ISSN 1213-2365

Autonomous flipper control for a tracked robot using 2D vertical laser scan

Filip Sedláček

{sedlafi2,svobodat}@fel.cvut.cz

CTU–CMP–2012–11

May 25, 2012

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/svoboda/Sedlacek-TR-2012-11.pdf

Thesis Advisor: Tomáš Svoboda

The work was supported by EC project FP7-ICT-247870 NIFTi
and by the Czech Science Foundation under Project P103/10/1585

Research Reports of CMP, Czech Technical University in Prague, No. 11, 2012

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Filip S e d l á č e k
Study programme:	Cybernetics a Robotics
Specialisation:	Robotics
Title of Bachelor Project:	Free-form Object Detection and Measurement by Using LIDAR and Omnidirectional Camera

Guidelines:

Design and implement a module within ROS (Robot Operating System, ros.org) environment for measuring scene features in mobile robot operating space.

The module will use mainly 3D point data acquired by a rotating laser range finder. It may also use images from the omnidirectional camera which is also mounted on the robot. The locomotion of the NIFTi robot allows for moving in a difficult terrain both outdoor and indoor. However, steering the robot in such a difficult terrain is very demanding for its remote operator. The new measuring module should reduce the operator load by providing data for a selfconfiguration module. Ideally, the operator would steer the robot by just forward, backward, turn commands without the current necessity of frequent reconfiguring the robot locomotion. Important values to be measured are e.g. width of narrow passages or height of stair steps.

Bibliography/Sources: Will be provided by the supervisor.

Bachelor Project Supervisor: Ing. Tomáš Svoboda, Ph.D.

Valid until: the end of the winter semester of academic year 2012/2013

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.
-6-

[illegible]

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 13, 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Filip S e d l á č e k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Detekce a měření objektů pomocí laserového dálkoměru a všesměrové kamery

Pokyny pro vypracování:

Naprogramujte modul do prostředí ROS (Robot Operating System, ros.org), který bude měřit parametry určitých oblastí v operačním prostoru robota.

Modul bude vyžívat data získaná rotujícím laserovým dálkoměrem a obrazu z všesměrové kamery. Pásový robot NIFTi je způsobilý k pohybu v náročném terénu, například vyjede i strmé točité schody. Řízení v takto obtížném terénu ovšem klade vysoké nároky na operátora. Nový modul by měl zvýšit autonomii robota například na schodech. Po změření charakteristických rozměrů schodů by mohl robot automaticky změnit konfiguraci tak, aby ho operátor řídil pouze na úrovni pokynů vpřed a vzad. Další důležitým rozměrem je je šířka zúžených míst, například dveří či průjezdu mezi velkými objekty, např. auty.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Tomáš Svoboda, Ph.D.

Platnost zadání: do konce zimního semestru 2012/2013

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

L.S.
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-

[illegible]

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 13. 12. 2011

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 25. 5. 2012 .... Sedláček

Acknowledgement

I thank Tomáš Svoboda for patience, willingness to explain everything and documenting progress and results, Vladimír Kubelka for the simple idea that the rear flippers angle can be the same as that of the front flippers, which allowed this work to become practically usable, Tomáš Petříček, for ROS-related C++ information, Michal Reinstein and all members of the CMP lab for friendly support.

Abstract

This work describes a Robot Operating System (ROS) module for automatic flipper adaptation during obstacle traversal with the tracked NIFTi robot.

This module is intended to relieve the robot operator from the difficult task of remotely controlling all four flippers by adjusting the flippers automatically. The operator may decide to switch on the automatic mode part of the time for example during a staircase traversal.

The algorithm does not model the terrain explicitly. It measures the terrain profile by a vertically positioned 2D laser scanner mounted on the front of the robot. The main goal is to achieve the flippers touch the ground in order to maintain robot's stability.

The proposed algorithm has been successfully tested on traversing staircases, curbs, rails and various other obstacles of different shapes.

Abstrakt

Práce popisuje programový modul pro Robot Operating System (ROS), který zajišťuje automatickou konfiguraci přídavných pásů (tzv. flipperů) pásového NIFTi robota během přejíždění různých překážek.

Tento modul má za úkol usnadnit obtížné manuální ovládání flipperů operátorem řídícím robota tak, že tyto flippery nastavuje automaticky. Operátor si automatický režim může zapnout například během vyjíždění do schodů.

Algoritmus překážky explicitně nemodeluje. Měří profil terénu 2D laserovým dálkoměrem přepevněným vpředu na robotovi tak, že laserová měřící rovina je vertikální. Cílem algoritmu je nastavovat flippery, aby se dotýkaly země a zůstala zachována stabilita robota.

Algoritmus byl testován při přejíždění schodů, obrubníků, kolejí a dalších překážek.

Contents

Chapter 1

Introduction

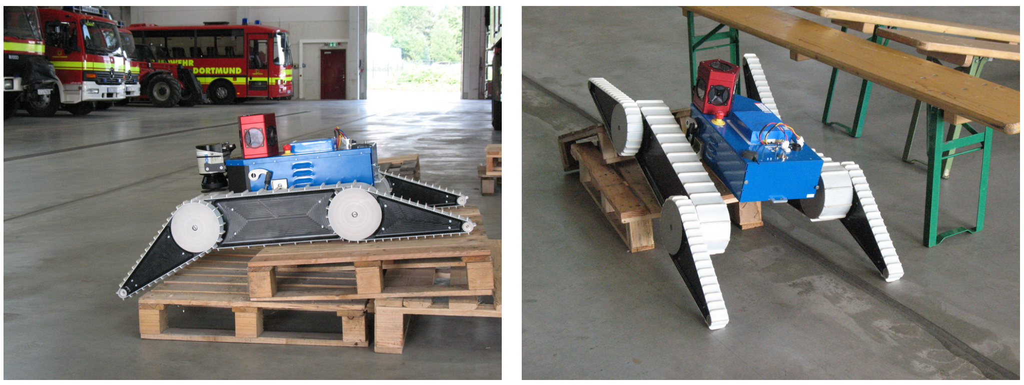

Figure 1.1: The NIFTi robot.

This work describes an algorithm for our robot, that makes easier the task of driving the robot across obstacles such as staircases during teleoperation.

There are robots with different constructions designed with traversing obstacles in mind. We work with the NIFTi robot, which is a small, tracked search-and-rescue robot designed to move in difficult rough terrain. The NIFTi project [1] works with this robot platform and investigates how human-robot teams can work together to explore a disaster area (i.e. car accident in a tunnel, railroad accidents or areas with chemical or radioactive contamination), to assess the situation and locate victims.

The NIFTi robot may be teleoperated – that means remotely controlled with the operator having only data from the robot's sensors (i.e. an image from on-board cameras or 3D scans of the surroundings from a laser scanner). Focus in the NIFTi project is to make the robot more autonomous in the future.

Our robot has main tracks and additional four reconfigurable tracked arms called flippers. Position of these flippers must be set manually during teleoperation. This – especially for an unskilled operator – is quite slow and the robot operator has to focus his entire attention to the demanding task of setting all the four flippers to the correct angle assuring safe navigation. Decreasing the time for example of a staircase climb is an important factor in an urban search and rescue mission.

This work describes an algorithm, by which the robot can configure its flippers autonomously and the operator controls only the main tracks speed, i.e. commands the robot to go forward and controls its direction.

Chapter 2 State of the Art discusses existing methods of climbing staircases autonomously with similar robots and also traversing a completely unstructured terrain. Chapter 3 The NIFTi robot describes the robot itself in greater detail and shows necessary flipper reconfigurations during a step-climbing task. Chapter 4 Exploring the laser scanner data describes an initial exploration of the data coming from the on-board laser scanner. It contains visualizations and comments about data quality. Chapter 5 2D vertical laser scan method describes the algorithm present in the working NIFTi robot's autonomous flipper control module. Its implementation is described in Appendix A. Finally, in the chapter 6 Evaluation and experimental results, we show results in the field with the presented algorithm implemented on the NIFTi robot.

Chapter 2

State of the Art

Research for and comments to existing methods of solving the staircase climbing and obstacle traversal task with a tracked robot.

2.1 Autonomous staircase climbing

The need to help a robot teleoperator during staircase climbing was here as far back as in 1994. Martens and Newman [2] were using a tracked robot Andros Mark VI, whose construction is now quite obsolete. They implemented a simple control system utilizing two ultrasound sensors and an accelerometer (measuring inclination of the robot with gravity as a reference) for maintaining the heading direction perpendicular to the steps of the staircase, which they identified as a requirement for a safe climb. This system assisted the operator and decreased the time required to perform the task. They concluded that the risk to the robot was reduced as well as the stress placed on the operator.

Point cloud segmentation approach with staircase measurement was presented for example by Sanchez and Zakhor [3] in 2012. They primarily focused on model fitting into point clouds of interior structures like walls and ceilings, but they also proposed a staircase model with six parameters (e.g. *tread depth*, *riser height* and *number of steps*). Their algorithm however employs point-wise Principal Component Analysis, which makes the algorithm run in the order of hours. It is definitely not a real-time solution we would like.

Point cloud segmentation for the purpose of stair-climbing was done by Osswald et al. [4] in 2011. They evaluated current state-of-the-art methods of segmenting a point cloud into planes. They built 3D models of complex staircases based on laser range data acquired with a humanoid. Their models were sufficiently accurate and enabled their robot to climb up the staircase. They used a scan-line grouping, an extremely fast algorithm.

A recent and comprehensive paper on the staircase climbing task is Zhang et al. [5]. They successfully implemented an entirely autonomous stair-climbing module for a modern robot PackBot, which has tracks and front flippers. They used a vertical laser scanner for distant staircase detection and then a bunch of sensors including sonar, IR proximity sensors and a gyroscope during the actual climb. As was also common in other papers, the staircase climbing procedure was divided into stages including *Detection and Approach*, *Climbing* and *Landing*. This is not consistent with our goal since such algorithm would be limited only to staircases – an quite likely not even all of them, because the authors make assumptions about the staircase parameters. We want to control the flippers automatically possibly for a wider set of obstacles.

Two papers from 2009 written by Mihankhah et al. [6], Kalantari et al. [7] describe staircase climbing with a modern rescue robot ResQuake, which has main tracks and four flippers with a little different construction than our robot. Their goals were to find the staircase and to develop a fast, safe and smooth autonomous stair climbing algorithm. The robot had two laser scanners: one mounted horizontally and the other vertically. They implemented a fuzzy control system (and later a kinematics-based controller) for keeping the robot away from stairway sides and to keep the robot body parallel to stairway sides.

The following papers deal with the autonomous stair-climbing task by using a camera and a gyroscope or accelerometers. Steplight et al. [8] in 2000 climbed staircases with the Urbie robot. They fused measurements from a sonar, a camera, and two accelerometers for attitude estimation. Xiong and Matthies [9] steers the robot in 2000 on a staircase using vision only. Later, Helmick et al. [10] in 2002 extend it by fusing 3D attitude information from gyroscope measurements, vision, and a laser scanner. The laser scanner is oriented horizontally, giving information about the surrounding walls and is thus important for keeping the robot's heading angle small. The same approach is further extended in a long paper written in 2007 by Mourikis et al. [11] who used a modern PackBot. Similar, a purely vision-based approach is presented by Cong et al. [12] in 2008. They estimate the position of UGV robot on a staircase and the orientation angle to stairs through robust extraction of the stair edges.

For example Helmick et al. [10] in their 2002 paper demonstrate a working solution to the staircase climbing problem, but they note a minor flaw in their conclusion – they do not slow down at the upper end of the staircase, so their robot lands quite roughly. This is precisely the situation, which we want to avoid. In our case, we do not want the robot to drive upstairs full speed and then slam its front part down. After the robot tips over on the last step of the staircase, the front flippers must be deployed under the robot to mitigate the impact. This was for example done later by Zhang et al. [5] in their 2011 paper.

The papers usually dealt with autonomous staircase traversal, where the main concern was to keep the heading angle as small as possible and to keep a sufficient distance from the walls of the staircase. Main focus of the algorithms was to constantly estimate values of these two parameters, which allowed a controller to drive the robot autonomously upstairs. In our case, we do not aim for a fully autonomous drive in such way.

Contrary to most papers dealing with the staircase climbing problem, we set the flippers angle as a direct mathematical function of the terrain profile. Others have usually been setting the flippers to some manually chosen stable angle specific for each stage of the staircase-climbing procedure.

2.2 Flipper control on an unstructured terrain

These papers describe methods for controlling the flippers autonomously and therefore lowering the work-load of the operator for maneuvering the robot on an unknown and *unstructured* terrain. These methods build on sensor information in real-time and enable a semi-autonomous operation in rubble (debris) environment found for example in areas with collapsed buildings.

Ohno et al. [13] is the first to propose in 2007 a semi-autonomous control of a robot with flippers in an unknown environment. The authors designed a sequence of motions for getting over an unknown upward and downward step. They also dealt with adjustment of the robot speed. The robot used an accelerometer, current sensors and distance sensors.

In 2008 Nagatani et al. [14] with a similar robot KENAF proposed another strategy of simple sensor-based control of flippers independently on both sides. They used two laser scanners whose scan planes are set in parallel to the workspaces of the active flippers and perpendicular to the ground for terrain sensing, and gyro sensors for the measurement of the robot's attitude. Their algorithm is very similar to the one used in our work. In the end, authors thought about stability and ended with a question whether more complex control would be effective or not.

Two laser scanners positioned the same as in the previous work (and on the same robot) were used by Okada et al. [15] in 2009. They had more complicated flipper position computation and they tested a desired posture for its stability before the flippers were physically set. They successfully tested it during slow speed obstacle traversal (5-10 cm/s) on a pile of concrete blocks.

Chapter 3

The NIFTi robot

The NIFTi robot is a 60 cm wide, approx. 40 cm high tracked robot. Its length depends on the position of additional front and rear tracked arms called flippers. When the flippers are fully deployed and touch the ground, the total length of the robot is 120 cm. When the flippers are in an upright position or retracted parallel to the main tracks in a horizontal position (this configuration is used for transportation of the robot), the robot length is determined by the length of the main tracks, which is nearly 70 cm.

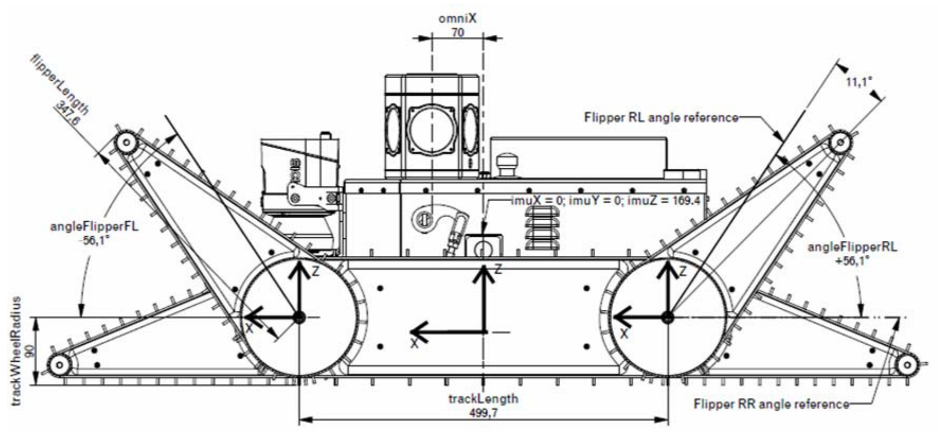

Figure 3.1: A side view of the NIFTi robot – a technical drawing.

Purpose of the flippers is to enhance the robot's mobility. The flippers allow the robot to climb a staircase, traverse a higher step, deal with shallow gaps in the ground or to cross easily railway tracks. This has been tested with the developed automatic flipper control and it will be shown later. Flipper angles can be actuated independently, but the track speed is uniform on each side. Driving with the robot while its weight is purely on the flippers is forbidden according to the manual, its weight must always rest mostly on the main tracks.

This configuration (sometimes only with the front flippers) has been used with successful robots like versions of Urbie robot [8, 9, 10], modern PackBot [5, 11], UGV [12], KENAF [14], [15], ResQuake robot [6, 7] and others.

By zero flipper position we mean that the flippers touch the ground, i.e. they are in the lower position on the technical drawing 3.1). A positive flipper angle means flipper up, a negative angle flipper down, under the robot body (this may actually be opposite in the ROS system, see A Implementation).

The robot does not have any touch sensors which could be used to sense whether the flipper has touched the ground.

3.1 Traversing obstacles

When driving on a flat ground, the flippers are usually in an upward position and additionally positioned in such way that they do not obstruct view of the camera or the laser scanner, see picture (1) in Figure 3.2. The flippers are used only during obstacle traversal to enable climb-up and prevent the robot from falling over.

Figure 3.2 shows the procedure necessary to climb up an obstacle with a sharp edge, e.g. a curb. First, the operator has to lower the front flippers to enable climb-up to the obstacle. As the robot inclination increases, it also becomes important to lower the rear flippers to support the robot and prevent it from falling back. As the robot reaches stage (4) in Figure 3.2, the center of mass crosses over an unstable point and the robot tilts over to the front. At this point, the front flippers must already be placed down under the robot to eliminate slamming down of the front part, which is dangerous for on-board equipment. During the next stages the robot body should become level again. This is usually achieved by progressively elevating the front flippers on which now the weight of the robot rests. It can be accompanied by lowering of the rear flippers. This is unnecessary and is usually omitted the by human operator, but our autonomous flipper control algorithm is capable of doing this. Finally, the flippers are returned into the original position used for navigating on a flat ground.

The robot has a differential that allows each of the main tracks to tilt separately, which can be visible in Figure 3.3. This differential can be locked, then the main tracks are both fixed in the position shown in the technical drawing 3.1. When traversing obstacles, it may be advantageous to unlock the differential and allow the tracks to fit to the terrain. This increases

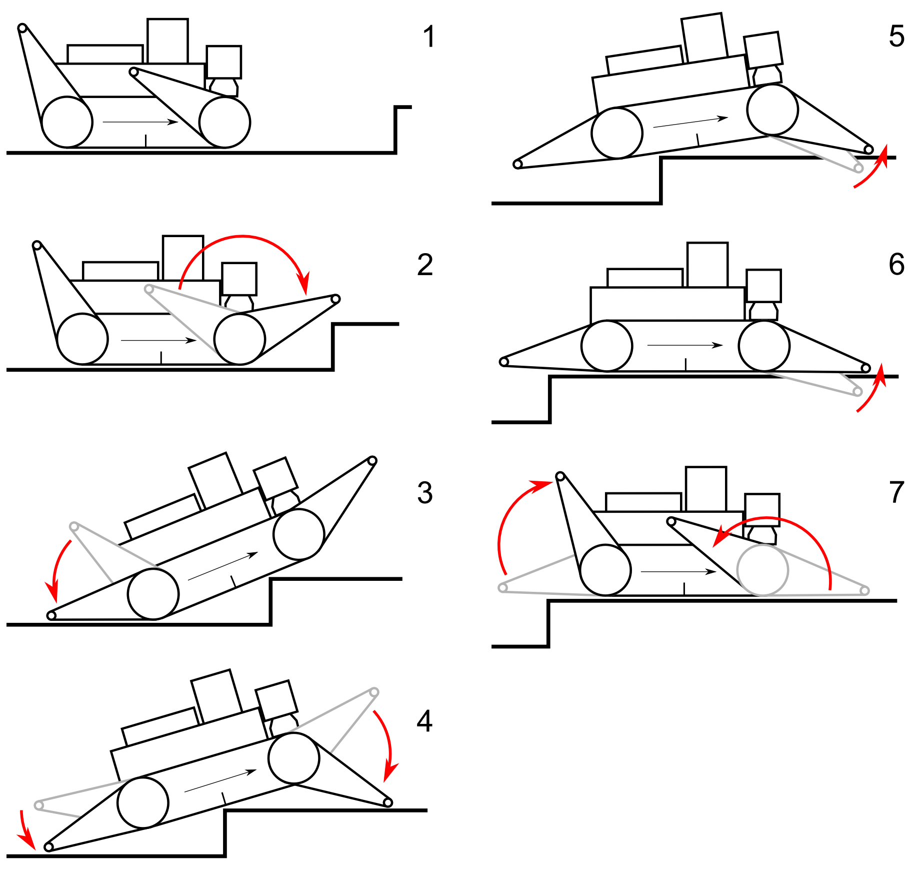

Figure 3.2: One possible version of a step climbing procedure. In each of these steps the operator must reconfigure the robot's flippers.

contact of the tracks with the ground and reduces slippages.

Since our algorithm works only in a single vertical plane in the center of the robot, flippers on both sides are set jointly. When driving with the autonomous flipper mode, it is therefore better to lock the differential, since the algorithm counts on the tracks being both in the position they are in the technical drawing 3.1 and the individual inclines of the main tracks are not taken into account. It is possible to drive with the autonomous flipper mode and the differential unlocked when crossing an obstacle where dissimilar inclination of the tracks occurs minimally.

3.2 The laser scanner

The NIFTi robot is equipped with a rotating laser scanner, an omni-directional camera handful of other sensors.

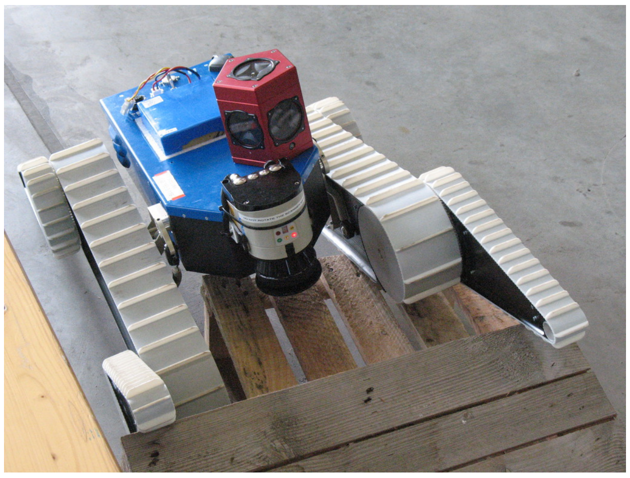

Figure 3.3: The laser scanner is mounted on the front of the robot and can be tilted. This image also shows different tracks inclination due to the unlocked differential.

The laser scanner is a standard 2D (two-dimensional) SICK LMS-100-type range finder mounted on a revolving base. By scanning during its rotation, the laser scanner can scan 3D point clouds. The laser scanner is mounted in the center on the front part of the robot approx. 20 cm above the ground, see Figure 3.3.

The laser scanner works by sending a light pulse and measuring the time till it receives an echo from a surface from which the laser beam has reflected (time-of-flight LIDAR). This laser scanner can measure objects as far away as 50 m, but reliably operates approx. up to 20 m.

The laser scanner will be used for obstacle sensing. The main difference between the laser scanner mounted on the robot and a sight of the operator is that the laser scanner is much closer to the ground, so it does not have such a good view. Further-away obstacles (e.g. steps in a staircase) may be occluded by the nearer ones. This is however not such a problem for our algorithm, since it works only with the nearest data in range of the front flippers. A bigger problem regarding our autonomous flipper task is that the area behind the robot cannot be seen by the laser.

In the next chapter, we analyze the data the laser scanner provides.

Chapter 4

Exploring the laser scanner data

Since the prospective algorithm would be based primarily on data from the 3D laser scanner, the focus was first on the data itself. Is it complete? How precise is it?

4.1 Laser scanner data representations

4.1.1 A point cloud

A point cloud is a set of 3D vertices, i.e. points with (x, y, z) coordinates. These points are intended to be representative of the surface of scene objects. A 3D point cloud is produced in the following way in our case. Our rotating 2D laser scanner produces lines of points, each scanned at a different angle. From these lines a full point cloud is constructed. Currently the robot must not move while it scans all the lines. All points are thus scanned from the same location.

Our point clouds usually contain 50k to 150k points. The number of points in the point cloud (i.e. its density) depends on the speed of the laser scanner rotation, which can be controlled on our robot.

4.1.2 A depth map

A depth map is an image similar to an image from a camera, except that instead of color information, it contains depth (i.e. distance) to the surface in the respective direction. A depth map, alike a photograph, is scanned from one viewpoint. The depth information in each pixel is usually converted to pseudocolors so that a depth map can be drawn and viewed as a normal image. It is easily understandable because it has similar geometry to an optical image. See a comparison in Figure 4.2.

If two points are close in 3D space, they also must be close in the depth map. The opposite is not true: two points close in the depth map can be far away in 3D space, i.e. there is a sudden depth change in the depth map.

Here I use mostly a pseudocolor scheme which displays the nearest points in blue and the furthest-away points in pink (blue-cyan-green-yellow-red-pink). This can be seen as somewhat counterintuitive, since one would expect near objects to be red (i.e. hot) and far objects blue (i.e. cold). An inverted scheme was used since greater distance means greater value and therefore a more intense color. White (background) means no data in these depth maps.

4.1.3 Comparison of a 3D plot and a depth map

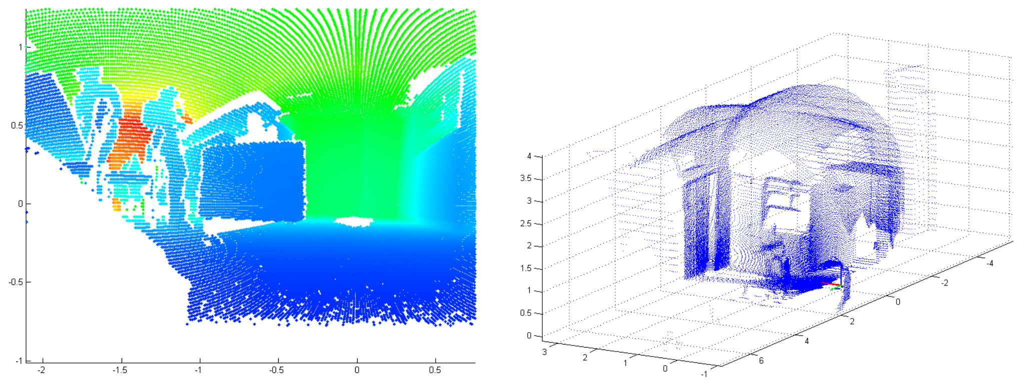

Figure 4.1: (a) A depth map. Colors code distance (depth), axes show angle in radians. We discuss areas with missing color in section 4.2 Laser failure areas. (b) A 3D points plot of the same point cloud. Axes are in meters.

An advantage of the 3D plot of the point cloud is that a human can easily identify flat structures like walls and ceilings whose shape is not clearly obvious from the depth map. The biggest disadvantage is overlapping points making it very confusing. Moreover, the 3D points plot is usually produced from a different viewpoint than from which it was originally scanned. A view on the 3D data from a different position is not completely valid, because if scanned from there, it would contain different points. This is because the point cloud is not a full 3D representation of the surroundings.

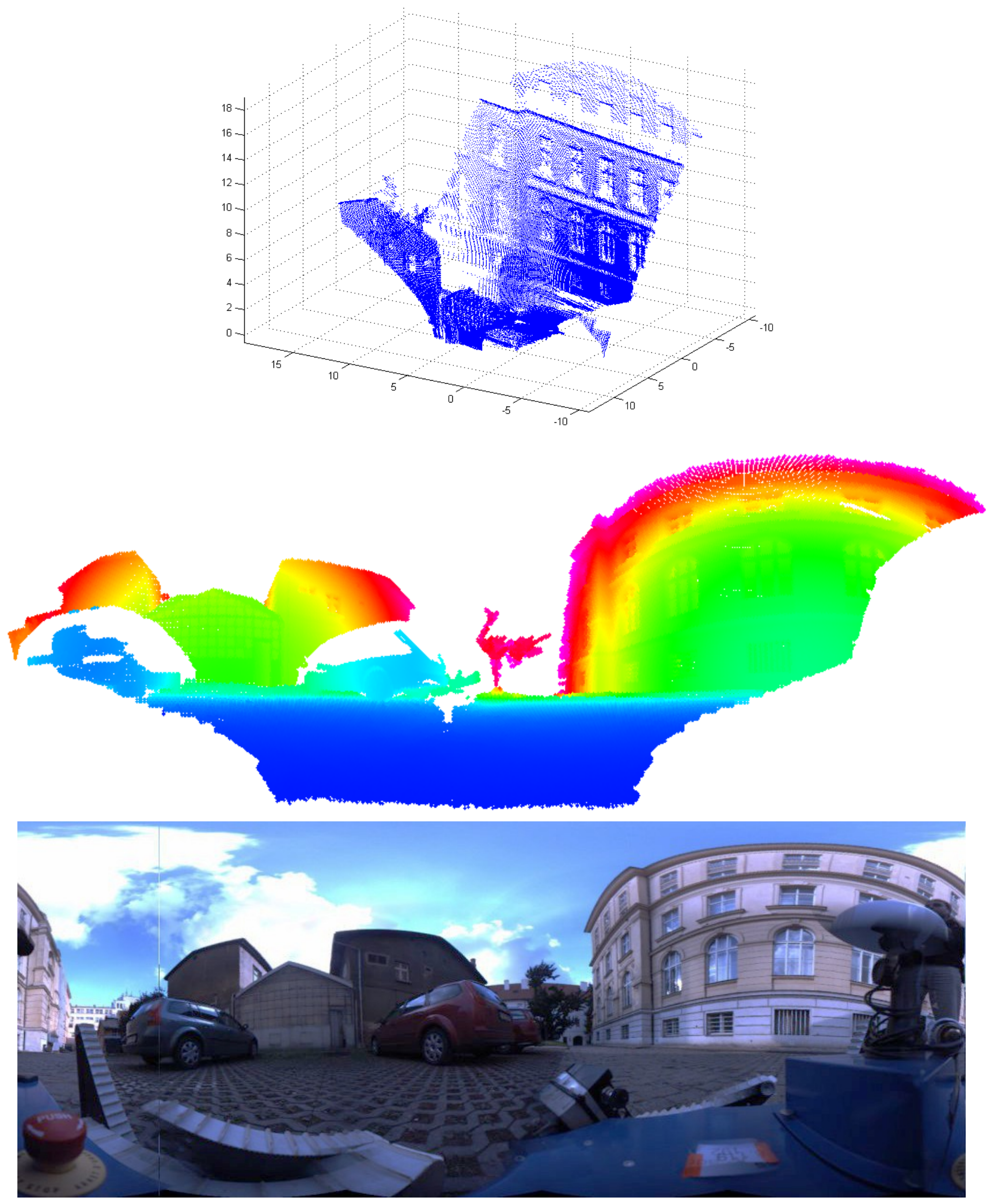

Figure 4.2: An outside scan. (a) This is a good arrangement of the 3D plot showing well buildings walls. (b) Drawing the depth map makes the scene more understandable and shows a problem with cars. (c) A photo of the scanned area. The depth map has a similar geometry to the photographic image to which humans are accustomed.

I found a depth map to be more understandable representation of the point cloud if a human wants to quickly assess what is in the scene. For best understandability of the 3D data, it is possible to combine pseudocoloring according to depth with the 3D plot of the point cloud, as it is done for example in RVIZ visualization program in ROS [16].

4.2 Laser failure areas

The depth map is the best for assessing areas where the laser fails, i.e. where it does not provide any distance information (points). Missing points can mean that the laser range finder did not get an echo from the surface or could not measure the time of arrival of the echo properly.

It turns out, the laser has a problem with shiny and wet surfaces. This can be for example visible in Figure 4.2 b. It shows extensive areas with missing points belonging to the cars.

Next, we progressed toward scanning staircases. Subsequent images show other areas where the laser is failing.

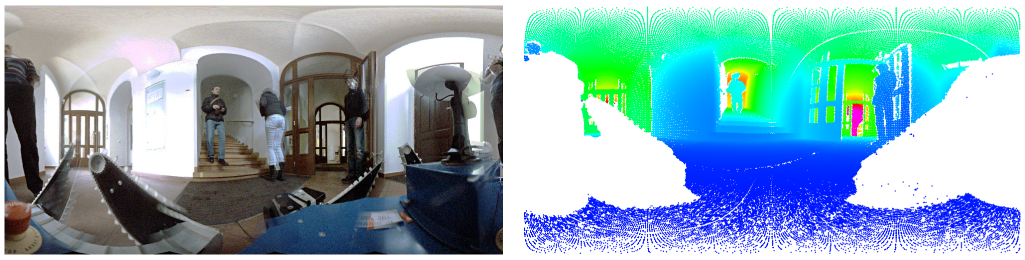

Figure 4.3: A scan of a staircase in the CMP building G. The robot does not stand perpendicularly to the stairs.

Large areas of missing points in the left and right part of the full depth map (Figure 4.3 b) are caused by the robot body occluding the laser scanner field of view. The robot body, strangely, does not reflect the laser (none or only a minimal number of points is present).

The scan in Figure 4.3 does not appear to have any major areas with missing points except one. It is the left wall of the stairway. This wall is physically present there but is almost parallel to the laser ray path. This can cause that the laser to fail to respond on this surface even though it is not a shiny or wet one.

There is a strange effect of missing points on edges of the stairs. The closer to the center, the more points are missing. Missing points on edges are unfortunate for the stair climbing task, because the edges of the stairs are the important structures which we may want to detect and they are also

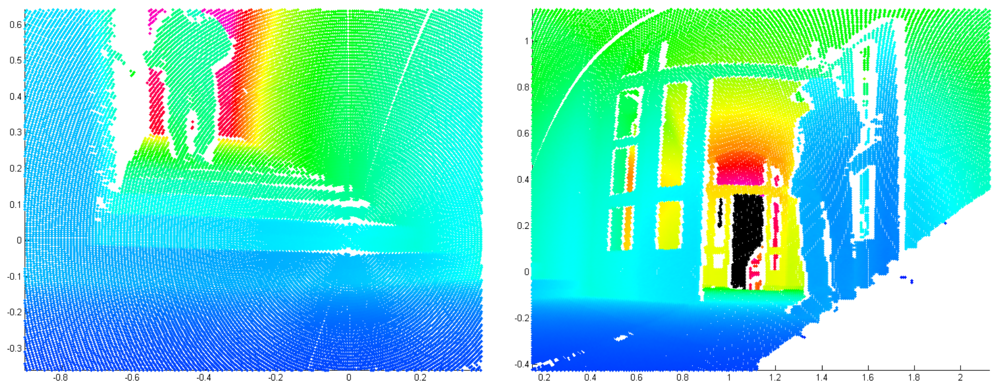

Figure 4.4: (a) A close-up of the center of the staircase depth map from Figure 4.3. Note missing points. (b) This image shows missing points on other edges and on *some* sudden changes in depth. Note that there are glass panes inside the holes in the doors. The laser goes through glass and measures the surface behind it.

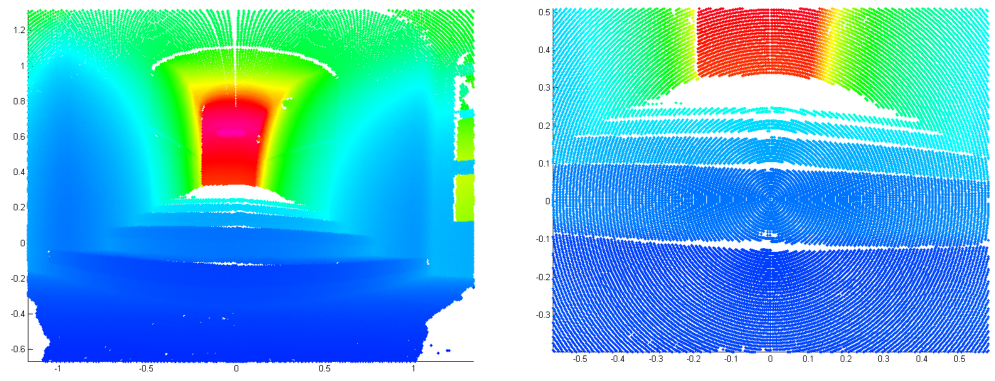

Figure 4.5: Because of simplification of the stair detection task, we may place the robot in front of the stairs so that it stands perpendicularly to the stair edges. (a) Such situation is shown in this image, which is a close-up from a full depth map. (b) Further close-up of the same situation. It shows a large area with missing points. The upper stairs are not visible at all, there is no data. This happens only when the robot stands perpendicularly to the edges of the stairs. It is not clear what causes this. It is possible, that the laser is getting multiple echoes from the further-away steps and cannot measure the distance properly.

the places where the robot is in touch with the stairs while actually climbing them.

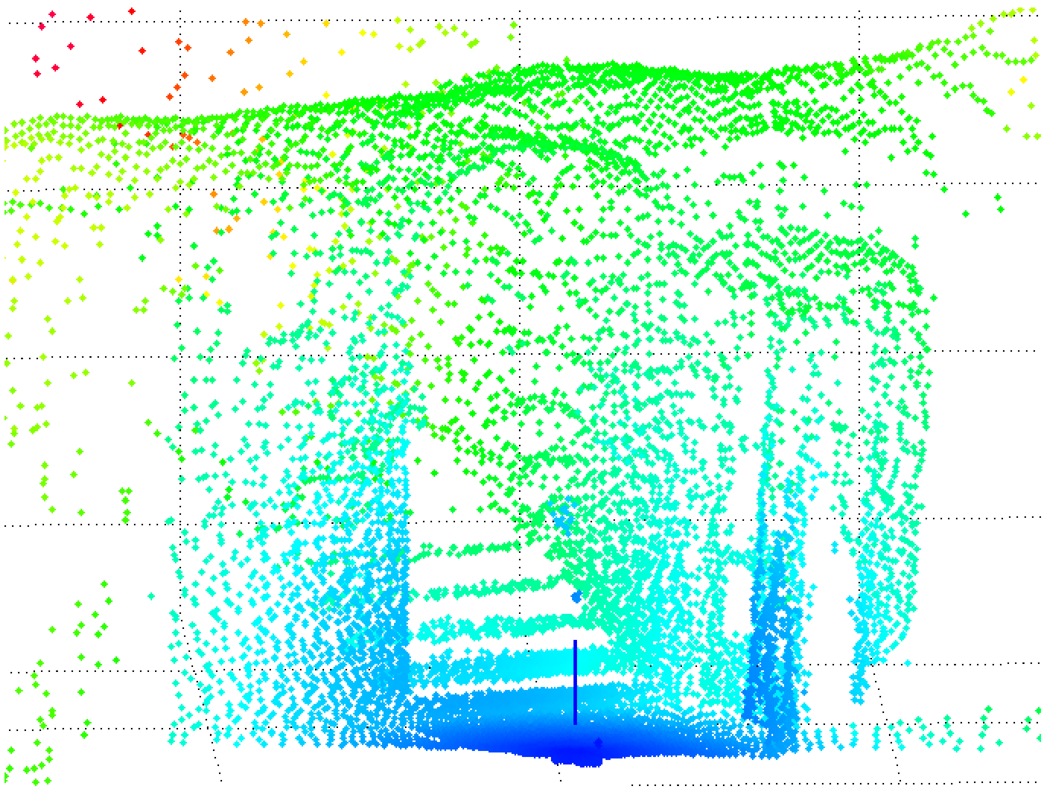
Figure 4.6: An image showing how the data of the scanned staircase actually look like in 3D. There are points belonging to the first to fifth step. Further-away steps have only a few points.

4.3 Conclusion from the data exploration

We gained clearer insight into the data provided by our laser scanner, which could be used as an input to the prospective algorithm.

There was an idea, that the robot would measure the whole staircase from below and count the number of steps for example. We discovered, that this could be extremely difficult, because the laser data (point clouds) does not contain sufficient information. As can be seen in Figure 4.6, the further-away steps are not discernible in the data and even a human can not count the total number of steps in the staircase. Such algorithm would also fail on a very long staircase.

The surrounding walls may be present in the scan, but a staircase does not have to have walls on its sides, it can have a balustrade. Further, since the laser scanner is mounted very low on the robot, the robot only sees the

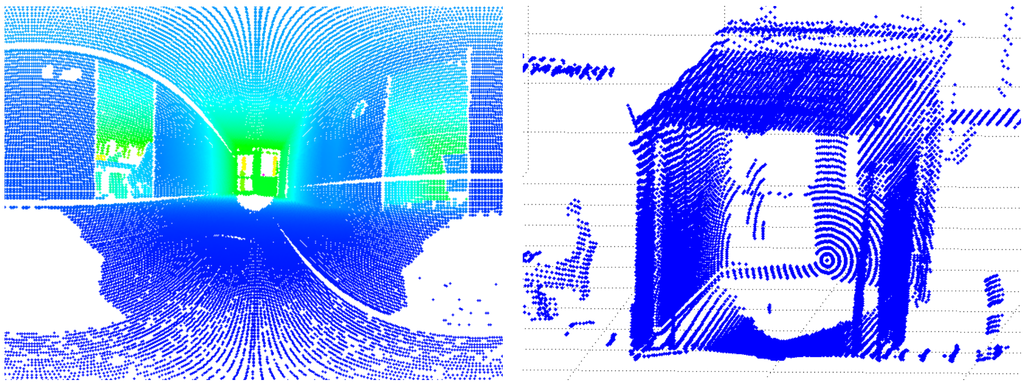

Figure 4.7: The first image shows another problem with missing points, whose cause is unclear. It shows a hallway with adjoining rooms with people. (a) On many depth maps, there appear to be missing points in an almost perfect circle near the center. (b) The second image shows the same scene in 3D, the viewpoint is near to that from which the point cloud was scanned. There is a lot of missing points belonging to the ground. This is unfortunate for obstacle detection, even though obstacles are more likely to get detected than a flat ground.

top horizontal surfaces of only one or two of the first steps of the staircase, assuming it stands underneath it on the floor. All this suggest that a general and at the same time robust model of a staircase would be very difficult to make and it would be difficult to fit data to it, since the data is actually missing a lot of information.

The approach based on point clouds would not produce a working algorithm in the time assigned to this task. The point cloud is very complicated and contains a lot of information completely unnecessary for the obstacle-traversing task. Even bigger problem is that the point cloud cannot be captured frequently enough. It takes a few seconds to capture one point cloud. However the staircase/obstacle-climbing algorithm – whatever it might be – will have to react more quickly and with more recent data.

Chapter 5

2D vertical laser scan method

Our task of automatically controlling the flippers could have been solved by processing the 3D data. Instead of trying to isolate a plane from the 3D point cloud, we decided to use the individual 2D laser scans directly. Everything that has been said about missing points in the previous chapter applies to individual laser scans as well. We care only about data, which could potentially come in contact with the robot's flippers during its subsequent movement. Upon tightening the loop, i.e. running the algorithm more frequently with up-to-date sensory data, it is possible to consider only terrain the robot can touch in its *current position*. In other words, we do not need to look much ahead.

The task does not require to separate objects in the data and measure them. For the robot it is only important, that there is something there, into which it can crash with its flippers. It does not matter to which object it belongs or what are its parameters. The final algorithm is based on an idea circumventing the problem that even if we could measure parameters of the obstacle, we would also need precise and more importantly *instant* information about the relative pose of the robot towards that obstacle.

5.1 The algorithm

We propose to focus on the terrain profile. The terrain profile in front of the robot can be measured by the laser scanner in a vertical (scanning plane) position, see figure 5.1.

Individual laser scans are up-to-date, frequently available and contain almost no erroneous points. There will always be accurate data from in front of the robot. The front flippers may be a little bit more important to control properly than the rear flippers, as there are robots, which has

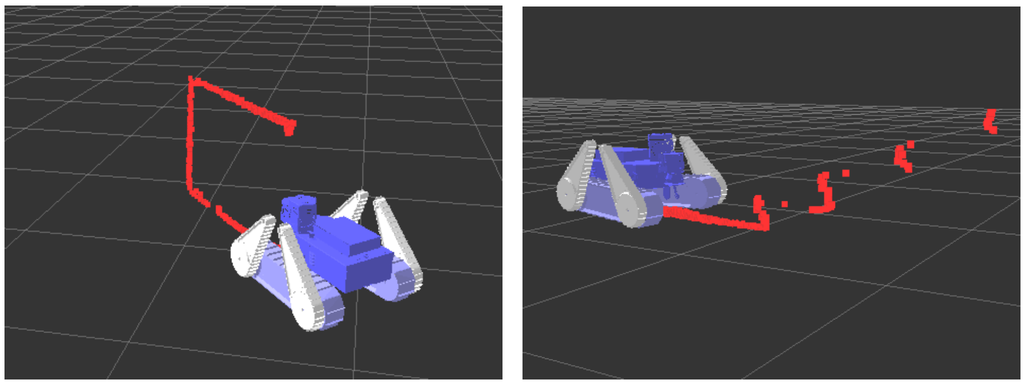

Figure 5.1: The *vertical* scanning plane when the laser scanner is tilted into a fixed position to capture the terrain profile. (a) Robot under a table and facing a wall. (b) Robot facing a staircase.

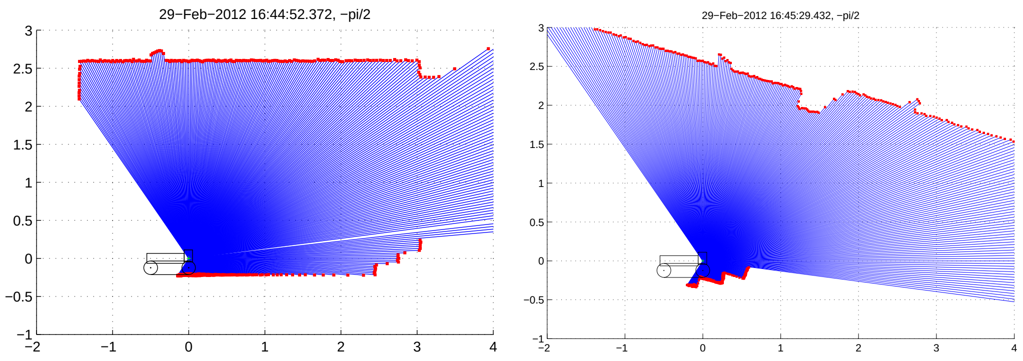

Figure 5.2: Example single vertical laser scans showing (a) the robot approaching a short staircase and then (b) the robot being on the staircase.

otherwise similar construction but they lack the rear flippers entirely, see chapter 3 The NIFTi robot.

The algorithm for adjusting the front flippers is the following:

1. Set the laser permanently into a vertical position. This way it has the most power focused on discovering the terrain profile. We get individual laser scans at a rate of approx. 40-50 Hz.

2. Compute the front flippers position (described in the next sections).

3. Set the flippers physically according to the computed angle.

Now it remains to discover, how to actually compute the front flipper position while having data such as shown in Figure 5.2.

5.2 Computing position of the flippers

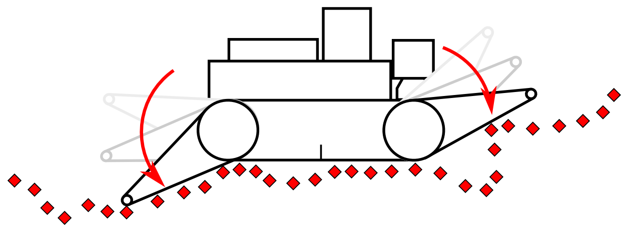
Figure 5.3: For every terrain profile and a known robot pose towards that terrain, there exists a flipper configuration such that the flippers touch the ground (or possibly they are in the lowermost position to mitigate possible impact if they cannot reach down to the ground).

Deciding on the flipper angles has always relied on operator's intuition during obstacle traversal. An optimal solution is not obvious for the complete problem, but is conceivable for a situation, in which the robot does not move, it just stands on a terrain with certain profile as depicted in Figure 5.3.

In this static case, the flippers should touch the terrain to support the robot. Nothing better can be done with the flippers in the static case; there usually is only one angle at which they touch the ground.

It seems to be a good starting point to compute this static position. It is not obvious, whether the static position is usable also in the dynamic case, i.e. when the robot moves and is actually traversing the obstacle. This needs to be discovered.

5.3 Geometrical problem formulation

The algorithm begins with a set of points from a single laser scan. These points have coordinates x_i and y_i. The laser range finder measures *distance* at a certain *angle*, so we obtain (x_i, y_i) by a transformation from polar to Cartesian coordinates. Figure 5.4 only shows the process for one i-th point (the red square) from the single laser scan.

We shift the origin down by distance d, since the laser scanner center is directly above the center of the front wheel. The flipper angle is determined here by the normal vector of the ψ_i line. This line goes through the bottom of the front flipper. Both belts of the main track and that of the flipper are rolled around this wheel. Therefore the ψ_i line forms a tangent to the front wheel. The ψ_i line is located at a distance r from the new origin O and we

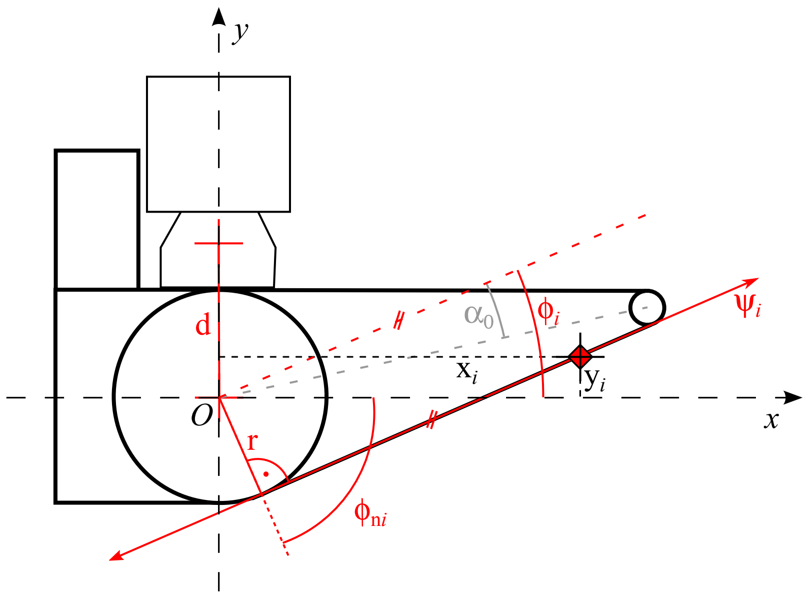

Figure 5.4: Geometry of the front flipper for the i-th point from a single laser scan. The goal is to compute ϕ_i from (x_i, y_i)

.

want to discover its angle ϕ_i or its normal angle $\phi_{\mathrm{n}i}$. The ψ_i line can be described by the equation 5.1 which is a form of 2D line equation used for example in the Hough transform.

$$x_i \cos(\phi_{\mathrm{n}i}) + y_i \sin(\phi_{\mathrm{n}i}) - r = 0 \tag{5.1}$$

Since r (wheel radius) is known, this equation provides a way to compute the normal angle $\phi_{\mathrm{n}i}$ for each and every point from the current laser scan with coordinates (x_i, y_i) lying on differently tilted ψ_i lines.

From the computed set of $\phi_{\mathrm{n}i}$ angles for all points, we choose the highest one ϕ_{nmax}. This is a normal vector angle, we want the flipper angle ϕ_{max} with values from $\frac{\pi}{2}$ (up) to $-\frac{\pi}{2}$ (down). The resulting flipper angle ϕ_{max} is obtained by adding $\frac{\pi}{2}$ to the normal vector angle ϕ_{nmax}.

This is the highest elevation angle and also the lowermost angle to which the flipper can be physically positioned without colliding with the terrain. The flipper then touches the ground in the point with the maximal ϕ_{max} angle; those points can be seen in Figure 5.5.

Note, that the angle reference for a flipper (i.e. $\phi = 0$) is where the lower part of the flipper is in parallel with the ground and with the x axis.

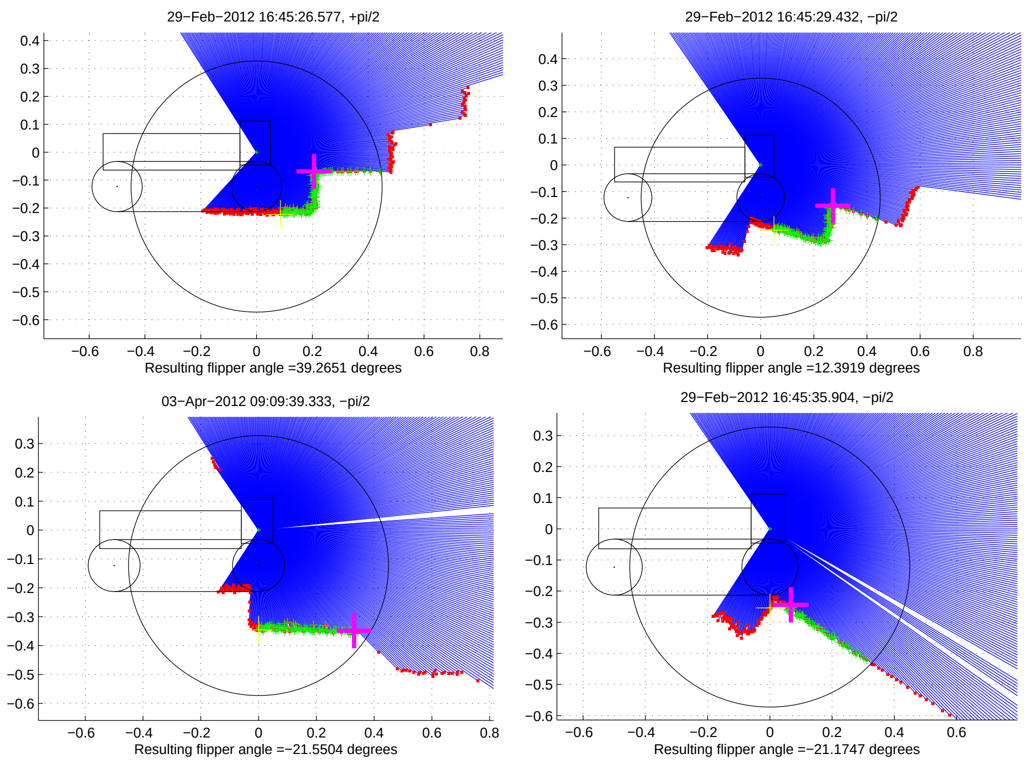

Figure 5.5: Laser scanner data (red), a subset the closed-form solution algorithm uses (green points) and the point with maximal $\phi_{\max}$ angle (pink cross), that determines the flipper position.

5.4 A closed-form solution

Since the equation 5.1 for obtaining the ϕ_{ni} angle is not easily solvable, I used Maple 9.51 to obtain the solution. The following is a slightly modified version of the used Maple script (not showing the i indices):

```
psi := x*cos(phi_n)+y*sin(phi_n)=r;
```

$$\psi : x\cos(\phi_n) + y\sin(\phi_n) = r$$

```
phi_n := solve(psi,phi_n):
phi_n1 := phi_n[1];
```

$$\phi_{\mathrm{n1}} = \operatorname{atan2}\left(\frac{-\dfrac{x\left(rx+\sqrt{x^2y^2+y^4-y^2r^2}\right)}{x^2+y^2}+r}{y}, \frac{rx+\sqrt{x^2y^2+y^4-y^2r^2}}{x^2+y^2} \right)$$

```
phi_n2 := phi_n[2];
```

$$\phi_{\mathrm{n2}} = \operatorname{atan2}\left(\frac{-\dfrac{x\left(rx - \sqrt{x^2y^2 + y^4 - y^2r^2}\right)}{x^2+y^2} + r}{y}, \frac{rx - \sqrt{x^2y^2+y^4-y^2r^2}}{x^2+y^2} \right)$$

```
phi_n1 := simplify(phi_n1);
```

$$\phi_{\mathrm{n1}} = \operatorname{atan2}\left(-\frac{x\sqrt{y^2\left(x^2+y^2-r^2\right)} - ry^2}{\left(x^2+y^2\right)y}, \frac{rx + \sqrt{y^2\left(x^2+y^2-r^2\right)}}{x^2+y^2} \right)$$

```
phi_n2 := simplify(phi_n2);
```

$$\phi_{\mathrm{n2}} = \operatorname{atan2}\left(\frac{x\sqrt{y^2\left(x^2+y^2-r^2\right)} + ry^2}{\left(x^2+y^2\right)y}, \frac{rx - \sqrt{y^2\left(x^2+y^2-r^2\right)}}{x^2+y^2} \right)$$

There are two solutions. In the end, from the ϕ_{n1} and ϕ_{n2} angles, the negative one is chosen preferring ϕ_{n2} if both are negative. The chosen result becomes the $\phi_{\mathrm{n}i}$ angle in our previous notation.

5.5 Integration

So far has been described, how to set the flippers as a reaction to a single vertical laser scan. In order for this algorithm to work practically, we have to run it *continuously.*

Individual laser scans are available with frequency of approximately 50 Hz. For each laser scan we make the computation described earlier, from which we get one maximal $\phi_{\max}$ angle. Since the $\phi_{\max}$ angle varies slightly between individual laser scans even when the robot is stationary due to measurements noise, we employ an averaging of the $\phi_{\max}$ angles. The averaging window can be set for example to 20 laser scans. This means, every 20th scan, we compute the average value of the last 20 $\phi_{\max}$ angles.

Next, the result is *rounded* to tens of degrees (with hysteresis) and then finally sent to the front flippers to physically change position. This way the flipper position is updated with a reasonable frequency, roughly twice a second. Beside averaging, the rounding is supposed to further suppress oscillations of the flippers.

For the rear flippers, we *mirror the same* angle as the one computed for the front flippers as can be seen in Figure 5.6. There is no particular reason, why this should work. It is an idea to be tested.

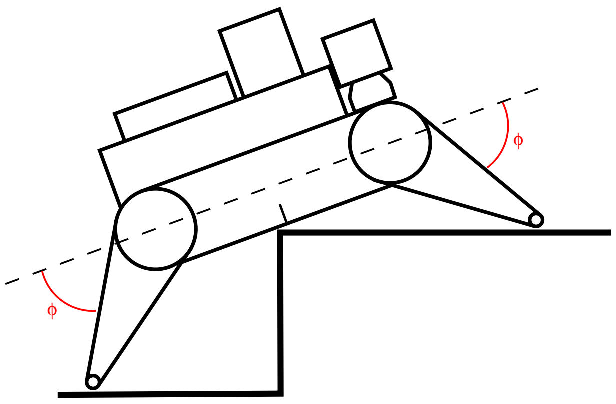

Figure 5.6: Mirror rear flippers angle idea.

The autonomous flippers mode is supposed to be turned on only during traversal of more difficult obstacles, so we need to implement an on/off switch. The whole system function is depicted in Figure 5.7.

Feedback from the flippers is currently not used. The necessary feedback is now provided by a robot operator, who has to adjust the robot's speed, so that the flippers has time to catch up on.

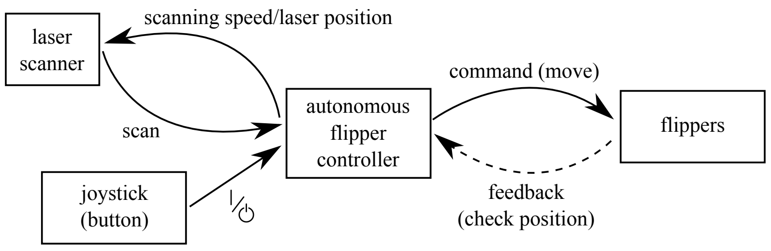
laser scanner
scanning speed/laser position
scan
autonomous flipper controller
command (move)
flippers
feedback (check position)
joystick (button)
I/⏻

Figure 5.7: System diagram.

Chapter 6

Evaluation and experimental results

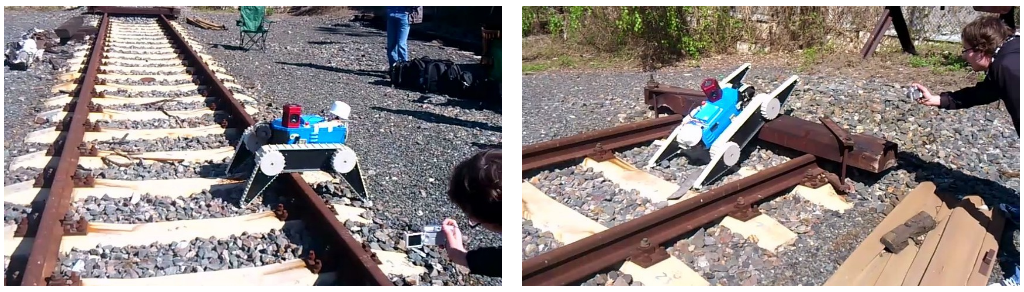
Figure 6.1: The NIFTi robot traversing rails and high sleepers.

The algorithm is capable of traversing staircases and curbs, but it was most intensely tested during a railway yard visit. The operator controlled only the robot's speed and direction and the automatic flipper control algorithm set the flippers so that robot remained safe during traversal of difficult obstacles. We tested it by making both upwards and downwards rides across a fairly high step-shaped railway sleeper pile. See sequences in Figure 6.2 and 6.3. The robot with the automatic flipper control algorithm was operated by different people and no difficulties or failures to traverse an obstacle were reported.

The algorithm does not have any parameters requiring tuning for different obstacles. Even though originally the module was supposed to be only useful for climbing staircases, it ended as a completely general obstacle traversal aid.

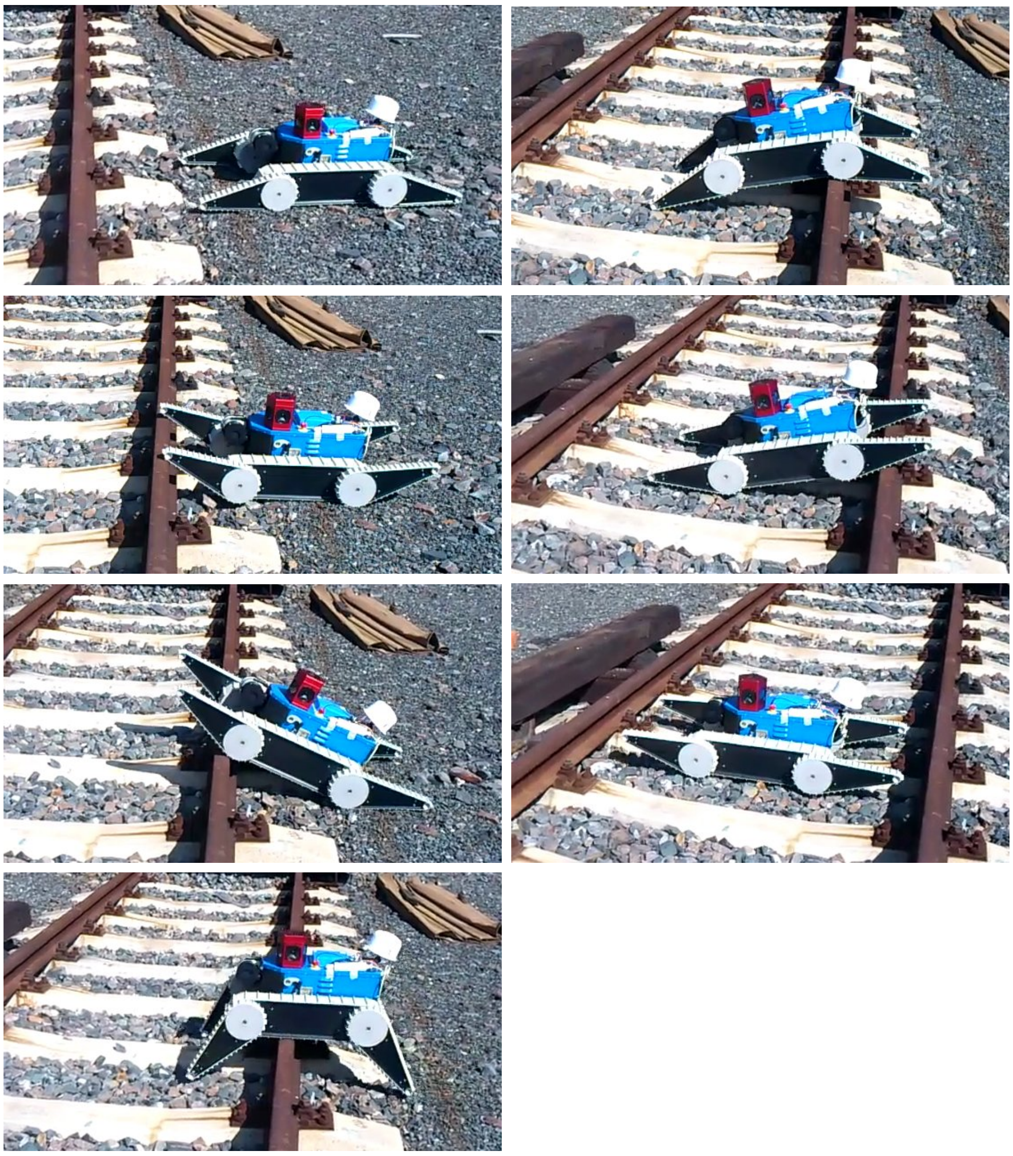

Figure 6.2: The NIFTi robot traversing rails. Flippers are controlled automatically. This situation benefits most from the mirroring method for the rear flippers.

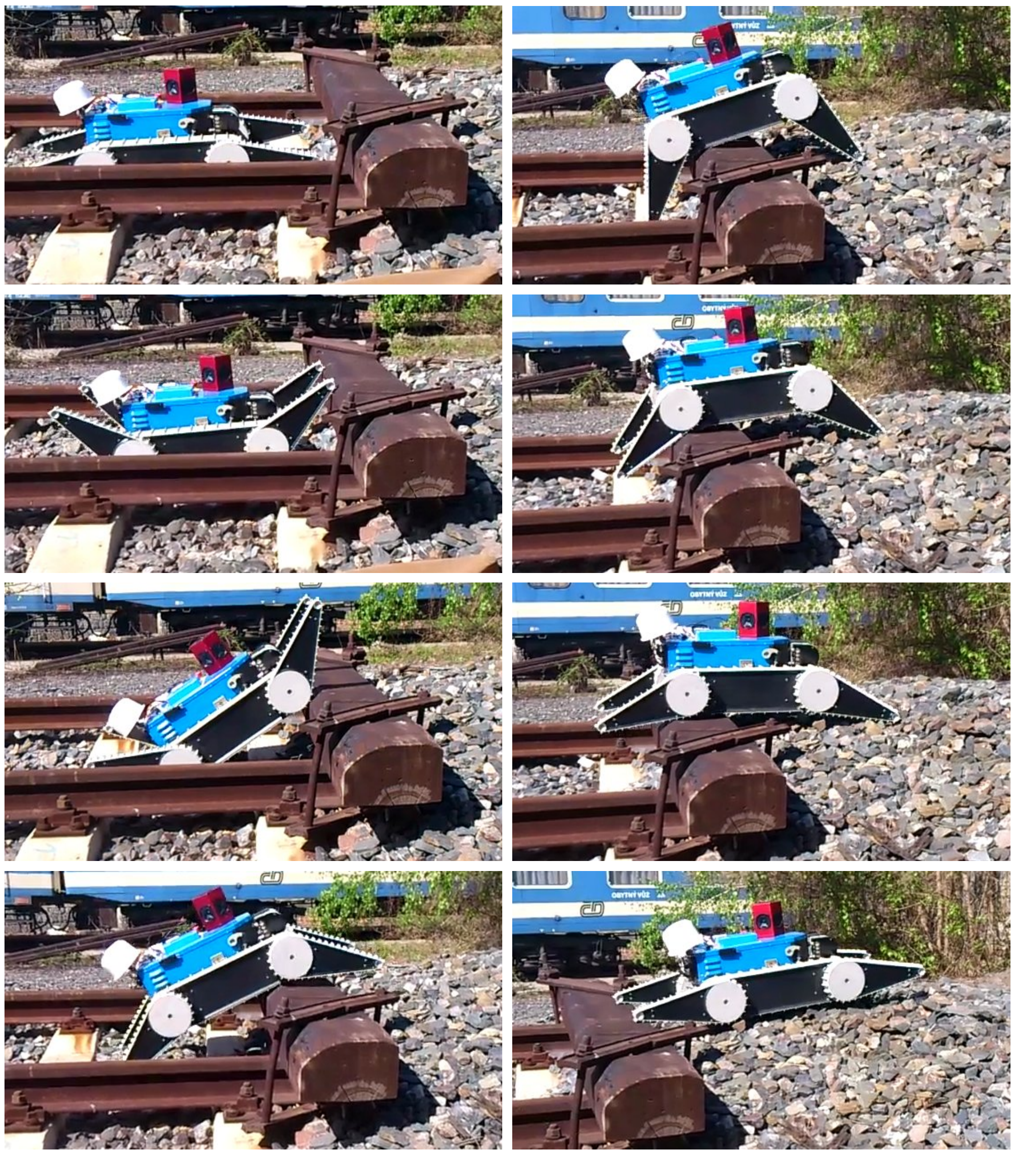

Figure 6.3: The NIFTi robot traversing a high railway sleeper while its flippers are controlled automatically.

6.1 Shortcomings and further discussion

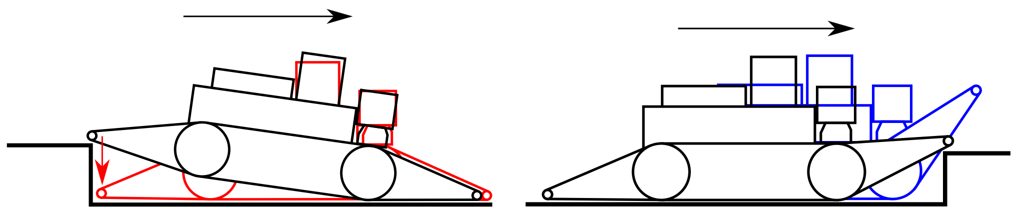

Figure 6.4: (a) Robot back falling when going down from a step. (b) Front flippers unfavorable attack angle.

One question concerns leaving the flippers in the zero position in different situations. For example it was found that the flippers should not be in contact with the ground (where the algorithm places them) when navigating on a flat safe ground. Flippers placed on the ground hamper turning and worsen maneuverability in confined spaces. On the other hand, it is very helpful to leave the flippers fully deployed in zero position during climb in the middle of a staircase. This increases contact of the tracks with the ground and reduces slippages. See Figure 6.5.

When going down a step, the rear flippers are set too low by the algorithm and this causes the robot back to fall a little bit (Figure 6.4 a), if the operator does not anticipate this and does not slow down. The rear flippers should be a bit higher.

When climbing up a step, the attack angle of the front flippers chosen by our algorithm is much higher than necessary (Figure 6.4 b, blue). This is caused by the algorithm trying to avoid collision with the flippers, putting the flippers still higher and higher during the approach. A human operator usually tends to set the attack angle as low as possible (Figure 6.4 b, black).

A general problem is that our robot has only one laser scanner mounted in front of the robot. We assume traversed obstacles have the same profile in the center where the profile is measured by the laser as they have under the tracks and flippers where it actually matters. This is roughly true for most obstacles and a common robot use and a small difference does not matter very much.

In the future the situation may be improved by not using single laser scans, but utilizing a virtual cross-sections of a 3D map of the environment. This will become feasible when sufficiently precise and up-to-date information about the robot position in such map becomes available.

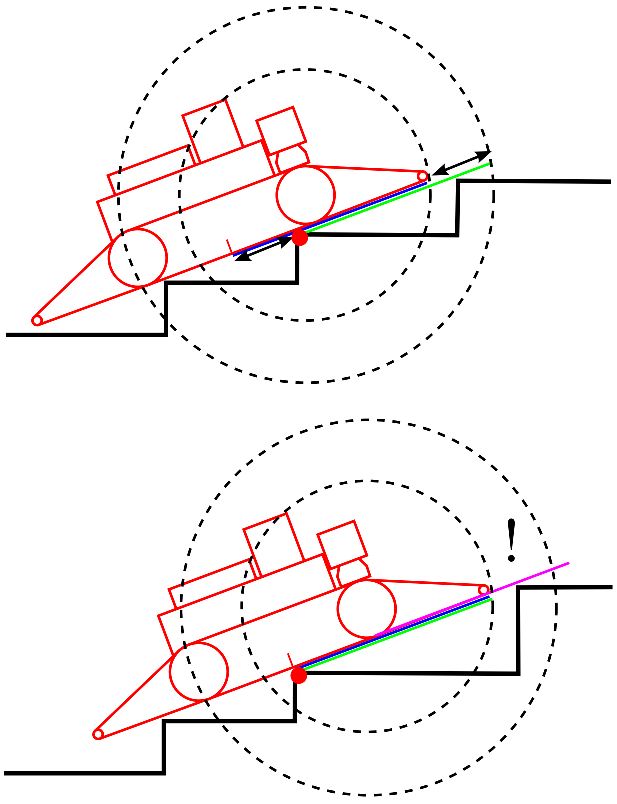

Figure 6.5: It is OK to leave the flippers in zero position in the first case, i.e. to leave the front flipper higher than it would normally be set by the autonomous flippers algorithm, which would try to make it touch the ground below the last step. Such behavior improves performance in the middle of a staircase. However, it is *not* OK to do this in the second example – the robot is going to fall forward before reaching the last step. The problem is, it is hard to discern between these two situation. Knowledge of the position of the red point where the robot is supported is necessary.

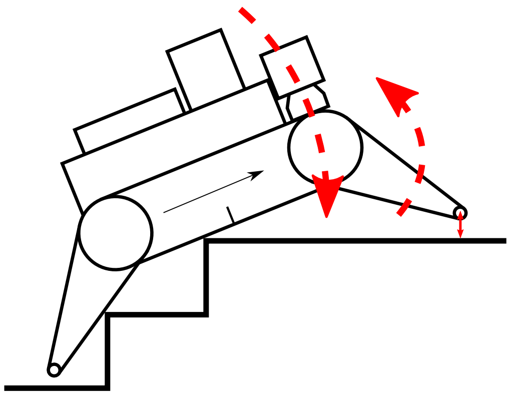

Figure 6.6: The resulting behavior of the robot with the autonomous flipper control is complex and does not depend only on the algorithm itself. For example during landing at the end of a staircase and in similar situations we exploit the algorithm imprecision to gradually lower the robot. It works the following way: the algorithm computes the front flipper angle (negative, under the robot) but this angle is a little bit closer to the zero position than it should be, i.e. the flipper end does not reach the ground. This causes the robot to fall forward a little bit, which in turn makes the robot think that the ground is a little bit higher, which results in putting the flipper even higher. This way the robot progressively lowers its front part until it touches the ground with its main tracks.

Chapter 7

Conclusion

A rescue tracked robot with flipper arms has high ability to get over different kinds of difficult terrain, but it is a slow and demanding task to control its flippers in remote control. A highly skilled operator is usually required.

We present a simple yet effective algorithm based on data from a single vertical 2D laser scanner. The algorithm is constantly setting the front flippers into such position, that they touch the ground. The rear flippers are set to the same angle, *mirroring* the front flippers.

We implemented an autonomous flipper control module for the Robot Operating System (ROS). It is designed to be used intermittently for example during staircase climbing or traversal of any other difficult obstacle, which would otherwise require frequent manual reconfiguration of the flippers.

We used this module to control the NIFTi robot's flippers and examined function of the proposed control law in multiple environments. The module performed well during riding across various obstacles and allowed even an unskilled operator to maneuver quickly on many types of terrain like staircases, steps, curbs or rails by controlling just the main tracks speed and the robot's direction. Since the algorithm tries to hold the front flippers close to the ground, this method also eliminates rough falls of the robot front part and therefore limits damage inflicted to on-board devices. The algorithm does not require any prior knowledge of the obstacle nor any tuning, so the robot is likely to traverse successfully completely unknown obstacles as well.

Aside from relieving load of the robot operator, autonomous flippers control may become relevant as commanding of the robot shifts to a higher level – i.e. the robot movements will not be directly controlled by the teleoperator, but the robot may receive higher-level commands, for example to reach a target located a few meters away. Since there can be obstacles on the robot's path, autonomous flipper control may become very important.

Appendix A

Implementation

Technical description for programmers (innovators and bug-fixers).

The autonomous flipper control algorithm is implemented as a program for the Robot operating system (ROS) running on the NIFTi robot. It is written in C++ and uses the *Eigen* matrix library.

A.1 Notes about the Robot Operating System

Here I will briefly describe some ROS concepts and our autonomous flippers software module.

The `sedlafi2_experiments` name refers to a ROS *package*, which is a filesystem entity. A package is a group of files needed to compile and run one or a small set of related programs in ROS.

Single programs (for example our `fflip_touch` module, which actually controls the robot flippers) are called *nodes* in ROS. One package can contain several nodes and files required to compile and run these nodes. These are typically its source code, launchfiles and message definitions.

The programs in ROS are called nodes because when they are run, they form a *Computation Graph*, a peer-to-peer network which organizes their mutual communication. Nodes communicate using data *messages*. A message is a data structure containing typed fields. The following is an excerpt from [`http://www.ros.org/wiki/ROS/Concepts`] with a straightforward description:

"Messages are routed via a transport system with publish/-

> subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence."

The whole system is coordinated by the *ROS Master*. There are also other interesting ROS concepts, like *services* and *bags*, which are not covered here, because these are not directly used with the autonomous flippers module. More information to understand ROS can be found at:
`http://www.ros.org/wiki/ROS/Concepts`
`http://www.ros.org/wiki/ROS/Tutorials/UnderstandingNodes`

A.2 Interface

This section describes how the autonomous flipper program communicates with its surroundings.

A.2.1 Subscriptions (input)

The `fflip_touch` node subscribes to the following ROS topics:

- `/scan` for `sensor_msgs/LaserScan` messages. Each message carries one 2D scan made by the laser scanner. These messages arrive approximately at a rate of 50 Hz. Content (definition) of the message can be found here: `http://www.ros.org/doc/api/sensor_msgs/html/msg/LaserScan.html`.

- `/joy` for `joy::Joy` joystick messages containing information about button and joystick axes changes.

- `/robot_status` for `nifti_robot_driver_msgs::RobotStatus` message. The node uses only one its subfield, `RobotStatus::scanning_speed` to discover the current angular velocity of the laser scanner rotation.

- `/flippers_state` for `nifti_robot_driver_msgs::FlippersState` message. This message is used marginally and tells the node the current position of the four flippers.

A.2.2 Publications (output)

The node publishes messages to the following topics:

- `/flipper_cmd`, message type `nifti_robot_driver_msgs::FlipperCommand`
- `/flippers_cmd`, message type `nifti_robot_driver_msgs::FlippersState`
- `/scanning_speed_cmd`, message type `std_msgs::Float64`
- `/laser_center`, message type `std_msgs::Bool`

A.3 Constants (thresholds)

The program contains several categories of constants. The first set of constants is tied to our NIFTi robot. Since the algorithm performs computations based on the robot dimensions, these dimensions are hard-coded in the source code. Their values has been derived from the robot technical drawing (Figure 3.1). Lengths are in meters and angles in radians.

- **double** downFromLaser = −0.123;

  The y-axis shift from the center of the laser scanner (the coordinate frame origin) to the center of the front wheel. This point is a center of rotation of the front flippers.

- **double** wheelRadius = 0.090;

  The radius of the front robot wheels (holding the main tracks and flippers tracks).

- **double** flipperLength = 0.45;

  This constant's meaning is to specify the length of the flipper (from the center of its rotation – center of the front wheel – to its end). The value however does not state the real length of the flipper. The real value is 0.356 m and a higher value is used to make the algorithm react more usefully. Now the algorithm thinks, that the robot has longer front flippers, it raises it earlier and the flipper does not collide with the step.

- **double** alpha0 = 0.1937;

An angle offset. It comes from the flipper shape and is visible on the robot technical drawing (Figure 3.1).

Another category of constants is affecting the node function. There are two constants affecting the program before it outputs the flipper angle:

- **unsigned int** anglesAveragingWindowSize = 20;

  This value specifies the size of an averaging window. It specifies the number of scan messages the node has to receive and compute corresponding flipper angles to publish a command for changing the flipper angle. It publishes only if the rounded average value is different from the last one.

- **unsigned int** roundPublishedAnglesTo = 10*PI/180;

  A magnitude of the angle change step for the flippers, it is used for rounding. This value is an flipper angle and is in radians even though its value is set default to be 10 degrees.

Finally, there are constants tied to the laser scanner itself and corrections of its output (described later in section A.4.3 The core – flipper angle computation).

A.4 Program structure

The autonomous flipper control node resides in several .cpp files. In the `main` function, the code only initializes ROS and constructs the node class.

A.4.1 Constructor

In the constructor, class fields are initialized. Some values are initialized to a values adopted from a launch file (if the node is run using a launch file). The node subscribes to all topics described in the Interface section and advertises, that it is going to publish on topics. Loading parameters from a launch file is facilitated by the `getParam` method. This method comes from the nifti_laser_assembler source.

A.4.2 Callbacks

The rest of the program comprises of callbacks. These are methods, which get executed after a ROS message is received by the node on a specific topic. Each subscription to a specific topic has its own callback function (class method).

Some callbacks are only used to receive and store the last message published on some topics. These include `/robot_status` topic (the `robotStatusCallback` method), which contains specific messages connected to our robot (message type `nifti_robot_driver_msgs::RobotStatus`). We are only interested in the `scanning_speed` subfield, which contains the current laser scanner angular speed.

The `flippersStateCallback` method stores the last message received on the `/flippers_state` topic, which gives our node the true position of the flippers. This value is currently used only marginally during the autonomous flippers mode shutdown to stop the ongoing flippers relocation immediately. This is intended to be a safety element – if the robot is doing something undesirable or dangerous with the flippers, the operator can promptly disengage the autonomous flippers mode and the flippers must stop their movement *without completing* the ongoing (the last) relocation command by the control algorithm. Our node is not actually constantly checking, whether the flippers are at the position, to which the control algorithm sends them. It is, however, quite easy to implement such functionality.

The `joystickCallback` receives a message of the type `joy::Joy` and implements a reaction to a button press. Since the joystick (node) sends out messages every time a joystick state changes – which means, someone has pushed or released a button, but also that a joystick axis has changed slightly – and buttons state is send out as an array of boolean values (0/1), it is not clear at first, when a button has been *pressed*. In other words, from the many messages we receive on the `/joy` topic, we have to detect, which message truly represents the event, that the button dedicated as an on/off switch for the autonomous flippers mode has been pressed.

When detected, that the on/off button has just been pressed, we execute the `buttonAction` method. This method implements the on/off functionality. It toggles the state of the autonomous flippers mode. The `buttonAction` method controls the behavior of the node in the following way:

If the autonomous flippers mode is off (i.e. we are turning it on), it:

1. Stores the most recent value of the rotating laser scanner angular speed (`scanning_speed`).

2. Turns the laser to one of the two vertical positions using the

`turnLaserVertical` method.

This is not true if the module is in a *manual laser angle mode* where responsibility of turning the laser into vertical position is solely on the robot operator. In such case, the node does not even check, whether the laser is in the right position and the flippers control algorithm operates anyway. This mode was used when the laser scanner rotation mechanism broke down. It partly could not rotate properly, but also reported a wrong angle to the ROS system.

3. Toggles a boolean variable:

   ```
   staircaseModeState = true;
   ```

   This boolean variable controls the autonomous flipper control algorithm, which is started off from the `scanCallback` method discussed later.

If the autonomous flippers mode is on and we are turning it off, the program:

1.
   ```
   staircaseModeState = false;
   ```

   Toggles the boolean state variable the so that the flipper control algorithm knows immediately (upon receiving the next laser data) not to operate.

2. Cancels the ongoing (if any) flipper movement by publishing a new flipper command containing the last *physical* position flippers reached and reported it back to this node (through the `/flippers_state` topic).

3. Attempts to return the rotating laser scanner into a motion state prior to turning the autonomous mode on. It either restores it by publishing the saved value from before turning the autonomous mode on to the `/scanning_speed_cmd` topic, or sends the laser into zero position (and zero speed) if the data are not accessible or the laser scanner was not moving when the autonomous mode was turned on. It means, that if the laser was moving (rotating), the node restores its rotational speed. If the laser scanner was stopped at certain angle, this angle is lost. The node would have to set the laser scanner back into this angle and that is a difficult task. Thus, the laser is returned to (and stopped at) the zero position, i.e. horizontal scanning plane.

The `turnLaserVertical` method implements a relatively complicated task of turning the laser scanner into one of the vertical positions, in which it can perform our task, i.e. scan the terrain profile. The task is complicated, because the current robot implementation does not have any low-level command to send the rotating laser scanner into specific position (angle). So the command has to be implemented using a pseudo-closed-loop software controller.

The main function of the autonomous flipper control algorithm is started off from the `scanCallback`. The algorithm works upon receiving a `sensor_msgs::LaserScan` message on the topic `/scan`. Everything that follows (which is the core of the autonomous flipper algorithm) is executed only **if** (staircaseModeState == **true**).

First, the laser scanner angle is determined using the `get_laser_angle` method by querying the ROS TF subsystem for the laser angle at a specific time. The laser scanner angle value is the angle the laser scanner had at the time of the laser data capture - this time is discovered using message timestamp.

Then the the flipper computer class (`FlippersAngle`) is initialized with the current laser data and scanner angle and is asked to compute the front flippers angle:

```
FlippersAngle flippersAngleComputer(ranges, sensorAngle);
double resultingFFAngle =
    flippersAngleComputer.getFrontFlippersAngleClosedForm();
```

The resulting flipper angle is stored in a buffer. The size of this buffer is specified by the value of the `anglesAveragingWindowSize` constant described in the Constants (thresholds) section. The algorithm does nothing more in the `scanCallback` method, if the buffer is not fully filled. When it contains the target number of values, the algorithm proceeds:

1. It averages all the computed angles in the buffer.

2. The resulting average is rounded and compared to the current value (i.e. the last published and most likely also physically achieved by the flippers). A hysteresis is implemented, so that the new value needs to be a little bit higher than the half of the difference between the angle steps after rounding.

3. It publishes the resulting angle, i.e. sends it to the flippers to physically to execute the movement only if the new computed rounded value is different from the last one. This is done using the `publishFlippersAngle` method.

   Not publishing the value if it is the same as before is intended to relieve the load of the ROS messaging system, but it has a downside. If one of the messages is lost, the flippers may remain in the old position, because the algorithm does not send out the message again and it also does not check, whether the flippers are in the target position.

Currently, there is a bug, which could be solved by publishing the angle always, even if it has not changed. Currently, the flippers do not move immediately after the autonomous mode is turned on, they move only after there is an obstacle in front of the robot (i.e. which causes the computed angle to be dramatically different and therefore a new value is published).

4. Erase the buffer and start accumulating a new batch of angles.

A value of the `anglesAveragingWindowSize` constant is typically set to 20 (twenty last angle values averaged), which at an approximately 45 Hz rate of the `/scan` messages gives 2.25 angle updates per second. An in-between delay is 0.44 s.

The `publishFlippersAngle` just does the ROS message publishing of the computed flippers angle. The sign of the angle is inverted because the robot actually uses an opposite angle direction. There is also a possibility of an offset correction, even though currently the offset correction is zero. It also publishes the *same* angles for the rear flippers with changed sign once more.

A.4.3 The core – flipper angle computation

This code is currently in the `FlippersAngle.cpp` file. This file also contains the older discretized version of the flipper angle computation algorithm, which was replaced by a shorter, simpler version using a closed-form solution.

The `FlippersAngle` class initializes with the data from a single laser scan and the scanner angle, with which this scan was captured. For the scanner angle only two values are possible: $\pm\frac{\pi}{2}$. Even though they both signify the vertical position of the laser and the autonomous flipper mode cannot be used in any other laser scanner angle (this actually could be changed in the future), the algorithm needs to know the angle at which the data was captured. It has to first somehow determine where is up and down and secondly, it uses minor corrections, which differs slightly for to the two positions of the lase scanner.

After initialization, the `FlippersAngle` class proceeds by running the `prepareRanges` method from its constructor, where it:

1. Reverses the laser ranges data, if the laser scanner angle is $-\frac{\pi}{2}$.

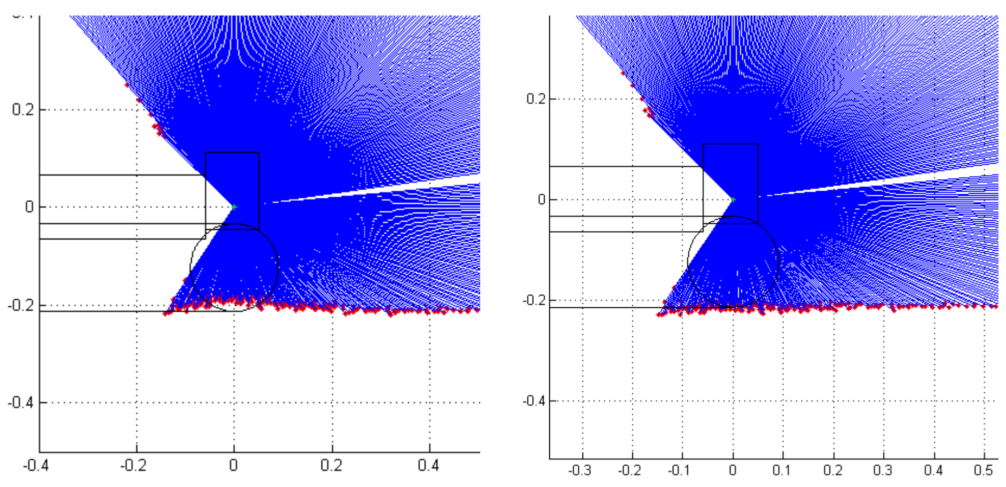

Figure A.1: (a) Uncorrected (as it comes from the laser scanner). (b) Corrected. Correction of the near range measurements. The correction uses the formula ranges$[i]$ = ranges$[i]+0.17e^{-10\cdot \text{ranges}[i]}$ for all points closer than 0.5 m, which has been simply made up to make the points in the left plot appear as in the right plot.

2. Corrects the range measurements, which are nearer than 0.5 m:

   ```
   ranges[i] += corrCoef1 * exp(-10 * ranges[i])
   ```

   where `corrCoef1` is set to 0.17 by hand, see Figure A.1 and A.2.

3. Since the `sensor_msgs::LaserScan` contains the range measurements but it actually does not contain the angles, at which these range measurements have been made, the angles needs to be computed. This task is again slightly dependent on the value of the laser scanner angle ($\pm\frac{\pi}{2}$). The angles are in either case a set of 540 values, which divide the total scanning angle of the laser scanner (270°) into half-angle pieces. If the laser scanner angle is $-\frac{\pi}{2}$, the angles are values from -135.0 to 134.5 (both inclusive) with a 0.5 degree increment. If the laser scanner angle is $+\frac{\pi}{2}$, the angles start at value of -134.5 and goes to 135.0 (both inclusive) with a 0.5 degree increment. When the angles are computed, we actually get coordinates of all the laser points in polar coordinates.

   Note: it is possible, that this step of the algorithm could cancel out with the (now probably unjustified) order reversal of the values in the first step of the preparation stage of the algorithm. Then the angles computation could be independent of the laser scanner angle.

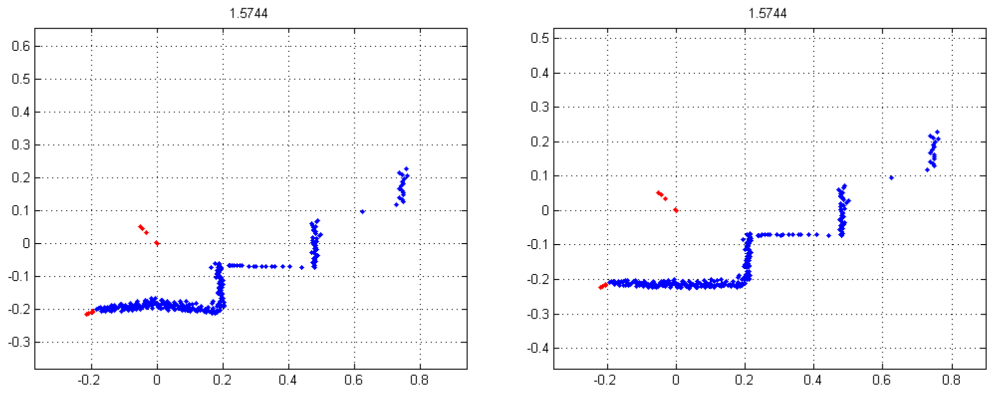

Figure A.2: (a) Uncorrected (as it comes from the laser scanner). (b) Corrected. Correction of the near range measurements as it affects a range scan of a staircase. The red points are completely wrong measurements caused probably by reflection of the laser from the robot body. These points are occur on the ends of the circular sector of the laser scan and we attempt to discard them by cutting off a number of points at the ends (these points are not useful anyway, since they are not in the front direction).

4. Go through all of the 540 values and complete the individual points coordinates set with the Cartesian coordinates (x, y) of the points.

$$(\rho, \theta, x, y) = (\rho, \theta, \rho \cos \theta, \rho \sin \theta)$$

   Each point is in the algorithm then represented by a vector with four components, first two ρ, θ represent the polar coordinates (ρ is the measured range and θ is the computed angle at which the range measurement was acquired).

5. The algorithm filters points from the sides of the total angular range. The concrete cut-off boundaries were discovered experimentally, but their value can change and they should be checked once in a while. The goal here is to remove unwanted points, which for example belong to the robot body (or the antenna typically). These points are at the sides of the angular laser scanner range and they can interfere with the subsequent algorithms working with the points. We have to assure, that the subsequent algorithms will not be confused by points belonging to the robot body confusing it with an obstacle.

   Note: the current algorithm probably does not depend on this filtering step, since it takes only points, which have positive x coordinate.

That cuts off a substantial side portions of the angular section scanned by the laser and probably all the problematic points caused by laser measurements on the robot body.

Moreover, the algorithm discards all points, whose ρ coordinate (i.e. distance) is smaller than 0.1 m. These can be zero values (not measured, error, or did not get a reflection) or very near values caused by reflecting some part of the robot body. The laser measurement is invalid on a such near range anyway.

The `FlippersAngle` class contains a method `getFrontFlippersAngleClosedForm()`. This one actually performs the angle computation. This method contains several constants connected to the robot dimensions and kinematic structure described in the section A.3 Constants (thresholds).

```
double downFromLaser = -0.123;
double wheelRadius = 0.090;
double flipperLength = 0.45; // 0.356 correct, 0.45 working
  // 11.1 deg according to the technical drawing:
double alpha0 = 0.1937;
double p = wheelRadius * cos(alpha0);
```

Then the front flippers angle algorithm proceeds:

1. It first transforms the points origin into the center of the robot's front wheel. This is just a translation in the vertical (y) direction.

2. It recomputes all points distances to the new origin..

3. Only points inside a circle around the origin are chosen for further processing. Specifically, only points fulfilling this condition are chosen:

   ```
   if (x > 0 && dist < flipperLength
        && dist > wheelRadius * sqrt(2))
   ```

   It roughly means: we want only points in front of the front wheels (since there are the flippers), no further away than the flipper length and a little bit further away from the edge of the front wheel, since these points would produce weird results.

   Currently, it is not very well handled if there are no points, which fulfill this condition. The algorithm should do nothing in such case.

4. For all chosen points, $\phi_{01,02}$ angles are computed using the closed-form solution (for derivation see 5.4 A closed-form solution):

$$y_1 = -\frac{x\sqrt{y^2(x^2+y^2-p^2)}-py^2}{(x^2+y^2)y} \qquad x_1 = \frac{px+\sqrt{y^2(x^2+y^2-p^2)}}{x^2+y^2} \qquad \phi_{01} = \operatorname{atan2}(y_1, x_1)$$

$$y_2 = \frac{x\sqrt{y^2(x^2+y^2-p^2)}+py^2}{(x^2+y^2)y} \qquad x_2 = \frac{px-\sqrt{y^2(x^2+y^2-p^2)}}{x^2+y^2} \qquad \phi_{02} = \operatorname{atan2}(y_2, x_2)$$

5. Then only one of these angles is chosen for each point. The second angle is preferred, if it is negative, if it is not, then the first one is used, if it is negative. If neither of the angles is negative, this is an unexpected situation, the code issues a warning.

6. The algorithm finds the maximum angle.

7. It adds $\frac{\pi}{2}$ to the result, since the algorithm returns a normal vector angle and we are interested in a tangent angle (counter-clockwise-turned from the normal angle). This is the angle of the flipper which is finally returned.

A.5 Used source code

`nifti_laser_assembler.cpp`

Bibliography

[1] The NIFTi project website, may 2012. URL `http://www.nifti.eu/`.

[2] J.D. Martens and W.S. Newman. Stabilization of a mobile robot climbing stairs. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 2501 –2507 vol.3, may 1994. doi: 10.1109/ROBOT.1994.351135. URL `http://dx.doi.org/10.1109/ROBOT.1994.351135`.

[3] Victor Sanchez and Avideh Zakhor. Planar 3D reconstruction from large-scale unstructured point data for 3D modeling and visualization. sept. 2012. URL `http://www-video.eecs.berkeley.edu/papers/victors/Sanchez_Zakhor_ICIP2012.pdf`.

[4] S. Osswald, J.-S. Gutmann, A. Hornung, and M. Bennewitz. From 3d point clouds to climbing stairs: A comparison of plane segmentation approaches for humanoids. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 93 –98, oct. 2011. doi: 10.1109/Humanoids.2011.6100836. URL `http://dx.doi.org/10.1109/Humanoids.2011.6100836`.

[5] Qun Zhang, Shuzhi Sam Ge, and Pey Yuen Tao. Autonomous stair climbing for mobile tracked robot. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 92–98, nov. 2011. doi: 10.1109/SSRR.2011.6106757. URL `http://dx.doi.org/10.1109/SSRR.2011.6106757`.

[6] E. Mihankhah, A. Kalantari, E. Aboosaeedan, H.D. Taghirad, S. Ali, and A. Moosavian. Autonomous staircase detection and stair climbing for a tracked mobile robot using fuzzy controller. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 1980 –1985, feb. 2009. doi: 10.1109/ROBIO.2009.4913304. URL `http://dx.doi.org/10.1109/ROBIO.2009.4913304`.

[7] A. Kalantari, E. Mihankhah, and S.A.A. Moosavian. Safe autonomous stair climbing for a tracked mobile robot using a kinematics based controller. In *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, pages 1891 –1896, july 2009. doi: 10.1109/AIM.2009.5229765. URL `http://dx.doi.org/10.1109/AIM.2009.5229765`.

[8] S. Steplight, G. Egnal, S.-H. Jung, D.B. Walker, C.J. Taylor, and J.P. Ostrowski. A mode-based sensor fusion approach to robotic stair-climbing. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 2, pages 1113 –1118 vol.2, 2000. doi: 10.1109/IROS.2000.893168. URL `http://dx.doi.org/10.1109/IROS.2000.893168`.

[9] Yalin Xiong and L. Matthies. Vision-guided autonomous stair climbing. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 1842 –1847 vol.2, 2000. doi: 10.1109/ROBOT.2000.844863. URL `http://dx.doi.org/10.1109/ROBOT.2000.844863`.

[10] D.M. Helmick, S.I. Roumeliotis, M.C. McHenry, and L. Matthies. Multi-sensor, high speed autonomous stair climbing. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 733 – 742 vol.1, 2002. doi: 10.1109/IRDS.2002.1041478. URL `http://dx.doi.org/10.1109/IRDS.2002.1041478`.

[11] Anastasios I. Mourikis, Nikolas Trawny, Stergios I. Roumeliotis, Daniel M. Helmick, and Larry Matthies. Autonomous stair climbing for tracked vehicles. *Int. J. Rob. Res.*, 26(7):737–758, July 2007. ISSN 0278-3649. doi: 10.1177/0278364907080423. URL `http://www-users.cs.umn.edu/~trawny/Publications/IJCV07_stairclimbing.pdf`.

[12] Yang Cong, Xiaomao Li, Ji Liu, and Yandong Tang. A stairway detection algorithm based on vision for ugv stair climbing. In *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, pages 1806 –1811, april 2008. doi: 10.1109/ICNSC.2008.4525517. URL `http://dx.doi.org/10.1109/ICNSC.2008.4525517`.

[13] K. Ohno, S. Morimura, S. Tadokoro, E. Koyanagi, and T. Yoshida. Semi-autonomous control system of rescue crawler robot having flippers for getting over unknown-steps. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3012 –3018,

29 2007-nov. 2 2007. doi: 10.1109/IROS.2007.4399271. URL `http://dx.doi.org/10.1109/IROS.2007.4399271`.

[14] K. Nagatani, A. Yamasaki, K. Yoshida, T. Yoshida, and E. Koyanagi. Semi-autonomous traversal on uneven terrain for a tracked vehicle using autonomous control of active flippers. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2667 –2672, sept. 2008. doi: 10.1109/IROS.2008.4650643. URL `http://dx.doi.org/10.1109/IROS.2008.4650643`.

[15] Y. Okada, K. Nagatani, and K. Yoshida. Semi-autonomous operation of tracked vehicles on rough terrain using autonomous control of active flippers. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 2815 –2820, oct. 2009. doi: 10.1109/IROS.2009.5354549. URL `http://dx.doi.org/10.1109/IROS.2009.5354549`.

[16] The Robot Operating System (ROS) website, may 2012. URL `http://www.ros.org/`.