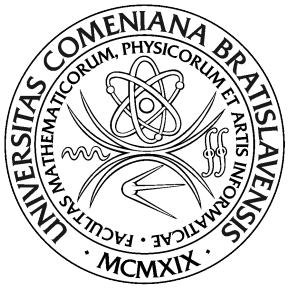
COMENIUS UNIVERSITY, BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND
INFORMATICS



LOCAL MAP FOR A ROBOT FOR THE ROBOTOUR CONTEST

Diploma thesis

2019 Bc. Michal Fikar

COMENIUS UNIVERSITY, BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

LOCAL MAP FOR A ROBOT FOR THE ROBOTOUR CONTEST

Diploma thesis

Study programme:	Applied Informatics
Study field:	2511 Applied Informatics
Department:	Department of Applied Informatics
Supervisor:	Mgr. Pavel Petrovič, PhD.

Bratislava, 2019 Bc. Michal Fikar

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

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

Názov: Local map for a robot for the Robotour contest
Lokálna mapa pre robota do súťaže RoboTour

Anotácia: Smelý Zajko je mobilný robot, ktorého cieľom je doručovanie nákladu vo vonkajšom prostredí. Bol vyvinutý v predchádzajúcich troch diplomových prácach a každoročne sa zapája do súťaže RoboTour. Jeho riadici systém založený na platforme ROS využíva neurónovú sieť na rozpoznávanie cesty na základe obrazu z kamery. Na globálnu navigáciu používa kompas, GPS a mapu vonkajšieho prostredia. Lokálna navigácia je však naprogramovaná ako jednoduchý reaktívny systém kombinujúci informácie z lidaru, globálnej navigácie a kamery. Cieľom práce je rozšírenie softvérového riadiaceho systému o nové črty. Napríklad, o lokálnu mapu, vytváranú z dostupných senzorických dát, v ktorej si robot plánuje trajektóriu. Predpokladá sa inštalácia druhého lidaru snímajúceho povrch chodníka. Očakáva sa účasť študenta na nasledujúcom ročníku súťaže.

Literatúra: H. Choset, et.al.: Principles of Robot Motion, Theory, Algorithms, and Implementations, MIT Press, 2005.
F. Duchoň: Lokalizácia a navigácia mobilných robotov do vnútorného prostredia, Nakladateľstvo STU, 2012.

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

Dátum zadania: 06.10.2017

Dátum schválenia: 12.10.2017 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I declare that this Diploma thesis is my own work, using only the sources listed in the bibliography and with help of my supervisor

. . . . . .

Bratislava, 2019 Bc. Michal Fikar

Acknowledgments

I would first like to thank my supervisor Mgr. Pavel Petrovič, PhD. for his help and advice with my thesis. I would also like to thank my family and friends for their support during my studies.

Abstract

This thesis describes a new navigation system for robot Smelý Zajko that takes part in Robotour contest - an outdoor delivery challenge. The described navigation system is based on a local map that fuses data from various sensors into one unified representation of nearby obstacles. We further propose 2 algorithms to determine the best course of action for the robot, based on the state of the local map. The new navigation system outperforms the old one, which often struggled with obstacle avoidance. Finally we evaluate performance of the robot on the Robotour 2018 contest in Lednice, Czech Republic.

Keywords: local map, outdoor navigation

Abstrakt

Táto práca popisuje nový navigačný systém pre robota Smelý Zajko, ktorý sa zúčastňuje súťaže Robotour, ktorej cieľom je doručovanie nákladu vo vonkajšom prostredí. Popisovaný navigačný systém pracuje na báze lokálnej mapy, ktorá kombinuje dáta z rôznych senzorov do jednej spojenej reprezentácie okolitých prekážok. Ďalej navrhujeme 2 algoritmy na určenie najlepšieho postupu robota v danom stave lokálnej mapy. Nový navigačný systém dosahuje lepšie výsledky ako predchádzajúci, ktorý mal často problém vyhnúť sa prekážkam. Na záver vyhodnocujeme výsledky robota na súťaži Robotour 2018, ktorá sa konala v Ledniciach v Česku.

Kľúčové slová: lokálna mapa, navigácia vo vonkajšom prostredí

Contents

Chapter 1

Introduction

Autonomous mobile robots are slowly becoming parts of our daily lives. Some are more common, such as the robotic vacuum cleaners that can be seen helping with chores in many households. Others are still in the near future, like the delivery drones Amazon is planning to use. Probably the most prominent branch of autonomous mobility applications nowadays are self-driving cars. Manufacturers like Tesla are including advanced driver assistance features such as automatic steering, adaptive cruise control and automatic parking. It might be even possible that fully self-driving cars will become reality in a matter of years once legal and technical issues get solved.

Autonomous mobility challenges are also staples of many robotics competitions worldwide, from simple line followers to advanced outdoor navigation challenges. The latter category also includes the Robotour contest - outdoor delivery challenge organized by Czech robotics community robotika.cz. Goal of the contest is to successfully navigate park roads and carrying a 5 kg load (usually a beer keg) for bonus points.

Smelý Zajko is an autonomous outdoor robot built for the Robotour contest. Developed at the Department of Applied Informatics since 2010 as the

subject of multiple theses and smaller projects, this robot takes part in the contest annually.

Subject of this diploma thesis is to create a new navigation system for Smelý Zajko, based on local map of the environment. The map will hold recorded data from various sensors and determine how to proceed in any given situation. Chapter 2 will overview the outdoor navigation problem and common approaches to solve it and chapter 3 describes the hardware and software solutions on Smelý Zajko at the start of this thesis. Following chapters describe our local map design and details of the implementation. Concluding, we will discuss the performance of the robot on the 2018 Robotour contest and evaluate areas for improvement.

Chapter 2

Problem overview

Robot localization and environment mapping are two closely related topics. Mapping refers to construction of a map of some sorts, from grids of numerical values to topological representation of spaces. Localization is the process of finding position of an agent in a given space (or map), and further maintaining it as the agent moves.

Simultaneous localization and mapping (SLAM) refers to a problem, where an agent needs to solve both of these problems at once. These are most often found with vehicles, such as self-driving cars, robotic vacuum cleaners or unmanned drones.

In this chapter we will cover mapping and localization techniques related to our problem, navigating an outdoor mobile robot. First we will examine obstacle detection methods using laser rangefinding and camera image analysis, and then mapping.

2.1 Lidar obstacle detection

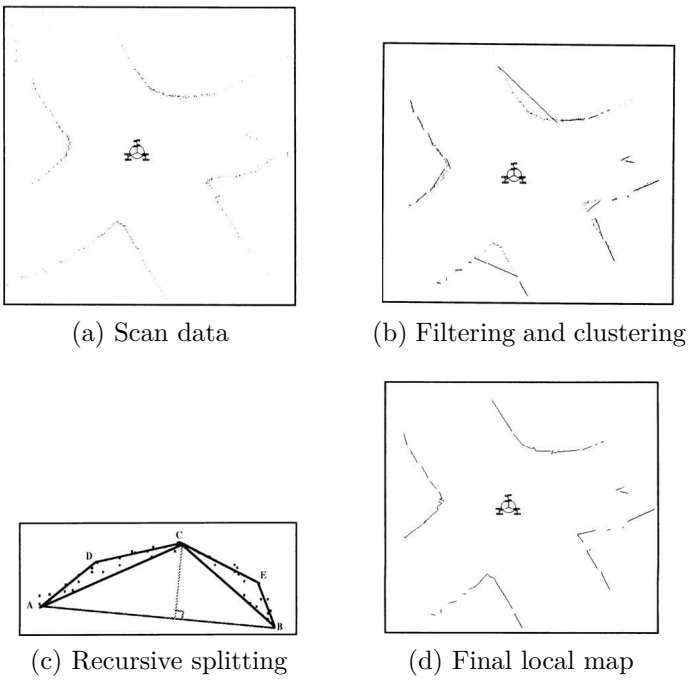
Lidar (combination of words light and radar, later made into an acronym for Laser Imaging, Detection And Ranging) is a distance measuring method based on shinig a pulsing laser on a target, and measuring the reflected pulses with a sensor. Modern solutions usually mount the devices in rotating housings, allowing them to measure distances in a 2D plane or even complete 3D space.

Gonzalez et al. [GOH93] describes a method to build a local map of indoor environment of an indoor robot from data provided by laser rangefinders. They define local map as the representation of environment, as perceived by the robot from a given pose (position and orientation). On the other hand, global map refers to the whole environment, in which the robot operates.

Authors break down the global map building process into 5 simple steps: build a map given the initial robot pose, move to next location, determine new pose, build new local map for the new pose and merge to the original. As a result, they are trying to obtain a map built from 2D line segments, approximating the shape of obstacles.

In order to create local map (fig. 2.1d) for a given pose, authors first need to filter out the sensor data (fig. 2.1a) to get rid of unreliable measurements. Filtered data are then broken down into clusters based on distance between points and then further segmented into groups suitable for line fitting (fig. 2.1b).

The segmentation is calculated by a recursive algorithm. At start, first and last points of a cluster are connected with a single line. Then the point with the maximum distance from this line is found and the cluster is split in two parts. The process then repeats for the pars, until the furthest points are close enough to their respective line segment (fig. 2.1c). The value of this



(a) Scan data (b) Filtering and clustering

(c) Recursive splitting (d) Final local map

Figure 2.1: Building local map from sensor data [GOH93]

threshold parameter needs to be carefully selected based on the properties of sensor and environment.

2.2 Camera obstacle detection

Gini and Marchi [GM02] developed a indoor robot navigation system based on image from a single camera. Camera mapping offers multiple advantages compared to laser rangefinding methods. Biggest difference is in cost of such system, as even cheap cameras can provide sufficient quality of images. They are also able to use more features for mapping, such as detecting obstacles based on textures. On the other hand, their reliability can be worse than laser scanners.

In order to properly use a camera for navigation, it needs to be calibrated first. Pinhole camera model allows for conversion of image coordinates into

world space, but consideration for other camera characteristics (such as focal distance or its position and rotation in the world) is needed. To actually calibrate the camera, we need to take a picture of an object of known size and position in relation to the camera and measure its position in the image space. From these values the calibration matrix can be solved and then used to find world space positions of image features.

2.2.1 Statistical approach

In a known environment, distinguishing road from obstacles can be done on pure statistical basis. One can analyze sample images to find out important features for given areas (roads, sky, trees, buildings ...) and then use them to label new images. There are many possible features that can be used for this, from simple ones such as pixel colors to more advanced textures, gradients or combinations of other features.

Choosing correct one depends mostly on the conditions in which the algorithm will run. For example separating sidewalk from grass can be easily done by pixel color in HSL color space. Dark sidewalk will have low lightness and grass will be dominantly of green hue.

In order to detect floor from obstacles, [GM02] decided to perform statistical analysis based on textures. They calculate 4th order moments for each RGB value of a pixel. Since the 4th order moment has high values when there is a sharp change of colors, it can be used to detect areas where the obstacles meet the floor.

This process is computed only for some reference pixels to save time, and then interpolated for the other pixels. After the computation a new image is constructed from these values, transformed into HSL color space, filtered and binarized by a dynamic threshold to obtain floor and obstacle regions.

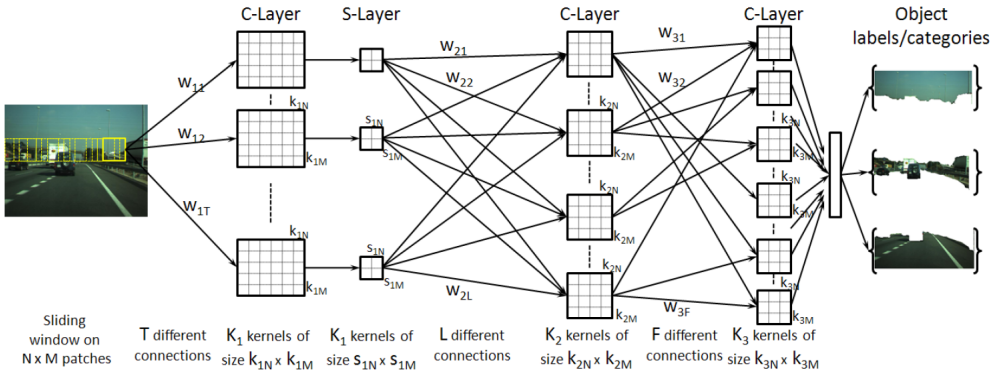


Figure 2.2: Example of CNN architecture [AGLL12]

2.2.2 Convolutional neural networks

Convolutional neural networks are a powerful tool for image processing tasks. Using the convolutional layers (fig. 2.2), they are able to analyze areas of the image with context and with sufficient depth also identify higher-level features. Simple obstacle detection can be equivalent to basic image segmentation, but deeper CNNs can be easily trained to also classify the obstacles (pedestrians, cars, trees...).

There are multiple approaches to road segmentation (distinguishing road surface from obstacles) from images. Common models use pixel level features (color, textures) to group areas of the image [AGLL12], but these have certain issues. Color approaches struggle in different lightning conditions and textures rely on repeating features, such as road markings. Alvarez et al. propose a new kind of texture descriptor to obtain maximal uniformity in road areas [AGLL12] and create a CNN model to segment images into horizontal areas (road), vertical areas (obstacles) and sky.

Their texture descriptor tries to minimize variance (estimated from a histogram) of a small area. Histograms for many different color planes (such as intensity or RGB color channels) are considered, and their minimal vari-

ance linear combination is calculated. The weights are then used to evaluate pixels of the original image, to calculate likelihood of belonging to the road surface.

2.3 Mapping

Gini and Marchi [GM02] use a grid map of numerical values, representing how much the robot "trusts" them to be free, obstacle or unknown. The size of the grid is chosen to provide best results based on the dimensions of the robot.

The robot fills the map with data obtained from sensors, assigning these "trust" values. When the robot moves and detects new obstacles, the map is enlarged and new data is added.

In order to properly merge these new values to the map, the movement of the robot needs to be tracked. In the following sections we will cover using motion sensors (odometry), satellite navigation and optical tracking.

2.3.1 Odometry

Odometry is a method of measuring movement relative to starting point using motion sensors (also referred to as dead reckoning). Solutions range from simple such as using optical sensors to measure wheel rotation to more advanced methods using gyroscopes, compasses and accelerometers. Odometry is fairly accurate over small distances, but gets increasingly inaccurate over longer distances.

Chong et al. [CK97] classify odometry errors as systematic (uncertainties in wheel diameters or their distance) and non-systematic. Systematic errors can be usually solved by calibrating the odometry system, for example

having the robot drive a predetermined test path and adjusting the system to account for inaccuracies. Non-systematic errors cover other sources, such as factors external to the odometry system (wheel slipping, floor roughness etc.).

Authors further propose robot design to minimize systematic errors. This robot has two independently driven wheels, as well as two extra odometry wheels. The odometry wheels are sharp-edged (to reduce wheel base uncertainty) and mounted on linear bearings, allowing them free vertical motion (to reduce effects of slippage due to weight load).

2.3.2 Satellite navigation

Satellite navigation can be used to determine robot's position and speed, but its not accurate enough to be reliably used as only means of tracking. Environmental conditions (cloud cover, proximity of obstacles such as buildings or trees) can severely degrade its performance or even disable it completely. Due to it's relatively small precision (only a couple meters at best) it's used almost solely for outdoor localization.

[PPU02] describes localization algorithm based on Kalman filtering that combines data from a GPS unit with map data and inertial sensors to calculate motion of an outdoor robot.

Authors identify GPS as the ideal means of localization in larger open outdoor areas (fig. 2.3), since it solves many of the problems that can trouble other types of sensors. Namely, ultrasonic and laser rangefinders have trouble with wind or rain and relatively short ranges.

Accuracy of GPS sensors is analyzed, showing that not the number of satellites, but their geometrical position of satellites in the sky is a good measure of precision. Another important aspect of GPS error is the way it

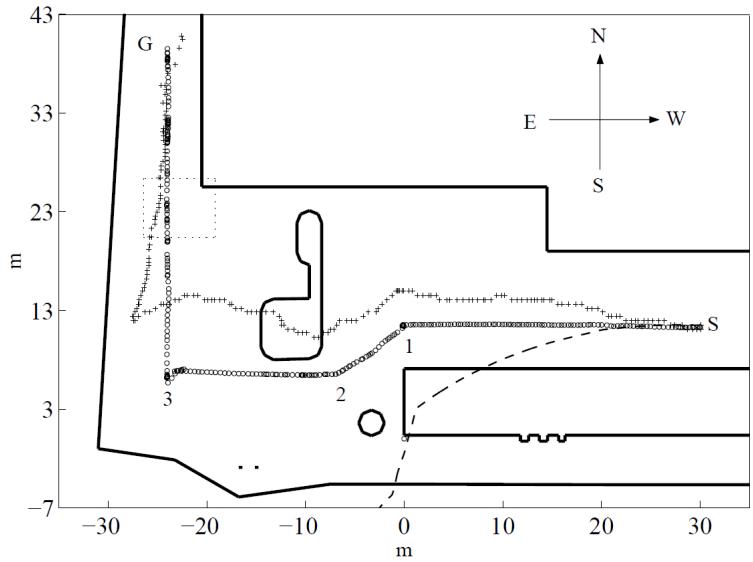


Figure 2.3: Comparison of actual robot path (circle), odometry (dashed line) and GPS (plus) [PPU02]

changes over time - fast changing errors can be easily filtered out but on the other hand slower errors that can't provide smoother movement.

They further describe the localization system for ATRV-Jr robot. The robot has differential drive (wheels on opposite sides can be controlled independently) and is equipped with a variety of sensors (motor rotation sensors, laser rangefinder, inertial measurement unit and GPS). Their algorithm uses inertial data as a precise measure of movement, correcting the inaccuracies of GPS.

2.3.3 Optical tracking

Optical tracking, also called visual odometry, is process of determining position, rotation and their changes based on camera images. Optical tracking works best in a calibrated stereo camera setup, but can be used even with one camera (albeit with less degrees of freedom).

[NNB04] describes visual odometry algorithm capable of working both with mono and stereo camera configurations. The algorithm detects point features in the video feed and matches them between pairs of frames. The feature points are matched based on their distance in the image to produce tracks over time. Tracks are then used to estimate the geometry of the environment. The estimation algorithms are the only difference between mono and stereo camera algorithms. Once the geometry is calculated, the camera motion is estimated using RANSAC.

The algorithm was tested on a vehicle with two cameras and compared against navigation system consisting of GPS and IMU. The visual system was able to outperform inertial navigation and able to provide very accurate position information even after driving hundreds of meters.

Chapter 3

Previous solution

The following sections will describe the construction and software of the robot Smelý Zajko at the beginning of work on this thesis. First will be hardware and software specifications, followed by description of the algorithms controlling the robot and evaluation of its performance.

3.1 Robot description

Smelý Zajko was incrementally developed over the past 5 years of Robotour contest as part of multiple projects and theses. It has a custom-built chassis from plywood and aluminium with 3 wheels - 2 motorized (differential drive - each wheel powered independently) and one pivoting. The robot is controlled by a laptop which runs most of the software. The sensor suite consists of following:

- 7 Ultrasonic sensors (2x SRF-04 forwards, 3x HC-SR04 back and 1 HC-SR04 on each side)
- GPS module

- Inertial measurement unit (HMC6343)
- Lidar (Hokuyo UST-10LX, mounted at the front of the robot, horizontal 180° field of view)
- Optical wheel rotation encoders on motorized wheels (Parallax Motor Mount & Wheel Kit)
- High field-of-view camera
- Infrared distance sensor (Sharp IR sensor 2Y0A21 for payload detection)

GPS, IMU and camera are connected to the laptop via USB and Lidar uses ethernet connection. Encoders and ultrasonic sensors are managed by two Arduino Nano boards (one used solely for odometry encoders and one for ultrasonic sensors and motor control). They communicate with one another over custom parallel connection, and with main control laptop via serial over USB.

Robot is controlled via ROS Kinetic (Robot Operating System) running on Ubuntu 16.1 on the laptop. The ROS modules are written in C++, with the exception of camera image processing, which is done in Python. Camera is accessed via OpenCV and the images are analyzed by neural network created in Keras and Theano.

3.2 Logic and control

The robot control is split into multiple ROS modules, which can be split into 3 logical groups: Sensor, control and visualization. Sensor modules interact with sensors on lower levels and publish their data into ROS. Control modules

subscribe to sensor data and use it to perform calculations. Visualizations can be done directly from sensor data or from computed control outputs.

3.2.1 Localization and routing

Since the robot operates in a known environment, map of the area is prepared ahead of time. Robot uses data from Open Street Maps to construct a network of allowed roads and walkways. Approximate position is obtained by GPS, which is then projected onto the nearest road to reduce GPS inaccuracies. Target points (loading, unloading or end zone) have their coordinates also defined.

Graph of roads is constructed in order to find best path to the target point. Pathfinding is done by a basic breath first search algorithm. Performance is not a big issue since the contest areas are relatively small. Path consist of line segments (equivalent to segments from OSM) and is stored in a list. As the robot advances further on this path, passed segments are removed. The path is recalculated if the robot finds itself on a segment which is not part of the path (either due to navigation or GPS errors).

3.2.2 Steering and obstacle avoidance

Steering is controlled by a central algorithm, combining data from localization and sensors. Robot knows its current bearing from the compass, and target bearing is calculated from the path to target. An circle is centered on the robots current position, and its intersection with the target path is found. At any point in time, the robot tries to directly reach this point.

Near sharp turns or crossroads the intersection point can be on the next path segment, which should result in smoothly steering the robot around the corner. This approach proved to be inefficient, since GPS inaccuracies were

large enough to shift the detected position to another path near crossroads. As a result, the robot often tried to turn too early or too late and ended up trying to leave the road.

GPS issues were meant to be compensated by camera road detection, but this system wasn't working properly during our testing. The output of the image segmentation was extremely noisy, giving hardly any different results for roads and grass. If the system worked properly, it would steer the robot away from the grass, making it stay on the road until crossroad was reached and turning would be possible.

Lidar is used for obstacle avoidance, intended to detect vertical obstacles (such as benches, other robots or pedestrians). Its algorithm uses a PID controller to smoothly steer robot away from upcoming obstacles.

Since the robot is penalized for hitting any obstacle, ultrasonic sensors are used as emergency stops. If they detect any object in close proximity forwards, all other robot controls are overridden and robot is stopped. Since these kind of obstacles are likely to be pedestrians, the robot first waits for them to move out of the way, and after couple seconds tries backing up, turning and driving around.

3.2.3 Evaluation

Although Smelý Zajko was able perform well in previous Robotour competitions (even winning it in 2014), subsequent changes to hardware and software caused parts of it's control system to malfunction. As a result, the robot was only able to successfully navigate simple areas and avoid some obstacles. Navigation system couldn't compensate for GPS inaccuracies and camera obstacle detection proved to be completely unreliable.

The robot was unable to distinguish walkways from grass, which fre-

quently ended up with it getting stuck. Ultrasonic sensors detected tall grass as obstacles - stopping the robot while it was driving along the edge of the path. After waiting for several moments the robot reversed to try navigating around the obstacle, but it often drove right back into the grass.

Besides improvements to hardware (new GPS, camera...) and general software bug fixes, we decided to implement a new steering system using a local environment map. This map should fuse all known obstacle data (mainly from lidar and camera) and allow the robot to plan its path around obstacles while staying on the roads.

Chapter 4

Design

The topic of our thesis is to create local map of environment the robot operates in, that will fuse obstacle data form sensors. Robot will be able to use this map to plan its path along the road, safely avoiding obstacles and navigating around corners.

This chapter will cover our design of the local map, interpretation of sensor data and navigation algorithms operating over this map. Detailed descriptions will be covered in the chapter 5.

4.1 Local map

Our solution will use a grid based local map. Each cell on the grid will represent a square section from the robots surrounding area. The cells will have assigned numerical values, representing likelihood of being obstructed. As the robot moves, its position in the map will be updated, so that new data can be recorded in relation to previous information.

Since our goal is not to map the whole environment but only to track recent surroundings (for navigation and obstacle avoidance), we decided to

make the map toroidal. Toroidal map lets the system remember recent obstacles in case it needs to backtrack while not requiring large amount of memory. If the robot moves long enough in a given direction, it will eventually start overwriting old data.

4.1.1 Odometry

The movement of the robot will be tracked using wheel rotation sensors. Using known dimensions of the robot (wheel diameter and their distance), the movement of the robot can be calculated from the wheel rotations. The map will store robots position and rotation (in map space) and update it in real time as new rotation data are available.

Inaccuracies if rotation sensor odometry aren't that concerning in our case, because we only need to track robots position over small (but moving) area of the map.

4.1.2 Lidar

Original unit on the front detects obstacles ahead of the robot (forward, 180°, horizontal, 10 cm above ground). This lidar can detect most vertical obstacles (pedestrians, benches, walls) but can struggle with uneven terrain and obstacles that are low to the ground.

As part of hardware improvement of the robot, it should be equipped with a second lidar unit, which will be mounted on top of the robot (360°field of view, mounted 70 cm high). It will be tilted 10°forwards, so that flat ground will be detected at a distance of approximately 4 meters. This should allow it to detect curbs along the road, as well as tall obstacles to the sides and rear of the robot.

4.1.3 Camera

We decided to replace the original camera on the robot with a smartphone to correct previous image quality problems. Nowadays, even affordable Android smartphones have decent camera quality and are easy to program, making them suitable replacement. The smartphone will run Android app, that will capture images and transfer them to the computer.

The main use of camera images is to differentiate drivable and non-drivable surfaces (most often paved road from grassy areas) and help in detecting vertical obstacles (benches, lampposts, pedestrians etc.). A neural network to perform this segmentation task on Smelý Zajko is being developed as part of separate project, so we decided to implement a simpler, statistical approach based on HSV color space.

4.2 Robot navigation

Goal of the local map navigation is to steer the robot along a desired road, while effectively avoiding obstacles. Target direction (azimuth of the road) will be obtained using the global navigation system from the previous algorithm - using GPS to locate the robot in the environment map, finding the best path to destination and determining target headings from road segments. We are considering two different algorithms to calculate the heading the robot should move in at any given point.

The first algorithm evaluates corridors (rectangular areas the robot would drive over when moving in a given direction) placed radially around the robot based on the amount and distance of obstacles. In order to emphasise obstacle avoidance, detections that are closer to the robot influence the value more significantly. The final corridor values are further weighted based on

their angle in relation to the target direction. When the robot moves to the end of the current road segment, the target direction is interpolated between azimuth of the current and the next segment. Direction of the corridor with best value is then returned, to be used by the steering system.

Second algorithm tries to detect drivable surface by finding the rectangle areas aligned with the known road direction. The process works similarly to corridors from the first algorithm but instead of originating from the robot at different angles, the areas are placed parallel to each other in lanes, oriented in the direction of the road. Robot is then steered to drive along the center of the best lane to stay as far as possible from any obstacles. When switching from one road segment to another, lanes are calculated for both of them. Navigation is performed using the lanes from current sector until the following lanes find a suitable road.

Chapter 5

Implementation

This chapter will cover technical aspects of the implementation. Starting with hardware upgrades to sensors and processing power, followed by descriptions of different parts of the local map system and navigation.

5.1 Hardware upgrades

During the development of the local map, Smelý Zajko also went though couple hardware improvements (fig. 5.1). Obstacle detection capabilities were enhanced with addition of second lidar (Slamtec RPLIDAR A2), new GPS sensor (Navilock 8022 with Galileo and Glonass capabilities) improved precision of localization and Nvidia Jetson TX2 (Embedded computing board based on Nvidia Tegra X2 architecture for machine learning applications) boosted neural network capability for camera image analysis. Jetson is connected to the control laptop by ethernet and the communication is handled by ROS.

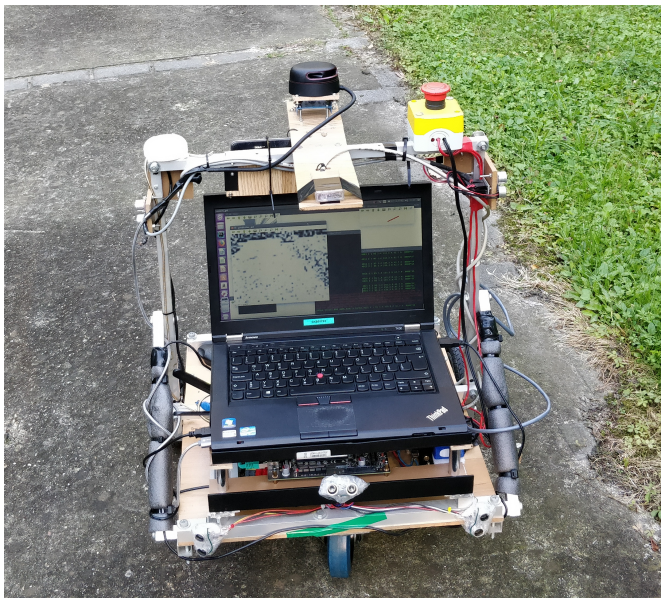


Figure 5.1: Smelý Zajko with upgraded hardware during a testing session. RPLIDAR A2 top center, Navilock 8022 top left and Jetson TX 2 mounted below the laptop

5.2 Local map

The local map will be stored in a two-dimensional array of floating-point values. Each cell represents a square area in the world surrounding the robot and its value is a measure of confidence that the area is obstructed (1 meaning certainly obstructed and 0 free).

Toroidal properties of the map will be accomplished by access functions. When trying to retrieve or change data at a given position (even outside of bounds of the array), the actual index of cell in the array will be calculated using modular arithmetic.

The map will be fixed relative to the world space and robot's pose (position and rotation) will be stored relative to the map. The pose will serve as base point to place sensor data into the map and also to perform navigation calculations.

5.2.1 Lidar data projection

The lidars output obstacle data in form of rays - angles and measured distances. The rays will be filtered, removing values that are too far (empty space) or too close (measurement errors or robot detecting itself), and transformed into the map space - translating and rotating them to account for robots pose and mounting locations of the sensors on the robot. Rays from the angled lidar also need to be projected into horizontal plane.

In order to focus on avoiding collisions, if a map cell would be marked obstructed by lidar, nearby cells are marked as well (see listing B.2). This way the representation of obstacles is larger than necessary, which helps correct measuring inaccuracies. As a side effect this also creates a "buffer zone" around the obstacles, preventing accidental collisions when navigating around them.

5.2.2 Camera data

As its main camera, Smelý Zajko will use an Android smartphone running a custom app. To simplify the app development, we started with an Android Camera2 API sample app[1], which already contained basic camera configuration (setting desired resolution, enabling features such as auto focus etc.).

The basic app was extended with a simple Java server, that broadcasts captured images to all connected clients. Communication between the smartphone and computer is carried over USB, using virtual network that is created by the phone when using USB tethering (sharing phones internet connection with the PC).

Images from the camera will be transformed into HSV color space and

[1]Android Camera2Basic Sample from `https://github.com/googlesamples/android-Camera2Basic`

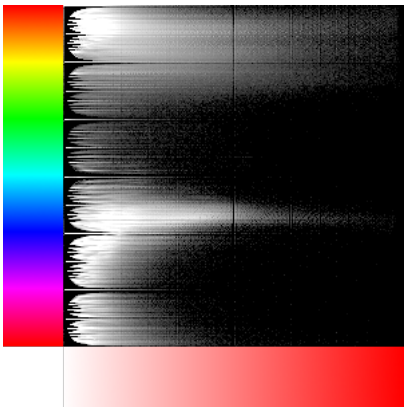


Figure 5.2: Histogram of road surfaces. Hue on vertical axis, saturation on horizontal. Shade of pixels represents occurrence in data set.

then processed by our color analysis algorithm. Resulting greyscale obstacle image need to be further transformed to fit into the local map.

Color analysis

In order to obtain an efficient and reliable segmentation model, we decided to perform color space on 338 sample photos from the park surrounding Castle Lednice (location of the 2018 Robotour contest).

Annotated photos were used to create a histogram (fig. 5.2) based on the HSV color space. Pixels that correspond to drivable surface were binned by hue and saturation (value proved to be of little significance due to shadows). After normalization, the histogram values serve as probabilities that a given hue-saturation pair is road or obstacle.

When analyzing a camera image (fig. 5.3), the algorithm uses the learned histogram values to create a greyscale obstacle image - each pixel is assigned value based on its color.

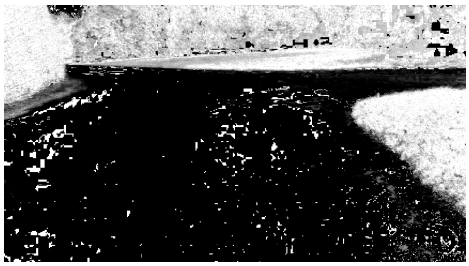
(a) Original image

(b) Oversaturated image

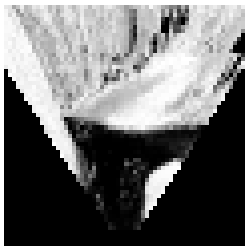
(c) Detected road surface

Figure 5.3: Road detection process. Algorithm processes original image (a) using histogram (fig. 5.2) to detect road surface (c). Oversaturated image (b) shows that the road is mostly red/magenta/blue hue.

(a) Original image



(b) Detected obstacles



(c) Perspective transformation

Figure 5.4: Transformation of image to local map. Original image (a) is evaluated (fig. 5.3), inverted to show obstacles (b) and transformed so that pixels represent local map cells (c)

Perspective transformation

In order to transform the images into obstacle maps (fig. 5.4), they need to be projected onto the map (ground) plane so that they can be fused with other sensor data. We calculated perspective transformation based on the properties of our smartphone camera and its mounting position.

Result of this transformation is a low-resolution image (fig. 5.4c), where each pixel represents single cell in the local map. Values in this image follow the same standard as local map cells (greyscale value from 0 to 1 is equivalent to the measure used to describe obstacles in map) and thus can be easily added to the map using matrix transformations to handle robots position and rotation.

5.2.3 Odometry

The main method of odometry on Smelý Zajko are optical rotary encoders mounted on its wheels. These track rotations of each wheel, and thus the distances each wheel moved. Since Smelý Zajko uses differential drive, we can use these distances to calculate its trajectory.

There are 2 common cases to consider - going straight and pivoting in place (both wheels are moving essentially same distances, but either the same direction or opposite). For these we only need to calculate the distance travelled and angle pivoted respectively.

When the wheels are moving arbitrary distances (and directions), the robot is driving along an circular arc. Since the wheelbase of the robot is known, we can find the center and angle of the arc. Finally we need to calculate the change in position between the beginning and end of the arc and update the robots position in the map.

Implementation of this function is show in appendix B, listing B.1.

5.3 Navigation

Both the radial and parallel approaches to navigation described in section 4.2 work on the same basic principle. First, rectangular areas are created, then they are scored by their obstacle values and finally the best area is selected and the robot is steered accordingly.

5.3.1 Area calculation

The algorithm creates areas based on a origin point, direction, length and width. This approach serves to simulate the robot driving along the given

direction, taking into account its width. The area algorithm operates as follows:

1. Find coordinates of the corner points of area with given properties.
2. Create an axis aligned bounding box of the area in map based on the corner points.
3. Iterate over cells in the bounding box, checking if they belong to the area:
   - Distance of the cell from center line of the area must be lower than width of the area.
   - Distance of the cell from the origin point of the area must be lower than the length of the area.
4. Calculate score for cells that belong and add it to total area score.
5. Normalize the score based on number of cells in the area.

The cell scoring equation 5.1 gives higher score to empty cells and weights them by distance from the origin point. The weighting gives more significance to cells closer to the robot, emphasising avoidance of close obstacles.

$$cellScore = (1 - cellValue) \cdot \left(1 - \frac{cellDistance}{areaLength}\right) \tag{5.1}$$

The score of the entire area needs to be normalized, because areas that are rotated at certain angles can cover different number of cells, which would put them at a disadvantage.

5.3.2 Radial approach

In the radial algorithm, the areas are generated in all directions around the robot (illustrated in fig. 5.5a), representing paths (corridors), the robot could take from its current position. Since the primary goal is to navigate the robot along a segment of the road, the scores of all the corridors are weighted based on their angle relative to target direction (eq. 5.2). This approach lets the robot veer off course to avoid obstacles, while prioritizing movement forwards.

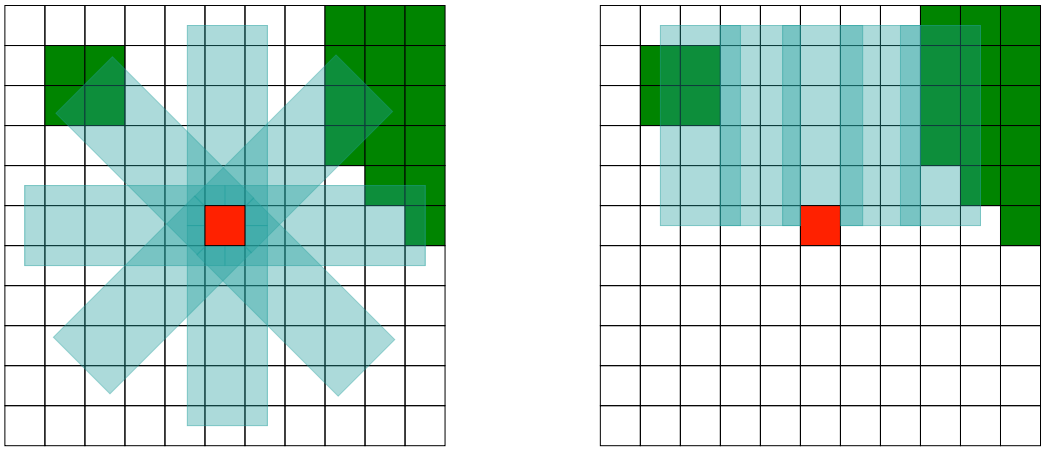
$$areaScore = areaScore \cdot \left(1 - \frac{angleDifference}{\pi}\right) \tag{5.2}$$

Transitions from one segment to another (turns, intersections etc.) are handled by linear interpolation. Once the robot is close to the end of a segment (distance calculated from GPS), the navigation target direction is interpolated between the directions of current and next road segment.

Result of the algorithm is the direction of the best area, which is then used as target direction for the robots steering. In open spaces with no obstacles, the resulting direction will be the same as direction of the road segment (due to angle weighting). If there are obstacles in front of the robot, the corridors that go around them will have better scores, steering the robot around the obstacles.

5.3.3 Parallel approach

The parallel algorithm generates areas aligned with the road segment direction, placed next to each other in front of the robot (fig. 5.5b). This approach weights the areas based on their offset from the robot, prioritizing lanes that are closer. If there is an obstacle in front of the robot, another lane will be chosen and the robot will turn to move to the new lane.



(a) Radial approach

(b) Parallel approach

Figure 5.5: Illustration of the two algorithms. Red grid cell shows robot position, green cells obstacles and blue rectangles the areas created by algorithms

When navigating into next segment, the algorithm creates additional set of lanes for the new direction. These are evaluated independently from the original areas, and if a sufficiently good lane is found the algorithm starts following it. As long as there aren't any good lanes for the new segment, the algorithm supposes that the robot hasn't reached the new road yet, and continues using the old directions.

Chapter 6

Results

In this chapter we will discuss performance of Smelý Zajko on the 2018 Robotour contest and evaluate the algorithms.

6.1 Robotour

Smelý Zajko competed in the 2018 Robotour contest in Lednice, Czech Republic. Goal of the contest was navigating in the gardens of Lednice Castle and delivering a payload. First the robots need to reach given loading point, where a 5 litre keg is loaded. Afterwards they need to travel to unloading point and finally to a finish zone.

The contest conditions proved to be challenging for many robots, including Smelý Zajko. Out of 11 contestants (fig. 6.1), only 4 robots managed to successfully navigate to the loading zone [rob]. Our most successful round ended approximately half-way to the loading point.

One of our more significant problems arose from construction of Smelý Zajko. The wheels are fairly narrow (approximately 4cm), which proved a problem on gravel walkways in Lednice Castle gardens and especially on

Figure 6.1: Robots that took part in Robotour 2018 [rob].

slopes. All 4 rounds of the contest were unfortunately starting from the same location where driving over slope was required, which proved to be too challenging for our robot. Wheel traction was acceptable during testing runs before the contest, but light rain and pedestrian traffic during the day degraded the surface. Smelý Zajko often experienced wheel slipping, which made odometry almost completely unreliable. The bumpy terrain further caused problems for lidar obstacle detection, detecting ground surface due to the robot rocking while moving.

Second major issue was caused by the camera image system, which wasn't properly tested before the contest. Analyzed images contained a lot of different visual artifacts, that made the image very noisy and unreliable. After further analysis we identified the cause of artifacts as inaccuracies in floating-point calculations used to transform camera images into HSV space. These aritfacts showed up when running the HSV transformation on the Jetson

TX2, but not during development on PCs. This led us to believe the artifacts were caused by different floating-point math implementation on hardware level (as our development computers run on x64 architecture and Jetson on ARM).

During the contest, only the radial approach to local map navigation was implemented, and it was also affected by these issues. Reliable odometry data are crucial to keep the map synchronized as the robot moves, and camera images are the main way of distinguishing road surface from grassy areas.

Apart from the technical issues, the radial navigation algorithm worked well. Appendix A shows visualization of the local map during one of testing runs.

6.2 Algorithms

The radial approach to navigation proved to be decent at avoiding obstacles such pedestrians, it had harder time navigating through sharp turns. This is caused by the target direction interpolation when changing segments, which is reliant on accurate GPS position. Park ways can often be only couple meters wide, as is the maximum accuracy of GPS. In worst case, the algorithm performs the turn too early or too late due to GPS inaccuracy, causing the robot to face the edge of the road with little information on how to proceed.

The parallel algorithm fairs better when navigating through turns, but the lane changing approach to obstacle avoidance struggles with obstacles that are close by (which is often the case when it comes to pedestrians or other robots). Creating a new algorithm, that will combine the obstacle avoidance of radial corridors with sharp turn handling of lanes should result in a more reliable navigation system.

Chapter 7

Conclusion

Smelý Zajko went through many hardware and software improvements. Localization and obstacle detection was improved with new GPS sensor and lidar. Camera system was replaced, letting the robot use regular Android smartphone instead of dedicated camera, and the image processing power was improved with addition of Nvidia Jetson TX2 embedded computer.

The old navigation logic was replaced with a new local map, capable of storing data from multiple sensors and deducting the best course of action based on the detected obstacles. Although the current versions of the navigation algorithms aren't completely reliable, they could be merged into one improved algorithm.

There are also further plans to improve Smelý Zajko, outside of this thesis. A new, dual lens Stereolabs ZED Mini camera can be mounted and used for optical odometry, as well as provide point cloud data to improve obstacle detection. The Jetson TX2 board is capable of running more than just our color-based image analysis and could be used to integrate deep neural networks into the control logic of the robot.

Even though Smelý Zajko wasn't able to complete the objective of the

2018 Robotour contest, but the improvements it went through were significant. It's very likely that it will take part in coming years of the contest, and maybe even end up winning it again.

Appendix A

Local map visualization

Robot is located in the center of the visualization (figure A.1), the black arrow showing its current heading (small red arrow is compass north for reference purposes). The solid green areas are tall grass around the road, detected by lidar. Light green cone in front of the robot is projection of camera data, showing false obstacles caused by image artifacts. Blob of blue points around the robot visualizes values for corridors in all directions - the further they are the better.Shorter blue arrow shows target direction (current road segment), which influences angular weighting of the scores. The chosen direction (best corridor) is displayed by the big red arrow.

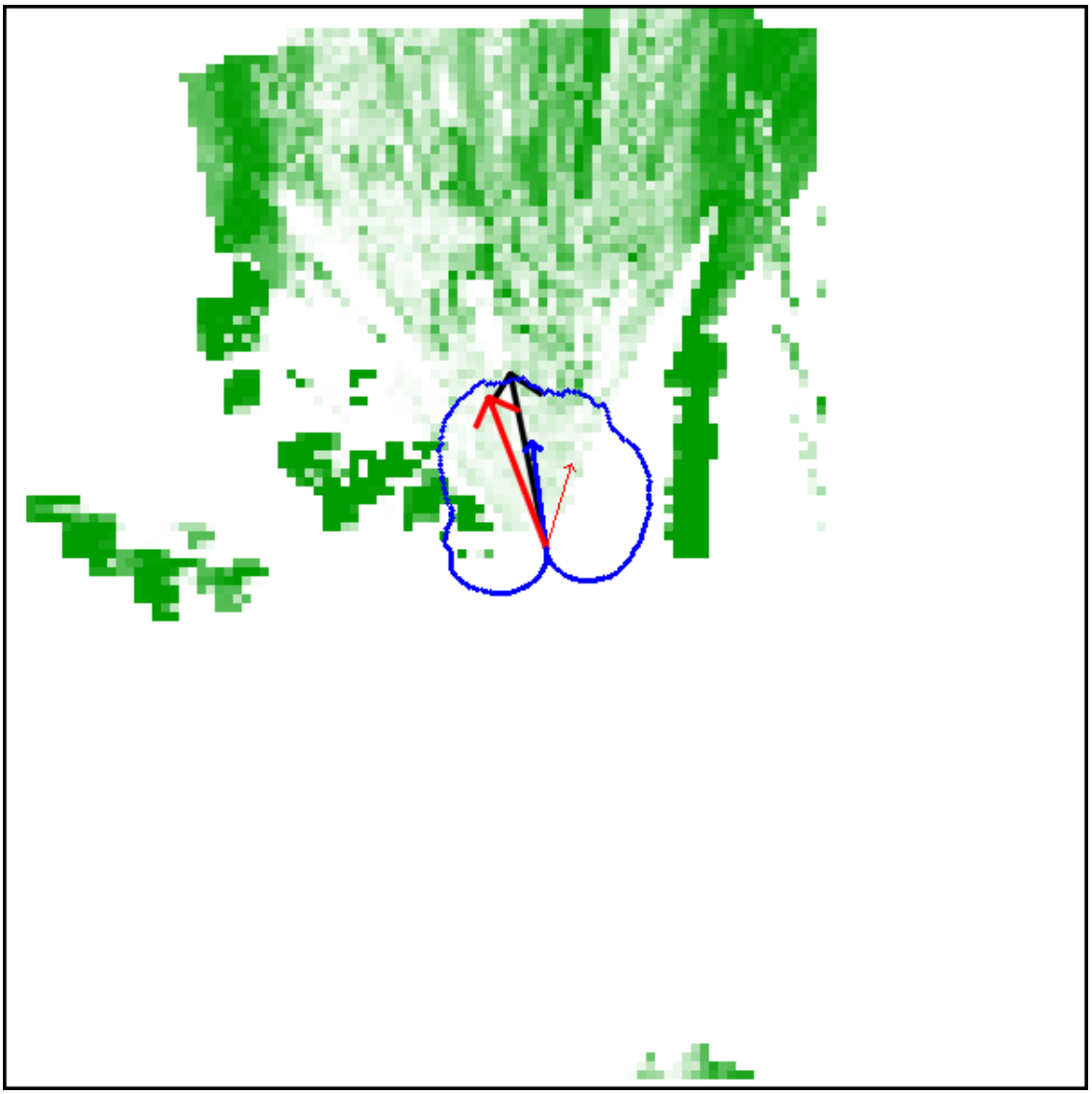


Figure A.1: Visualization of local map and radial algorithm during Robotour.

Appendix B

Code listings

This appendix contains listings of some key functions from the local map algorithm. Complete source code can be found in the project's GitHub repository: `https://github.com/Robotics-DAI-FMFI-UK/smely-zajko-ros/`

Listing B.1: Calculation of new pose from wheel rotations

```
void LocalMap::updateRobotPosition_(long L, long R, bool force) {
    L = -L;
    R = -R;
    double dL = wheelCircumference * (prevTicksL - L) / ticksPerRotation;
    double dR = wheelCircumference * (prevTicksR - R) / ticksPerRotation;

    // don't update on small changes
    if (!force && (fabs(dL) + fabs(dR) < minUpdateDist)) return;

    prevTicksL = L;
    prevTicksR = R;

    double d = (dL + dR) / 2;
    double turnRatio = 2;
    if ((fabs(dL) >= 1.0) || (fabs(dR) >= 1.0)) {
        turnRatio = dL / dR;
    }
    double newX, newY, newAngle;
```

```
    if (fabs(turnRatio - 1.0) < straightMovementThreshold) {
        // straight
        newX = posX + d * sin(angle);
        newY = posY + d * cos(angle);
        newAngle = angle;
    } else if ((dL * dR < 0) && (fabs(fabs(dL) - fabs(dR)) < 0.3)) {
        // rotate along center
        newAngle = angle + dL / (2.0 * wheelDistance);
    } else if (dL != dR) {
        // circular trajectory
        int centerRight = 1;
        double r1;
        if (fabs(dR) > fabs(dL)) {
            centerRight = -1;
            r1 = wheelDistance * dL / (dR - dL);
        } else {
            r1 = wheelDistance * dR / (dL - dR);
        }
        double r = r1 + wheelDistance / 2.0;
        double beta = d / r;
        double cX = posX + r * sin(angle + centerRight * pi / 2.0);
        double cY = posY + r * cos(angle + centerRight * pi / 2.0);
        newX = cX + r * sin(angle - centerRight * pi / 2.0 + centerRight *
            beta);
        newY = cY + r * cos(angle - centerRight * pi / 2.0 + centerRight *
            beta);
        newAngle = angle + beta * centerRight;
    } else {
        // not moving
        newX = posX;
        newY = posY;
        newAngle = angle;
    }
    // normalize new position data into map
    newX = newX > mapWidth? newX - mapWidth : newX;
    posX = newX < 0? newX + mapWidth : newX;
    newY = newY > mapHeight? newY - mapHeight : newY;
    posY = newY < 0? newY + mapHeight : newY;
    newAngle = newAngle > 2*pi? newAngle - 2*pi : newAngle;
    angle = newAngle < 0? newAngle + 2*pi : newAngle;
}
```

Listing B.2: Recording one lidar ray into the map

```
void LocalMap::applyRay(double sensorX, double sensorY, double rayAngle,
    double rayLen) {
    // ray end point
    double rayX = sensorX + rayLen * sin(angle + rayAngle);
    double rayY = sensorY + rayLen * cos(angle + rayAngle);

    // end point in grid
    int gX = map2gridX(rayX);
    int gY = map2gridY(rayY);

    // mark as obstacle (also mark 8-neighborhood)
    matrix[clampGridX(gX-1)][clampGridY(gY-1)] = (matrix[clampGridX(gX-1)][
        clampGridY(gY-1)] + 2) / 3;
    matrix[clampGridX(gX-1)][gY] = (matrix[clampGridX(gX-1)][gY] + 2) / 3;
    matrix[clampGridX(gX-1)][clampGridY(gY+1)] = (matrix[clampGridX(gX-1)][
        clampGridY(gY+1)] + 2) / 3;
    matrix[gX][clampGridY(gY-1)] = (matrix[gX][clampGridY(gY-1)] + 2) / 3;
    matrix[gX][gY] = (matrix[gX][gY] + 2) / 3;
    matrix[gX][clampGridY(gY+1)] = (matrix[gX][clampGridY(gY+1)] + 2) / 3;
    matrix[clampGridX(gX+1)][clampGridY(gY-1)] = (matrix[clampGridX(gX+1)][
        clampGridY(gY-1)] + 2) / 3;
    matrix[clampGridX(gX+1)][gY] = (matrix[clampGridX(gX+1)][gY] + 2) / 3;
    matrix[clampGridX(gX+1)][clampGridY(gY+1)] = (matrix[clampGridX(gX+1)][
        clampGridY(gY+1)] + 2) / 3;

    // mark points on ray as empty
    for (int j = 0; j < rayLen; j += 10) {
        double p = ((double) j) / rayLen;
        double pX = sensorX + p * (rayX - sensorX);
        double pY = sensorY + p * (rayY - sensorY);
        int gX = map2gridX(pX);
        int gY = map2gridY(pY);
        // don't overwrite obstacles found by previous rays from this batch
        if (matrix[gX][gY] < 1.0) matrix[gX][gY] = matrix[gX][gY] / 3;
    }

}
```

Bibliography

[AGLL12] Jose M Alvarez, Theo Gevers, Yann LeCun, and Antonio M Lopez. Road scene segmentation from a single image. In *European Conference on Computer Vision*, pages 376–389. Springer, 2012.

[Bai02] Tim Bailey. *Mobile robot localisation and mapping in extensive outdoor environments*. Citeseer, 2002.

[Cho05] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[CK97] Kok Seng Chong and Lindsay Kleeman. Accurate odometry and error modelling for a mobile robot. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 2783–2788. IEEE, 1997.

[Duc12] F Duchon. Lokalizácia a navigácia mobilnỳch robotov do vnútorného prostredia. *Bratislava: Vydavatelstvo STU*, 2012.

[GM02] Giuseppina C Gini and Alberto Marchi. Indoor robot navigation with single camera vision. *PRIS*, 2:67–76, 2002.

[GOH93] J Gonzalez, A Ollero, and P Hurtado. Local map building for

mobile robot autonomous navigation by using a 2d laser range sensor. *IFAC Proceedings Volumes*, 26(2):835–838, 1993.

[NNB04] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. Ieee, 2004.

[PPU02] Stefano Panzieri, Federica Pascucci, and Giovanni Ulivi. An outdoor navigation system using gps and inertial platform. *IEEE/ASME transactions on Mechatronics*, 7(2):134–142, 2002.

[rob] Robotour 2018 results. `https://robotika.cz/competitions/robotour/2018/results/en`. Accessed: April 30, 2019.

[TB96] Sebastian Thrun and Arno Bücken. Learning maps for indoor mobile robot navigation. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1996.

List of Figures