limits_{i=from}^{to} 1 - |i|} \sum_{i=from}^{to} \left[(1 - |i|) \frac{l_i}{max_i} \right] \tag{5.2}$$

$from$ = index of the first beam which we care about

to = index of the last beam which we care about

l_i = normalized beam at index i, value on the red ellipsis in Figure 5.3

max_i = max(beam at i position, max allowed range), max allowed range is blue semicircle in Figure 5.3

Figure 5.4: Equation for algorithm evaluation phase

5.2 Technologies

Due to previous works, we stuck on *C++* because we want to reuse some parts - mainly low-level parts, which communicate with sensors *Sbot* board, *GPS*, *IMU* and *Hokuyo*.

5.3 Architecture

We split previous code into few parts which are independent. These parts are *sensors publishers*, *debug GUIs*, *algorithms* and *robot control*. This abstraction allows us to easily maintain code, add new sensors, algorithms which result in faster development, faster testing new algorithm and last but not least introducing fewer bugs into code.

5.3.1 Overall

As we mentioned in previous chapter 3.3.2 there was low cohesion and high coupling between classes. Due to nature of *ROS* architecture, it was quite easy to rewrite code into more independent parts. We provide overall lookup at parts and their connection using *ROS* concept *publishers* and *subscribers* in the picture 5.5. The picture consists of several logical parts.

- Ovals are nodes.
- Nodes with the same namespace are in the same rectangle.
- Some nodes are connected with an arrow. The arrow begins in the publisher node and ends in the subscriber node. One publishing node can publish to more nodes and subscriber node can subscribe to more different publishers.

We explain each of those components in following sections.

5.3.2 Sensors publishers

Sensors publishers publish raw data from sensors. We implemented one publisher per one sensor. For each sensor publisher, we use the same template. We provide pseudo-code of this template in *Algorithm* 3.

This allows us to write code, which can be easily modified e.g. we use the same code for publishing data from the real sensor, reading data from log files with the different format or even generating synthetic data. For every sensor type, we created an abstract class with bare minimum of methods namely

- *Init* method initializes sensor
- *Read data* method reads data from sensor and saves to internal structure
- *Get data* method provides data in ROS message format

Classes can be easily extended without changing logic behind the publishing data.

This structure allows us to replace any hardware part of the robot without carrying a lot about software part because we just create the new class for the new sensor.

In the Figure 5.6, we provide overall lookup at the sensors architecture exported via *rqt_graph*.

```
Function main()
begin
    /* node initialization                                    */
    rosInit(node name);
    /* rate object allows us to specify a frequency that we
       would like to loop at. It keeps track of how long it has
       been since the last call to Rate::sleep(), and sleep for
       the correct amount of time                             */
    loopRate ←setLoopRate(loop rate);
    /* repeat while we are connected to ROS network, SIGINT is
       not received or shutdown was called by another part of the
       application                                            */
    while rosOk() do
        readData();
        data ← getData();
        rosPublish(data);
        loopRate sleep();
    end
end
```

Algorithm 3: Pseudo-code for sensor publisher

5.3.3 Debug GUIs

Debug GUIs are nodes which listen for sensors publishers or algorithms. In our work, we implemented original GUIs from the camera and improved version of localization and planning and Hokuyo. In previous poor implementation was quite hard to divide GUIs, algorithms, and control due to high coupling between modules. We implemented similar logic as in sensors publishers - one GUI for one sensor and connected algorithms. Due to nature ROS environment especially sockets, it is *not mandatory* to have running every algorithm. It basically means - we can visualize every part of sensor including algorithms independently. We created a template 4 similar to those in sensors publisher. We can have as much as possible subscribers in the node, but for simplicity, we show only one subscriber in pseudo-code.

As we can see there is one difference between subscribing and publishing. When subscribing to topic, we need to tell ROS to fire callback using *ros::spinOnce()* function. The reason is that when the message arrives it is stored in a queue until ROS gets a chance to fire our callback function.

There are two types of *spin* function, the first one is *spin* and second *spinOnce*. The main difference between these two functions is that *spin* function empties message queue immediately what causes the program to exit shortly after its start. On the other hand, *spinOnce* periodically processes message queue and executes the callback function.

In the Figure 5.7, we provide overall lookup at the GUIs architecture exported via *rqt_graph*.

5.3.4 Algorithms

Algorithm nodes are the combination of publisher and subscriber. The reason is simple, we need to get data from sensors or previous processing such as filtering, algorithms, and publish them to other nodes e.g. robot controller, visualization or next processing of data 5.

In the Figure 5.8, we provide overall lookup at the control and algorithms architecture exported via *rqt_graph*.

5.3.5 Robot Control

The robot control node needs to be divided into two parts, because we want to read and write data to Sbot simultaneously which allows us to communicate at higher speed, resulting in having better control over the direction.

- *Sbot publisher* has the spot between sensors publishers because it publishes basic data about robot odometry, data from ultrasonic sensors and information about the obstacle.

- *Robot control* is the only part, which sends commands to the robot. Robot control collects data from all algorithms nodes and makes final decision based on these data. There is the difference between algorithm node and robot control node because algorithm node behaves like the pipeline - receives data, runs algorithms and publishes results. On the other hand, the robot control collects all data and instead of evaluating result immediately, it has one main evaluation loop which runs at *30Hz*. It collects all saved data, combines data, runs algorithm for calculating the direction and send command (change direction, set speed or stop) to the *Sbot*. This whole process is done during the evaluation phase at *30Hz*. We provide pseudo-code for clarification in *Algorithm* 6.

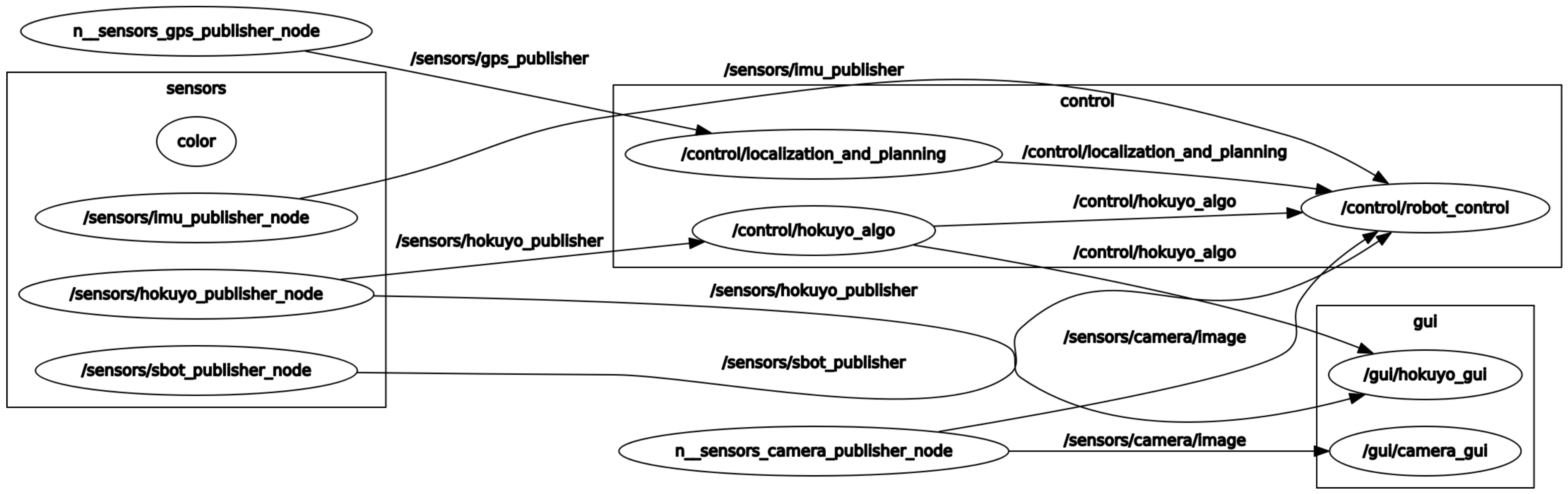

n__sensors_gps_publisher_node
/sensors/gps_publisher
/sensors/imu_publisher
sensors
color
/sensors/imu_publisher_node
/sensors/hokuyo_publisher
/sensors/hokuyo_publisher_node
/sensors/sbot_publisher_node
control
/control/localization_and_planning
/control/localization_and_planning
/control/hokuyo_algo
/control/hokuyo_algo
/control/hokuyo_algo
/control/robot_control
/sensors/hokuyo_publisher
/sensors/camera/image
/sensors/sbot_publisher
gui
/gui/hokuyo_gui
n__sensors_camera_publisher_node
/sensors/camera/image
/gui/camera_gui

Figure 5.5: New architecture built on ROS

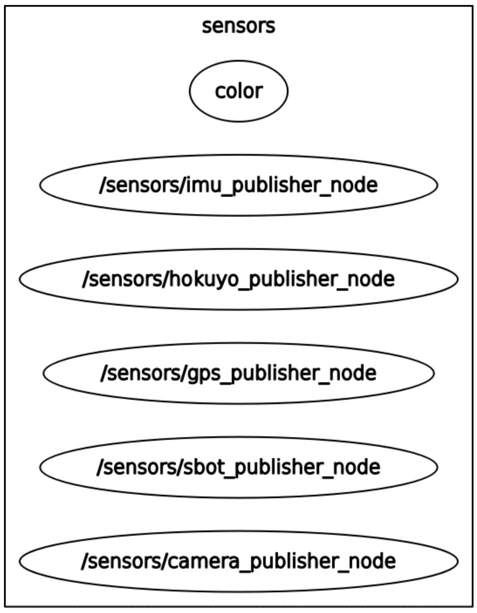

sensors
color
/sensors/imu_publisher_node
/sensors/hokuyo_publisher_node
/sensors/gps_publisher_node
/sensors/sbot_publisher_node
/sensors/camera_publisher_node

Figure 5.6: Sensors publishers namespace

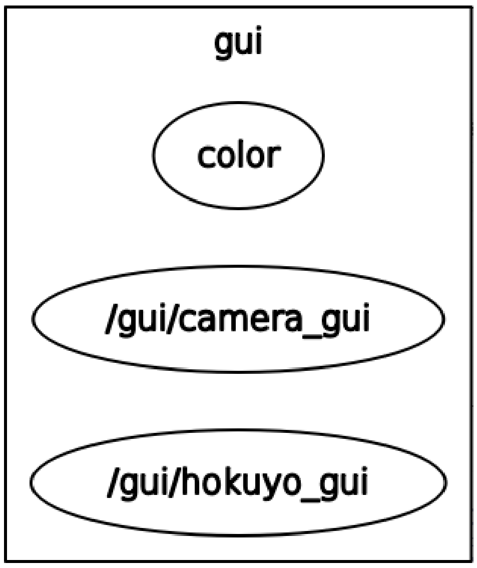


Figure 5.7: GUIs namespace

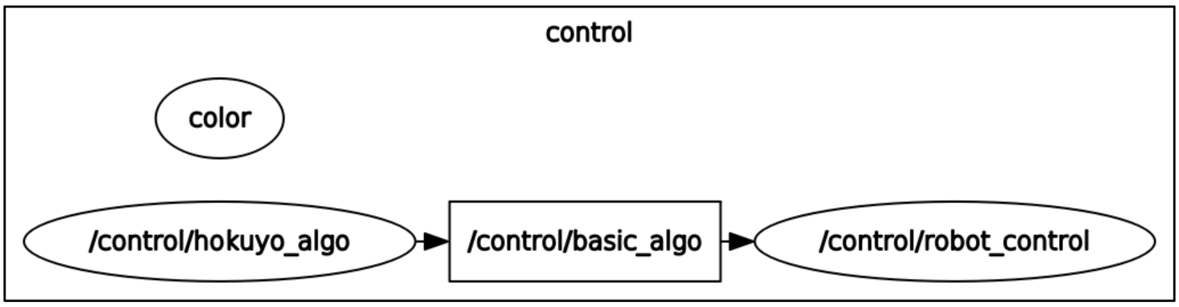


Figure 5.8: Control and algorithms namespace

```
/* callback function is called whenever a new message arrives.
   The only variable is received data                          */
Function callback(data)
begin
    globalData ←data;
end
/* function for visualization data                             */
Function renderWindow()
begin
    render(globalData);
end
Function main()
begin
    /* node initialization                                     */
    rosInit(node name);
    subscribe(channel name, message queue size, callback function);
    /* rate object allows us to specify a frequency that we
       would like to loop at.  It keeps track of how long it has
       been since the last call to Rate::sleep(), and sleep for
       the correct amount of time                              */
    loopRate ←setLoopRate(loop rate);
    /* repeat while we are connected to ROS network, SIGINT is
       not received or shutdown was called by another part of the
       application                                             */
    while rosOk() do
        renderWindow();
        spinOnce();
        loopRate sleep();
    end
end
```
Algorithm 4: Pseudocode for subscribing to publisher and rendering window

```
/* callback function is called whenever a new message arrives.
   The only variable is received data                          */
Function callback(data)
begin
    processedData ←algorithm(data);
    /* broadcast the message to anyone who is connected        */
    rosPublish(processedData);
end
Function main()
begin
    /* node initialization                                     */
    rosInit(node name);
    /* subscribe to the topic with the master.  ROS will call the
       callback function whenever a new message arrives.  The 2nd
       argument is the queue size, in case we are not able to
       process messages fast enough                            */
    subscribe(channel name, message queue size, callback function);
end
```
Algorithm 5: Pseudocode for algorithm node

```
/* callback function is called whenever a new message arrives.
   The only variable is received data                          */
Function callback(data)
begin
    processedData ←algorithm(data);
    /* broadcast the message to anyone who is connected        */
    rosPublish(processedData);
end
Function main()
begin
    /* node initialization                                     */
    rosInit(node name);
    subscribe(channel name, message queue size, callback function);
    /* rate object allows us to specify a frequency that we
       would like to loop at. It keeps track of how long it has
       been since the last call to Rate::sleep(), and sleep for
       the correct amount of time                              */
    loopRate ←setLoopRate(30);
    /* repeat while we are connected to ROS network, SIGINT is
       not received or shutdown was called by another part of the
       application                                             */
    while rosOk() do
        collectData();
        data ←combineData();
        direction ←evaluateData(data);
        /* send command to sbot board                          */
        sendCommand(direction);
        loopRate sleep();
    end
end
```

Algorithm 6: Pseudocode for robot control node

Chapter 6

Experiments

In this chapter, we provide the results of our work. We ran several experiments indoor and also outdoor (during the Robotour 2016 competition in Deggendorf). We tested the robot with proposed algorithm and a new architecture and compared it with the previous implementation.

6.1 GUIs explanation

In this section, we explain windows which contain data from the camera, Hokuyo sensor, and Hokuyo algorithms.

In Figure 6.1, we can see one frame from the camera. There is nothing special in this window, it just contains a plain picture, but we use it a lot in section, where we explain Hokuyo GUI.

In the Figure 6.2, we can see data from

- Hokuyo sensor
- algorithms
- direction result

Hokuyo sensor data begin in the middle and end when beams hit the obstacle. The beam is indicated by a grey line that ends with the turquoise circle (obstacle). There is the wall on the left side on the Picture 6.2 and 6.1 which creates ”turquoise line” on that picture.

The previous algorithm implementation probabilities (yes or no) for directions are in the picture with the red and green colors (red = obstacle, light green = free way

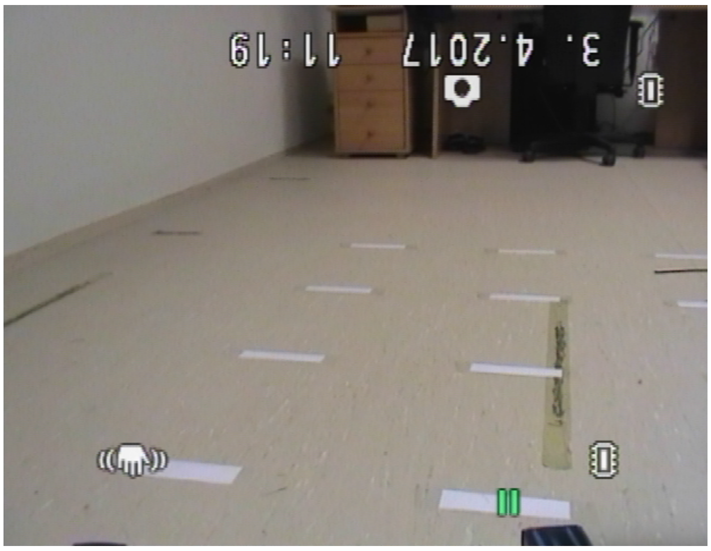


Figure 6.1: Camera GUI

for direction i). Our new algorithm is drawn with turquoise and red circles. The red circle is the direction with the highest probability of free way (the robot direction). Turquoise circles have the lower probability than highest.

The y position of turquoise or red circles in the picture is the probability, that goes from 0 to 1. 0 is at the bottom of the picture and 1 probability on top.

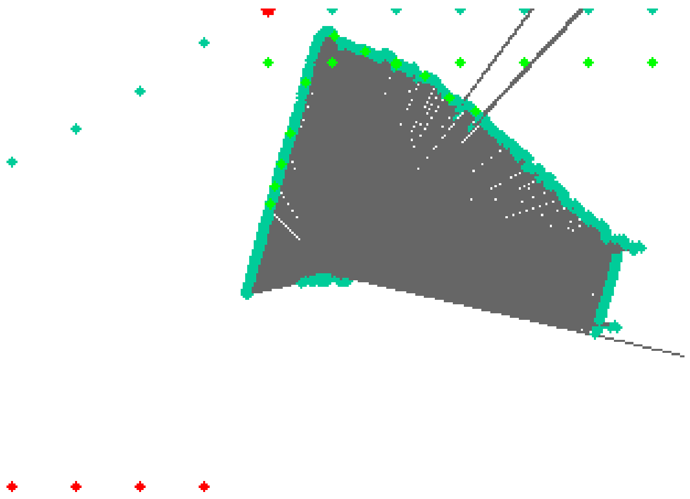


Figure 6.2: Hokuyo GUI

6.2 Comparison

In this section, we compare our algorithm with previous in some sample situations. We start our comparison with the simplest situation without obstacles to more complicated with more obstacles, even the wall.

6.2.1 No obstacles

There are no obstacles in this simple situation. We take records from the previous section that are showed on the Pictures 6.1 and 6.2. When we compare result from our algorithm with the previous, there is almost no difference between results. That means, there is almost no improvement in basic situations.

6.2.2 One near obstacle

In this example, we added one near obstacle in front of the robot. The situation is shown on the Pictures 6.3 and 6.4 We can see that our algorithm recommended the right most way. On the other hand, previous algorithm recommended only almost the direct direction, that is not correct because there is an obstacle.

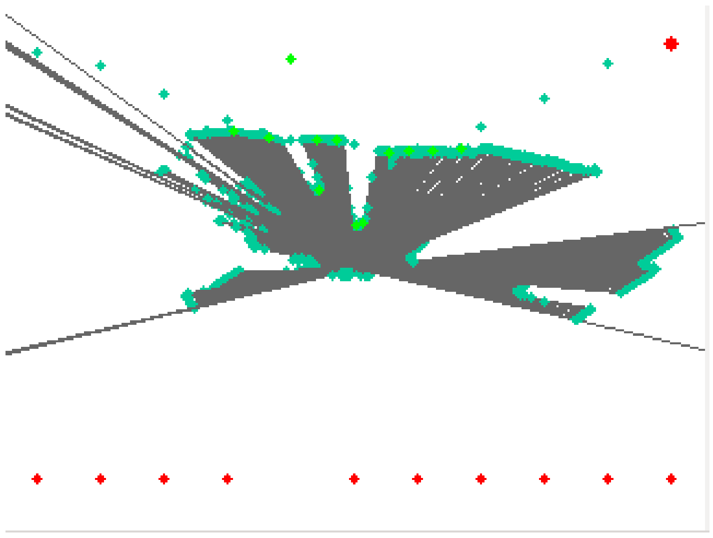


Figure 6.3: One obstacle, Hokuyo perspective

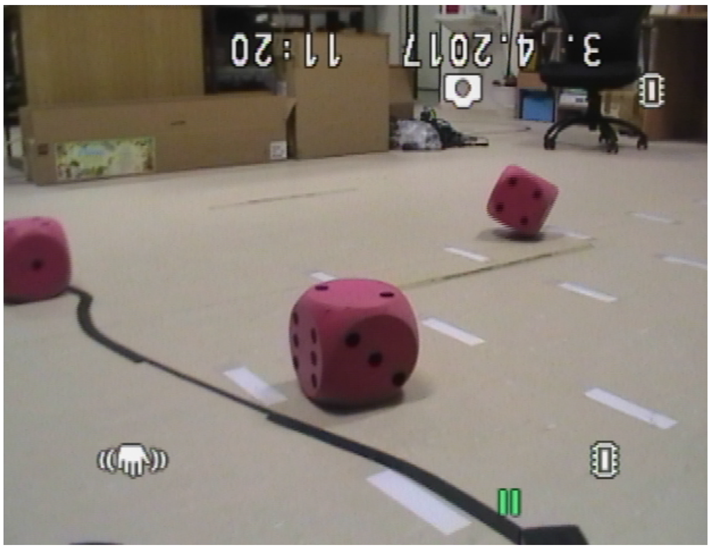


Figure 6.4: One obstacle, camera perspective

6.2.3 Free space in the wall

In this situation, we created a wall with one empty space. Pictures for this situation are 6.5 and 6.6.

The previous algorithm can not find the free space in the wall (on the right side). The algorithm thought that there is not space in front of the robot.

On the other hand, our algorithm found the free space in the wall and robot went through it.

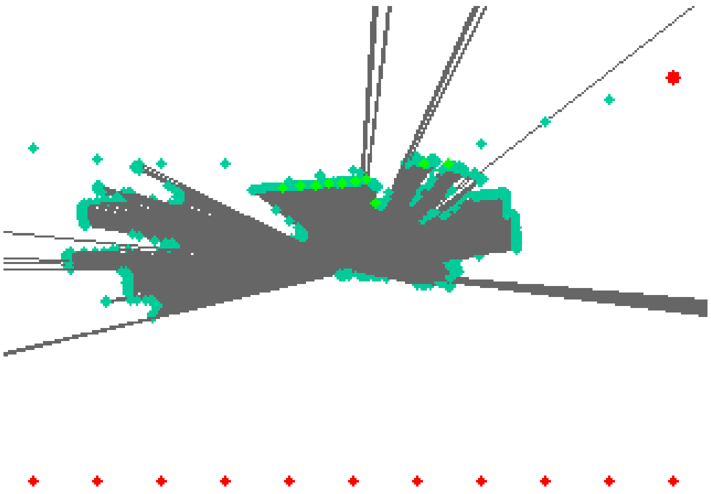


Figure 6.5: Free space in wall, Hokuyo perspective

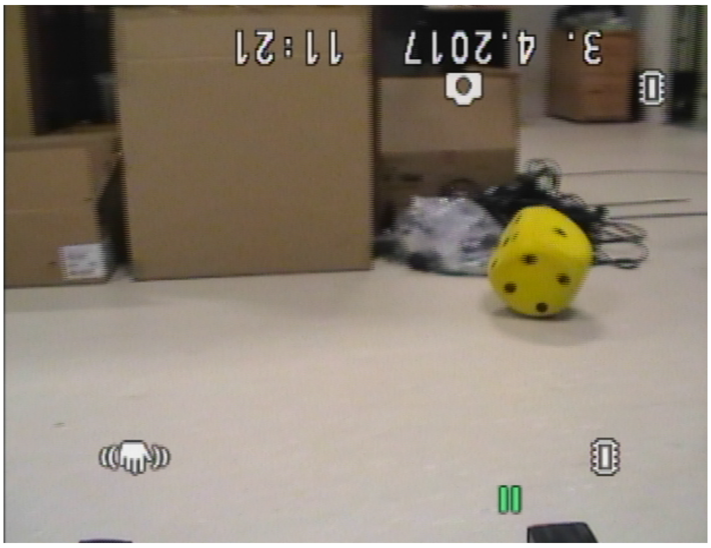


Figure 6.6: Free space in wall, camera perspective

6.2.4 Near the wall

In this situation, we compare the behaviour of algorithms near the wall. As we see the previous algorithm does not see any way. It is expected because the wall distance is closer than minimal distance for the obstacle.

In comparison with our new probabilistic algorithm, we can find the way on the right side because there is some free space which increases probability (not so high due to the distance to the wall) on the right side.

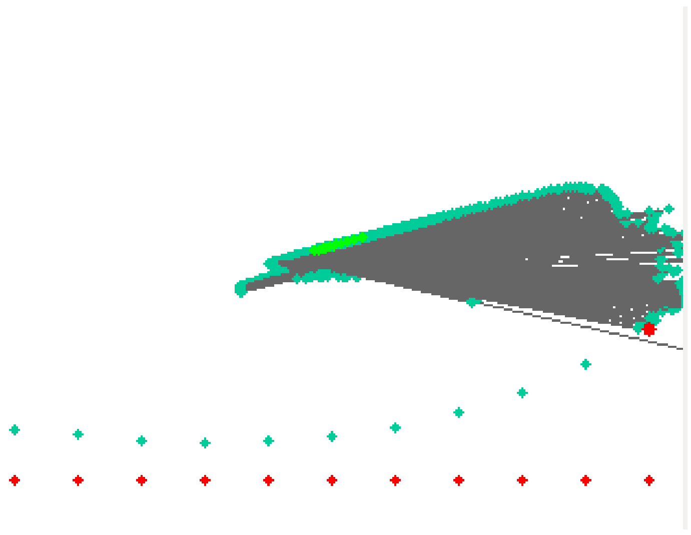


Figure 6.7: Near the wall situation, Hokuyo perspective

Figure 6.8: Near the wall situation, camera perspective

Conclusion

In this thesis, we worked with an autonomous robot Smely Zajko. This robot is a result of previous works [MN10] [Mor15]. We focused on integrating a new sensor - laser range sensor into the set of existing sensors like GPS, camera and compass.

At first, we provided a basic overview of terms used in artificial intelligence and robotics and we introduced the robot. We analysed the state of the robot before we started working on it from hardware and software point of view.

We proposed a new algorithm for reactive obstacle detection and avoidance using the laser range sensor (Hokuyo). We implemented this new approach into the existing code and it cooperated with other sensors. Using this improved version of Smely Zajko, we attended Robotour contest held in Deggendorf, Germany in September 2016.

Although we were quite successful and won the third place, we realised some shortcomings of our implementation. There was heavy rain during the contest and the main problem with Hokuyo sensor were rain drops. Beams from the sensors were stricken back into the sensor and were creating "imaginary" obstacle.

Based on these observations, we decided to relocate the sensor. There were also problems with integration of our algorithm. Many parts of the code had low cohesion (a class did a lot of jobs that do not have much in common) and very high coupling (a class has a lot of dependencies) that caused some performance problems. That is why we reimplemented software from the scratch and completely changed its architecture.

We described the process of reimplementation in the chapter Implementation. As the main implementation technology we chose ROS framework that is more flexible, allows running multiple independent experiments, record and replay data from log files and is easy portable to another robot. In the chapter New approaches, we described this framework and also further algorithms for obstacle avoidance.

In the chapter Experiments, we provided the results of our work. We compared the results from original system with the results from our brand new implementation and algorithm. We can conclude that our implementation was successful. The results show that the robot is able to detect a more suitable path with our algorithm mainly because we work with probability for each direction and not with the boolean values.

However, there is still some place for improvement and further research. We propose new ideas and improvements that can be implemented in the future work:

- Rewrite *GUI*s from *OpenCv* into *Qt* library
- Improve control with different algorithms
- Change *Sbot* board into *Raspberry Pi* or *Arduino*
- Change the main computer into the embedded board like *NVidia Jetson* which is capable for faster running operations on the graphic card (can increase neural network performance) and also has several *GPIOs* for direct connection more sensors
- Create *GUI*s which allow to change constants and algorithms during the run
- Create environment for tuning algorithms during the runtime

Bibliography

[AM13] Enrique Fernández Aaron Martinez. *Learning ROS for Robotics Programming.* 2013.

[Asi41] Isaac Asimov. *I, Robot.* Astounding Science Fiction, 1941.

[BK91] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, Jun 1991.

[DB08] J. W. Durham and F. Bullo. Smooth nearness-diagram navigation. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 690–695, Sept 2008.

[Hay98] Simon Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall PTR, 2nd edition, 1998.

[JSD$^+$14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[KBM98] David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems.* MIT Press, Cambridge, MA, USA, 1998.

[MM04] Javier Minguez and L. Montano. Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, Feb 2004.

[MM12] Ameet Talwalkar Mehryar Mohri, Afshin Rostamizadeh. *Foundations of Machine Learning.* The MIT Press, 2012.

[MN10] Pavel Petrovič Miroslav Nadhajský. *Robotour Solution as a Learned Behavior Based on Artificial Neural Networks.* 2010.

[Mor15] Michal Moravcík. *Autonomous mobile robot for Robotour contest.* 2015.

[MR93] Heinrich Braun Martin Riedmiller. *A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm.* 1993.

[Mur00] Robin R. Murphy. *Introduction to AI Robotics.* The MIT Press, 2000.

[Nis03] Steffen Nissen. *Implementation of a Fast Artificial Neural Network Library (FANN).* Department of Computer Science University of Copenhagen, 2003.

[PR06] P. Crowe P. Riley. Airborne and terrestrial laser scanning. 2006.

[QBN07] Morgan Quigley, Eric Berger, and Andrew Y. Ng. Stair: Hardware and software architecture. 2007.

[Rat86] Frank Rattay. *Analysis of Models for External Stimulation of Axons.* 1986.

[ros] Ros filesystem concepts. `http://wiki.ros.org/rosbuild/ROS/Concepts`. Accessed: 2016-11-15.

[Rum86] Geoffrey E. Hinton R. J. Williams Rumelhart, David E. *Neurocomputing: Foundations of research.* 1986.

Appendix A

Application

Application source codes, which we created in this work, are provided on attached CD.