CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Cybernetics

Safe Obstacle Traversal with Incomplete Data

Bezpečné přejíždění překážek s neúplnými daty

DIPLOMA THESIS

Author: Bc. Jakub Mareš
Supervisor: Ing. Karel Zimmermann, Ph.D.

Prague, May 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Jakub M a r e š

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Safe Obstacle Traversal with Incomplete Data

Guidelines:

The robot rescuer http://cmp.felk.cvut.cz/demos/robotics/mobile-robots is developed in the Center for Machine Perception (CMP) at the Department of Cybernetics within the European project TRADR developed. One of the most important functionalities of the robot rescuer is the autonomous control of auxiliary articulated tracks - the so-called flipper. Currently used method for autonomous flipper control requires the full knowledge of all the sensory data (laser scan, pitch, roll, torque in engines). This condition, however, is in real life situations often violated (e.g. the terrain in front of the robot is not visible). Design and practically verify the extension of the existing algorithms for the safe functioning with incomplete data.

1. Read, re-learn and re-implement (if necessary) the existing autonomous flipper control method [1] based on the reinforcement learning [3] and decision trees [2].
2. Propose a metric determining safety of available actions on partially visible terrain.
3. Extend the existing algorithm [1] by active perception of not visible terrain. Exploit the existing algorithm for 3D terrain reconstruction by using robotic arm proposed by V.Šalanský.
4. Compare your active perception strategy with the 3D reconstruction strategy and show how these strategies differ.

Bibliography/Sources:

[1] K. Zimmermann, P. Zuzánek, M. Reinstein, and V. Hlaváč: „Adaptive traversability of unknown complex terrain with obstacles for mobile robots," in IEEE International Conference on Robotics and Automation, pp. 5177–5182, 2014.
[2] Trevor Hastie, Robert Tibshirani and Jerome Friedman: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer, 2009
[3] Richard S. Sutton and Andrew G. Barto: Reinforcement Learning: An Introduction, MIT Press, 2012.

Diploma Thesis Supervisor: Ing. Karel Zimmermann, Ph.D.

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 23, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Jakub M a r e š

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Bezpečné přejíždění překážek s neúplnými daty

Pokyny pro vypracování:

V Centru strojového vnímání (CMP) na katedře kybernetiky je v rámci Evropského projektu TRADR vyvíjen robot záchranář http://cmp.felk.cvut.cz/demos/robotics/mobile-robots. Jednou z důležitých funkcionalit je autonomní řízení pomocných artikulovaných pásů – tzv. flipperů. Podmínkou dobrého fungování aktuálně navrženého algoritmu pro autonomní řízení flipperů je úplná znalost všech senzorických dat (laserových scanů, náklonů robota, proudů ve flipperech). Tato podmínka je však v reálných situacích často porušena (např. terén před robotem není viditelný). Navrhněte a prakticky ověřte rozšíření stávajícího algoritmu pro bezpečné fungování i při neúplných datech.

1. Nastudujte stávající algoritmus autonomního řízení flipperů [1] založený na posilovaném učení [3] a rozhodovacích stromech [2].
2. Navrhněte algoritmus rozhodující o bezpečnosti/nebezpečnosti přejetí jen částečně viditelné překážky.
3. Rozšiřte stávající algoritmus [1] o možnost aktivního průzkumu terénu pomocí robotické ruky. Využijte stávající algoritmus pro 3D rekonstrukci terénu před robotem naimplementovaný V. Šalanským a ukažte, jak se Vaše strategie kontaktního průzkumu terénu pro autonomní řízení flipperů liší od strategie pro 3D rekonstrukci terénu.
4. Navržený algoritmus experimentálně ověřte v hasičském tréninkovém centru pro Urban Search & Rescue mise (přístup zajistí školitel během review meetingu projektu TRADR).

Seznam odborné literatury:

[1] K. Zimmermann, P. Zuzánek, M. Reinstein, and V. Hlaváč: „Adaptive traversability of unknown complex terrain with obstacles for mobile robots," in IEEE International Conference on Robotics and Automation, pp. 5177–5182, 2014.
[2] Trevor Hastie, Robert Tibshirani and Jerome Friedman: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer, 2009
[3] Richard S. Sutton and Andrew G. Barto: Reinforcement Learning: An Introduction, MIT Press,2012.

Vedoucí diplomové práce: Ing. Karel Zimmermann, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 23. 1. 2015

Abstract

This thesis deals with terrain traversability for an unmanned ground vehicle (UGV) based on Niftibot platform. This mobile robot, which is dedicated for Urban Search and Rescue (USAR) missions, is equipped with auxiliary articulated tracks, so-called *flippers*. Flippers enhance robot´s ability to traverse complicated terrain, however they bring more degrees of freedom to control. Semi-autonomous control system which selects optimal flippers´ configuration with respect to traversed terrain is being developed at FEE, CTU.

A system based on reinforcement learning and decision trees had been previously implemented. This system, however, required complete data from sensors. As model of environment was built using solely data from laser scanner, this condition was violated in some scenarios, e.g. in case of reflective surfaces. Therefore a partial reimplementation and an extension of the former system is introduced in this work. A new mode which utilizes JACO robotic arm for tactile exploration of terrain has been incorporated to the system. This helps to explore terrain invisible to laser scanner. The experiments with aluminium foil were performed to demonstrate that the arm helps the robot to complete information in robot´s map and furtherly use it to safely traverse terrain.

Keywords

Adaptive traversability, mobile robotics, tracked vehicle, flippers, robotic arm, tactile exploration, digital elevation map, Robot Operating System.

Abstrakt

Tato práce se zabývá problematikou bezpečného přejíždění terénu pro mobilního pozemního robota, který je vyvíjen pro nasazení při záchranných operacích. Za tímto účelem je robot vybaven artikulovanými postranními pásy, *flippery*, které zlepšují schopnost robota pohybovat se terénem, ale přinášejí s sebou také více stupňů volnosti, které je potřeba řídit. Proto je na Fel, Čvut, vyvíjen semi-autonomní řídící systém, který volí optimální konfiguraci postranních pásů s ohledem na právě přejížděný terén.

Již dříve byl vytvořen řídící systém založený na posilovaném učení a rozhodovacích stromech. Avšak tento systém vyžadoval úplné informace ze senzorů. Jelikož mapa prostředí byla vytvářena výhradně z dat poskytovaných laserovým dálkoměrem, byla tato podmínka mnohdy nesplněna, např. v případě lesklých povrchů. Proto byla v rámci této práce provedena částečná reimplementace a rozšíření původního systému. Nový mód, který využívá robotickou paži JACO pro dotykový průzkum terénu, byl přidán do systému. Paže pomáhá prozkoumávat terén neviditelný pro laserový scanner. Experimenty s hliníkovou folií byly provedeny, aby demonstrovaly, že paže může skutečně pomoci robotu doplňovat nezbytné informace do mapy a ty následně využívat pro bezpečné přejíždění terénu.

Klíčová slova

Adaptivní přejíždění terénu, mobilní robotika, pásové vozidlo, flippery, robotická paže, dotykový průzkum, digitální výšková mapa, Robot Operating System

Acknowledgement

I would like to thank my family for their material and moral support for the whole time during my studies.

Prohlášení autora práce

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 ....

Podpis autora práce

Contents

List of annexes

1. AT_1.mp4
2. AT_with_JACO_1.mp4
3. AT_with_JACO_2.mp4
4. model_1.m
5. model_2.m
6. model_3.m
7. model_4.m
8. model_5.m

List of abbreviations

AI – artificial intelligence

AT – adaptive traversability

CIIRC – Czech Institute of Informatics, Robotics and Cybernetics

CTU – Czech Technical University in Prague

DEM – Digital elevation map

DoF – degrees of freedom

IMU – inertial measurement unit

FEE – Faculty of Electrical Engineering

MEMS – micro-electro-mechanical system

RL – reinforcement learning

ROS – Robot Operating System

TRADR – Teaming for Robot-Assisted Disaster Response

UAV – unmanned aerial vehicle

UGV – unmanned ground vehicle

USAR – urban search and rescue

1. Introduction

Tracked unmanned ground vehicle (UGV) for urban search and rescue (USAR) missions is being developed at Faculty of Electrical Engineering (FEE) and Czech Institute of Informatics, Robotics and Cybernetics (CIIRC); both under Czech Technical University in Prague, which is participating in European project TRADR. This robotic tracked vehicle is expected to be able to move in an unstructured, complicated terrain such as disaster sites and explore potentially dangerous areas, thus helping rescue workers. For this purpose, it is equipped by auxiliary tracks called *flippers* which are driven by independent servomotors. Flippers significantly enhance robot´s capabilities to overcome wide variety of obstacles as their attitude and compliance may be adjusted according to traversed terrain.

Although flippers may be controlled manually by a robot´s operator, this additional task, which requires operator´s attention, makes his work more difficult. It is more convenient that the operator only controls robot´s heading and forward speed and flippers are being autonomously adapted to the terrain´s shape so that the robot moves fluently across the terrain. We call such behaviour adaptive traversability (AT). This proposed semi-autonomous control system reduces cognitive load onto operator who may focus on higher level tasks instead.

One of crucial conditions for adaptive traversability algorithms, which had been developed at FEE so far [1] [2], to work properly is having a reliable model of surrounding environment. Laser range finder is able to provide necessary information for most of the time. However, there are situations, that may happen, in which laser scanner fails to detect objects in front of the robot, typically in case of reflective surfaces such as water or in the presence of smoke. Such situations are not uncommon in USAR missions, so it is better to be prepared on them.

Therefore the robot has been equiped by a robotic manipulator and possible utilization of this manipulator is investigated within this thesis. The basic idea is that whenever the robot has incomplete data about the terrain in front of itself, for example when crossing a stream, the robot performs tactile exploration of the terrain using the robotic arm. Its model is updated by these measurements and consequently the robot may decide whether it is safe to continue in terrain traversal and what flipper configuration to use.

2. TRADR project

This thesis deals with driving a tracked mobile vehicle with articulated flippers and its goal is to enhance robot´s ability to overcome rugged terrain even with incomplete data about surrounding environment. It is a subproblem which has been encountered during a development of unmanned ground vehicle for TRADR project in which Czech Technical University is one of affiliates. Therefore this chapter is devoted to the TRADR project.

Fig. 2.1: TRADR logo. [3]

TRADR is an acronym which stands for Long-Term Human-Robot Teaming for Robot-Assisted Disaster Response. It is an integrated research project funded by the EU FP7 Programme, ICT: Cognitive systems, interaction, robotics (Project Nr. 60963). Being a direct successor of NIFTi project, it is based on previous experience and carries on with the research. [3] Its main aim is to develop science and technology which would allow to make human-robot teams and to interact with each other in order to achieve shared goals. Both, NIFTi and TRADR, are primarly focused on tasks in Urban Search and Rescue (USAR) missions in which people and robots work together to explore a disaster area, assess the situation, locate victims or prevent further damage.

2.1 TRADR Consortium

There are 12 partners from 6 countries collaborating on the project including 5 universities, 3 research institues, 1 industry partner and last but not least 3 end-user organisations—

firebrigades from Germany, Italy and the Netherlands. All of participating institutions are listed in the table below, see Tab. 2.1.

Tab. 2.1: Institutions participating in TRADR project. [4]

German Research Center for Artificial Intelligence
Delft University of Technology
Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS
Royal Institute of Technology (KTH), Stockholm, Sweden
Swiss Federal Institute of Technology
Czech Technical University in Prague
La Sapienza University of Rome
Ascending Technologies GmbH
Stadt Dortmund
Corpo Nationale dei Vigili del Fuoco (CNVVF)
Openbaar Lichaam Gezamenlijke Brandweer
Dutch Organisation for Applied Scientific Research TNO

2.2 TRADR mission

The ultimate goal of the TRADR project is to build a team of robots which would be able to cooperate together with people, mainly firefighters and rescue workers, in a first response disaster scenarios, for example in case of industrial accidents, earthquakes, fires, etc. The greatest asset of using robots in such scenarios is that they may enter and explore areas which are potentially dangerous for human workers. Thus they lessen jeopardy of workers and prevent bad situation getting even more complicated.

Two types of robots are developed within the project—unmanned ground vehicles (UGV) and unmanned areial vehicles (UAV). While UAV may execute quick exploration of large area and provide overview from higher perspective, UGV is able to overcome piles of rubble and get to hardly accesible places looking for victims and gathering physical samples. Knowledge about disaster area is then gradually built in direct cooperation with human rescuers which results in better understanding of the situation and consequently taking viable decisions in critical moments. This should improve overall quality of the sortie and increase chances to succesfully secure the area and save human lives. [3]

2.3 TRADR Joint Exercises

As TRADR project is intended to develop robots which should be particularly used by rescue workers, joint exercise is held every year (already since the beginning of previous NIFTi project). These meetings of developers and USAR organizations are fundamental, because the both groups come into close contact and may exchange valuable experience. Rescuers may team up with robots to perform on training scenarios which enable evaluation of current progress in research and also help with identification of shortcomings. Above all, USAR workers, i.e. end-users of the whole system, have an opportunity to contribute with valuable insights in order to help developers to improve the system and make it actually useful in real-life situations. Over the course of the project, testing scenarios are intended to grow increasingly challenging with more complex circumstances, taking into account progress of the project.

Last such meeting was held from 23rd of September to 2nd of October at the Tremora hospital, ex American hospital of Calambrone, near Pisa, Italy, in cooperation with the TRADR partners and the firefighters from the Firebrigade of Pisa. [5]

2.4 Niftibot

2.4.1 Platform description

Unmanned ground vehicle (UGV) used in TRADR project is based on Niftibot platform produced by BlueBotics company [6], see Fig. 2.2. It is a tracked mobile robot with one pair of main tracks, each equipped with own motor to provide traction. The tracks are mechanically linked to the body via brackets and a differential. When one track is rotated, the differential induce an inverse rotation in the other track in order to maximize contact with the ground. Four additional articulated tracks, *flippers,*are connected to the main tracks. The flippers are driven by their own servomotors which may be controlled indivudually to change flipper´s attitude and compliance (by controlling maximal allowed torque) and thus improving robot´s traversability capabilities to overcome rugged terrain. [7]

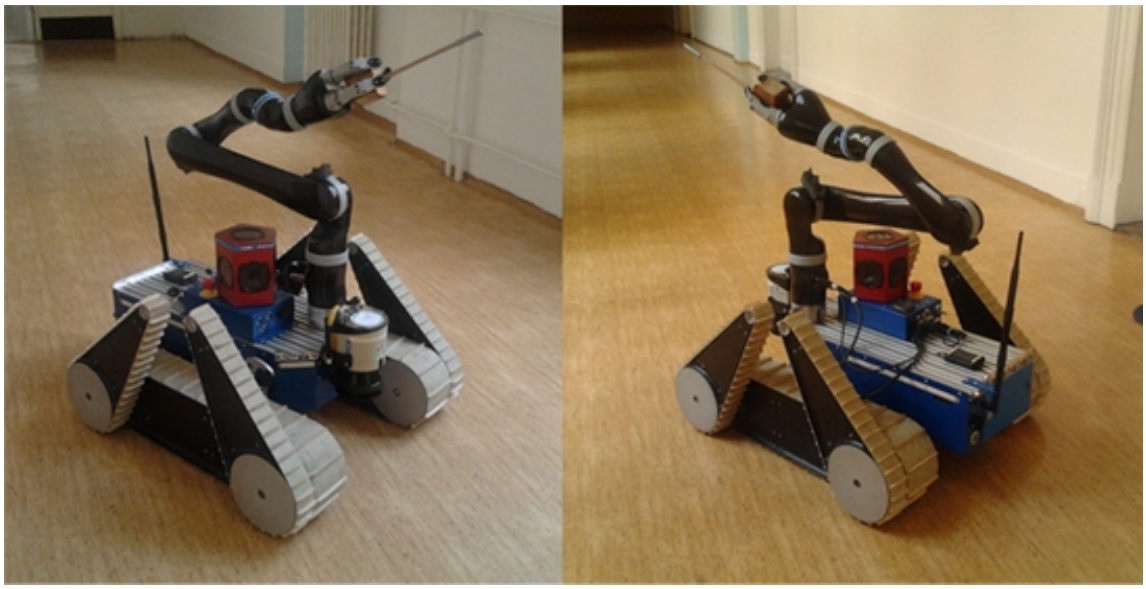

Fig. 2.2: UGV used in TRADR project (Niftibot platform) with mounted Kinova JACO robotic arm.

The robot´s body contains main electronic board of the platform with an embedded PC, a power PCB used for power distribution and a 3D PCB which manages rotation of the 3D sensor. Ubuntu Linux is installed as an operating system on the internal computer with Robot Operating System (ROS) running on top of Linux. USB and DVI connectors may be used to connect I/O peripherals to the robot´s computer and work in OS, for example installing new drivers, updating obsolete ones, uploading new programmes and scripts, etc. [7]

Next, the body contains a slot for a LiPol battery with two connectors, which means that two batteries may be connected at the same time. Although only one battery fit in the slot, this is particularly convenient, when it is needed to change depleted battery. It allows to hot-swap batteries while the robot is still running. Furthermore, there is also a connector for an external power supply in the back of the body. An emergency button is located on the top of the body dissconnecting power sources in case of safety issues. There are also LED diodes which may be used either as indicators or as a light source thanks to their high luminous power. [7]

Top of the body is manufactured in the way that additional features may be mounted on the robot. Kinova JACO Arm, which is utilized in this thesis, is one example of such feature, but also another sensors (like RGBD camera, infrared camera, etc.) or actuators (like different manipulator) may be installed as well.

2.4.2 Sensor suite

Ladybug 3 omnicamera

Ladybug 3 is a high resolution spherical digital video camera system. It consists of six MP cameras which enable to cover even more than 80% of 360° sphere together. They all are fit in a weather-resistant case allowing usage in adverse conditions. [8]
Camera provides panoramic image which is accesible by ROS nodes on ROS topic /viz/pano/image.

Sick LMS-151 laser scanner

LMS-151 is a laser measurement system. It is rotating 2D laser range finder with rotation from -90° to 90° from origin position. It is ultimately providing 3-dimensional point cloud constructed from all plane scans. The point cloud is accesible by ROS nodes on ROS topic /dynamic_point_cloud.

XSens Mti-G IMU

It is a MEMS based orientation sensor with integrated GPS. It provides odometry data like position, heading and velocity computed in real-time. The data are accesible by ROS nodes on ROS topic /imu/data.

2.4.3 Kinova JACO Robotic Manipulator

JACO robotic arm is one of the additional hardware mounted on UGV to enhance its capabilities. The possibility of utilizing the arm for tactile exploration of environment is a challenge currently being studied and solved at FEE and CIIRC (CTU). The other usage for an arm may be to pick up samples or manipulating with an environment.

It is a lightweight 6-DoF robotic arm with hand-like gripper produced by Kinova Robotics [9]. Thanks to carbon fibre structure it weights only 5.3 kg and therefore it can be carried by the robot with ease and it is also durable and able to endure adverse conditions during USAR missions.

In addition, Kinova offers Windows and Ubuntu compatible API for easier programming of the arm in C++ (or C#) and allows programmers to configure advanced parameters and integrate the arm to their own systems. Kinova has produced even a ROS driver for the arm, which is one of the reasons, why this manipulator has been chosen for our UGV. [9]

3. Robot operating system

Robot Operating system (ROS) is utilized throughout TRADR project, because it provides well-designed framework for writing modules of robotic software that is convenient for such large-scale project.

ROS is an open-source platform particularly employed by robot developers for writing robot software. Despite its name it is not literally an operating system, but it runs on top of existing one whereas Ubuntu Linux (currently its 14.04 LTS release) is officially supported. In other words, it is a robotics middleware like Player Project or OpenRDK.

"*It provides services that one would expect from an operating system including hardware abstraction, low-level device control, implementation of commonly used functionalities and message passing between processes. Furthermore, it provides a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust behavior across a wide variety of robotic platforms.*" [10]

Fig. 3.1: Official logo of ROS. [11]

3.1 Basic Concepts

ROS framework provides mechanisms to easily implement considerable number of processes which are supposed to run in parallel and communicate with each other at the same time. The processes are not even requiered to run on a single machine, which may come in handy particularly in robotics, because robots may be composed of multiple modules, each having own computer controlling sensors and/or actuators. This distributed approach to control may help to create autonomous AI for a robot or even a group of robots coopearting on a common goal. The system may be furtherly extended by interface for a human operator, who may then command robot(s) from his workstation. [10]

Another key idea behind ROS is to ease sharing advancements and codes in the community of robotics developers; does not matter if they are scientists researching in the field, industry

professionals or just hobbyist. Firstly, ROS framework defines uniform interface which is practically becoming a standard for robot software interoperability and it is widely known among the developers. [11] This uniformity and unicity of ROS allows seamless collaboration of several contributors on one project which is also the case of TRADR and it is exactly the reason why ROS has been chosen for such extensive project.

Secondly, the whole ROS is open project and anyone may contribute to the general progress. Thanks to that, there are many ROS packages from many authors publicly available on the ROS website for all users. [12] One may browse either ROS standard packages for stable, debugged implementations of algorithms used in robotics for all common tasks (such as planning, mapping, navigation, building world models, etc.) or other ROS packages with experimental cutting-edge algorithms. One way or another, developer does not have to spend much time reimplementing already well-mastered algorithms on his own and he may rather focus on another tasks which have not been satisfactorily solved yet. [10]

Community using ROS is growing in numbers which means, among others, that anyone may get some support in case of problems from other users of the system, either on official forum or other Q&A sites like Stack Overflow. There is plenty of information on ROS provided by ROS developers on official web-site [11] including tutorials, extensive documentation and reference manual describing usage of ROS functionalities in detail.

Another huge advantage of ROS is that it enables easier testing of new algorithms and their different implementations. The authors of ROS are aware that testing in robotics may be time-consuming and error-prone. In addition, practicle experience suggest that robots may not be available for testing for considerable amount of time due to problems with hardware or other parts of software or the testing may be simply dangerous and expensive. This is where well-designed ROS system enables to separete low-level control of robot and high-level processing. Developer then has two possibilities of testing high-level reasoning. Firstly, he may use simulator which substitues low-level part of the system. Secondly, he may use *rosbag* utility. This option facilitates recording robot´s sensor data while driving the robot. Later the same data may be replayed again infinite times and used to compare performance of different approaches to processing them. [10]

Finally, ROS offers some built-in tools for visualization that may help in development and debugging of a project, for example *rqt_graph*, *rqt_plot*, *image_view* or *Rviz*.

3.2 Framework

3.2.1 Package

Package is a fundamental term in ROS representing a base unit of ROS software organization. In practise, it is a single directory which may contain almost anything, most likely executables, scripts, libraries, tools, etc., but every time it includes a manifest file (named package.xml in catkin build system or manifest.xml in rosbuild) and CmakeLists.txt. Manifest is an xml structured file defining basic properties of package—name, version number, atuhors, maintainers and dependencies on other packages—and it is particularly this file is what make difference between ROS package and any generic directory. [10]

Usually one project comprises of many packages. Each of them serve as logical unit, one piece of puzzle, and has its own purpose. One may be used for mapping while other for planning and the third one for driving the robot. And all of them may depend on each other in some ways, thus making a complex system altogether. The division into packages significantly contributes to easier orientation in vast projects and in addition well-maintained package may be reused in different projects on different robotic platforms.

3.2.2 Node

Node is an elementary part of every ROS project. It denotes launched instance of practically any executable within any ROS package; either compiled from C++ source or being a Python script (or lisp, java, Matlab,...). It may be launched anytime after Roscore has been initiated in order to perform process it is supposed to do. The most significant difference between ROS node and any other executable file is that ROS node is able to connect to ROS master and consequently communicate with other nodes. [11]

The advantage of ROS is that nodes may be written in different languages, e.g. in C++, Python, Java, Lisp, Octave (Matlab), yet they are able to mutually communicate thanks to ROS client libraries (roscpp, rospy, rosjava, roslisp, rosmatlab). Therefore the whole project does not have to be written in an only one language.

3.2.3 Roscore

Roscore is initiated by typing *roscore* command in terminal window and it has to be done everytime before any of ROS nodes starts. It consists of Master, Rosout and Parameter server.

ROS Master is a naming and registration service for ROS, i.e. it helps nodes to find each other. It tracks publishers and subscribers to all topics as well as services. Communication between two nodes over topics (or services) is done on peer-to-peer basis, once the both are connected with each other by Master.

Rosout is ROS equivalent of stdout and stderr and is used particularly for debugging.

Parameter server is a multi-variate dictionary. Nodes may use the server to store and read parameters at runtime. It is best used for static data such as configuration parameters. (On the other hand, exchange of dynamic data between nodes is better done via topics or services.) [11]

3.2.4 Topics and messages

Topics and messages are one way of communication between nodes provided by ROS framework. Any topic may be viewed as a uniquely named channel for streaming messages. Message itself is a data structure containing at least one (but often more) field, which may be of an arbitrary type. ROS supports common standard types as integers, floating points, booleans, strings as well as arrays and even other messages nested within a message. Programmer then may choose to use either from standard ROS message types or he can prepare user-defined messages in his package saved as *.msg files. Programmer also should not forget that one topic is restricted to transfer messages of only one type, i.e. type of message determines type of topic. [11]

A node, which wants to send messages to other nodes is said to be *publisher* and it is publishing messages on the topic. A node which is supposed to receive and process messages is said to be *subscriber* and it is subscribing to the topic. There may be even multiple publishers and subscribers to one topic at the same time. [11]

ROS tools for dealing with topics and messages are called *rostopic* and *rosmsg* and they allow user to show information about currently available topics; for example what node(s) is (are) publishing, what node(s) is (are) subscribing to specified topic, what type of message is passed, publishing rate, trafic volume, etc. *rqt_graph* is also helpful tool which graphically visualize network of communicating nodes and topics.

3.2.5 Services

Services are the second way how nodes may communicate with each other. While communication via topics is unidirectional on many-to-many basis from publishers to subscribers (publisher does not await any response; in fact it does not even require actual existence of any subscriber to the topic), services enable bidirectional communication.

Besides, service calls implement strict one-to-one communication. One node is in a role of a *client*. It sends a request to other node which is in a role of a *server*. The server processes the request, it performs an action (e.g. computes something, configures hardware, makes a measurement, etc.) and finally it sends back a response which the client is waiting for. [11]

To ensure that the client and the server understand each other, special data structure *srv*, defined in custom *.srv file, is used. It is composed of a pair of messages—request and response. The both contain fields which may be of an arbitrary type analogically to messages used on topics. [11]

ROS tools for dealing with services are called *rosservice* and *rossrv* and they allow user to show information about currently available services. These tools are analogical to rostopic and rosmsg.

4. Reinforcement learning

Reinforcement learning (RL) is widely used within this thesis. At first, it is used to learn a model for adaptive traversability which chooses optimal flipper configuration depending on robot´s state and terrain features. Secondly, it is used to learn a strategy for tactile exploration of terrain to speed up the process. Therefore this chapter is devoted to reinforcement learning.

4.1 Markov Decision Processes

Markov Decision Processes (MDP) are commonly used models for reinforcement learning problems. They provide a decent level of abstraction, thus they simplify description of a problem, but in spite of that they are sufficently robust and flexible to describe a large set of real-world problems. [13] MDP is defined as a tuple

$$MDP = \langle S, A, p(s'|s, a), r, s_0 \rangle \quad , \tag{4.1}$$

where S is a set of possible states in which an agent may find itself, A is a set of possible actions which the agent can do, $p(s'|s, a) : S \times A \times S \to [0, 1]$ is a transition probability that the agent will get to state s' if it performs action a in state s, $r(s, a, s') :$ $S \times A \times S \to \mathbb{R}$ is a reward which the agent obtains when it reaches state s' from previous state s using action a and $s_0 \in S$ is an initial state. [14]

Let us denote $s \in S$ a single state and $a \in A$ a single action. Basic idea of reinforcement learning problem is that the agent (i.e. decision maker and learner) and environment interact continually. Agent selects an action and environment is responding —introducing the agent to a new state and giving reward to the agent, see Fig. 4.1. The evaluative feedback is one of the most important concepts of reinforcement learning. The learner is told how good an action is in terms of reward, which is received after performing the action. It is in contrast with instructive feedback used in supervised learning, where learner is told what action is optimal. [13]

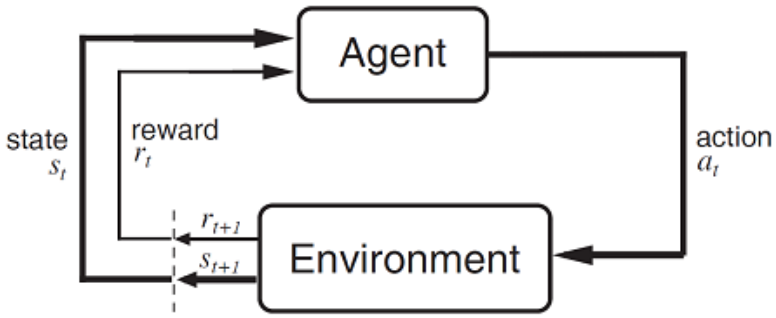

Fig. 4.1: Interaction between agent and environment during reinforcement learning. [13]

A sequence of visited states and actions performed by the agent is called a trajectory τ:

$$\tau = (s_0, a_0, s_1, a_1, s_2, a_2, \ldots) \quad . \tag{4.2}$$

At each time step, agent receives a reward $r_t(s_{t-1}, a_{t-1}, s_t)$ which is a simple real number. Agent is then supposed to collect as much reward as possible throughout its actions. Optimally, it should avoid greedy actions with high immediate rewards and rather choose actions that would maximize cummulative reward in longer time horizon (along the whole trajectory). Therefore a return R of rewards when agent follows trajectory τ beggining at time t is introduced:

$$R_t(\tau) = \sum_{n=0}^{N} \gamma^n \cdot r_{t+n} \quad . \tag{4.3}$$

It is a sum of discounted rewards from time t onwards. N denotes time horizon, i.e. how many steps to the future are considered (in some cases N may be equal to infinity meaning no time horinzon at all) and $\gamma \in (0, 1]$ is a discount factor which is used to lessen influence of states in far future.

Another important concept used in MDP is a policy $\pi(s) : S \rightarrow A$ which is a function mapping a state to an action (in deterministic case) or to a probability $p(a_t = a|s_t)$ of selecting action a in state s (in stochastic case).

The ultimate goal of learning is then to find an optimal policy in terms of maximizing expected return:

$$\pi^*(s) = \arg\max_{\pi} E\{R(\tau)\} = \arg\max_{\pi} \int_{\tau \in T} p(\tau|\pi) \cdot R(\tau) \cdot d\tau \quad . \qquad (4.4)$$

Term $R(\tau)$ appearing in formula (4.4) is a return R (4.3) obtained after the agent was following trajectory τ starting from state s when policy π was in charge. As transitions between states may be stochastic and therefore different trajectory may occur every time the same policy is forced, mean value over probability distribution of all possible trajectories induced by policy π is considered. (T denotes a set of possible trajectories and $p(\tau \mid \pi)$ denotes probability that trajectory $\tau \in T$ occurs when policy π is used.)

4.2 Q-function

Some of reinforcement learning algorithms are based on estimating value functions of states for policies. Generally, there are two types of value functions. The first type is *state-value* function for policy π, which quantitatively estimates how it is advantageous or disadvantageous to be in a certain state when policy π is used. It is an expected return when the agent is using policy π starting from state s:

$$V^{\pi}(s) = E\{R(\tau) \mid s_0 = s\} = E\{\sum_{n=0}^{N} \gamma^n \cdot r_{n+1} \mid s_0 = s\} \quad . \qquad (4.5)$$

The second type is *action-value* function (also frequnetly reffered as *Q-function*) for policy π, which quantitatively estimates how it is advantageous or disadvantageous to perform certain action a in specific state s and then follow policy π:

$$Q^{\pi}(s,a) = E\{R(\tau) \mid s_0 = s, a_0 = a\} \quad , \qquad (4.6)$$

$$Q^{\pi}(s,a) = E\{\sum_{n=0}^{N} \gamma^n \cdot r_{n+1} \mid s_0 = s, a_0 = a\} \quad . \qquad (4.7)$$

A fundamental property of value functions is that they satisfy particular recursive relationship—value function at one state s may be determined as a combination of reward for getting to another state s´ and value function at the new state discounted by factor γ. If we take into account stochastic transitions between states, we may bring together following formula for Q-function, which is called Bellman equation: [13]

$$Q^{\pi}(s,a) = \sum_{s'} p(s'|s,a)\cdot\left[r(s,a,s') + \gamma \cdot Q^{\pi}(s', \pi(s'))\right] . \qquad (4.8)$$

Analogical formula may be derived for state-value function as well.

Defining Q-function allows us to estimate what actions are useful in particular states. As it takes into account possible succesor states, it estimates how much reward agent may acquire over long run; not just by a single greedy action. For example, this property may prevent getting ultimately to severely bad state due to choosing greedy actions in the first place and not thinking about possible future consequences.

Having known Q-function, we may compare two policies—we may argue that one policy is better over another when it has higher Q-values for all (s,a) pairs:

$$\pi \geq \pi' \text{ iff } Q^{\pi}(s,a) \geq Q^{\pi'}(s,a) \text{ for all } (s,a) . \qquad (4.9)$$

Naturally, the ultimate goal is to learn what policy is the best one, looking for optimal action-value function:

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) . \qquad (4.10)$$

Finally, we may formulate Bellman optimality equation, which describes realtionships between optimal Q-functions of state-action pairs: [13]

$$Q^{*}(s,a) = \sum_{s'} p(s'|s,a)\cdot\left[r(s,a,s') + \gamma \cdot \max_{a'} Q^{*}(s', a')\right] . \qquad (4.11)$$

4.3 Reinforcement Learning process

MDP tasks may be solved by dynamic programming methods, described for example in [13]. However these methods require complete and accurate model which describes interaction between robot/agent and environment. Unfortunately this assumption is not often satisfied, particularly probabilities $p(s'|s, a)$ remain unknown in many real-world problems. Secondly, they compute Q-values based on iterative approach (via Bellaman equation) using sweeps through complete sets of states and actions, which may be prohibitively expensive.

Great advantage of reinforcement learning methods, including Q-Learning, lays in the fact that they draw from experience and particularly does not require any prior knowledge about environment and its dynamics. [15] Learning is run in episodes. Agent is following some policy in each episode and trajectory τ is recorded along with obtained rewards. For each pair (s,a) visited within the episode, Q-value $Q(s,a)$ is then updated by obtained return R (see equation 4.3) following the visit. Optimal policy π^* is updated accordingly to be consistent with actual Q-values. [13] [15]

One of critical matters in reinforcement learning is a choice of policy π which will be used to select actions in states occuring within an episode. One the one hand, agent should greedily follow optimal policy π^* computed so far which ensures relatively high rewards. On the other hand, there is no guarantee that such policy is really optimal; it is just a temporary assumption. There may be some better policy which has not been found yet. It is a problem of balancing exploitation (using the best known policy) and exploration (trying other actions and looking for potentially better policies). There are many approaches how to deal with it, however this topic is still a subject or research and there are many unresolved questions. [13]

Probably the simplest approach, however quite effective, is ε-greedy strategy. Rule which is used to select actions is simple. The greedy action is performed for most of the times, but also a random action may be chosen with a little probability ε. This behaviour enables us thoroughly explore optimal strategy and strategies which are close to the optimal one. In other words, $Q(s,a)$ is preferably estimated for potentionally interesting state-action pairs while state-action pairs with no promising asset, which are not likely to be visited at all, may not be explored or they are tried only several times (just to figure out that they are useless). This is a huge advantage over dynamic programming which has to perform systematic sweeps over the whole set of state-action pairs to estimate Q-values.

There exist several learning algorithms used for *Q*-learning. The one which has been utilized in this thesis to learn exploration strategy using robotic arm is denoted as On-policy Monte Carlo Control in [13].

On-policy Monte Carlo Control algorithm [13]

Initialize:

For all $s \in S$, $a \in A(s)$ do:

$Q(s,a) \leftarrow$ arbitrary

$Returns(s,a) \leftarrow$ an empty list

$\pi \leftarrow$ an arbitrary ε-soft policy

Repeat forever:

Generate an episode using policy π

For each pair (s,a) appearing in the episode do:

$R \leftarrow$ return following first occurence of (s,a) in the episode

Append R to $Returns(s,a)$

$Q(s,a) \leftarrow average(Returns(s,a))$

For each s appearing in the episode do:

$a^* \leftarrow \arg\max_a Q(s,a)$

For each $a \in A(s)$ do:

$$\pi(s,a) \leftarrow 1 - \epsilon + \frac{\epsilon}{|A(s)|} \quad if \quad a = a^*$$

$$\pi(s,a) \leftarrow \frac{\epsilon}{|A(s)|} \quad if \quad a \neq a^*$$

Note: $\pi(s,a)$ denotes probability of choosing action *a* in state *s*

5. Terrain traversability—State of Art

This thesis deals with a movement of an unmanned ground vehicle (UGV) in an unknown environment. Extending abilities of mobile robots to overcome an unstructured terrain beyond paved roads is a challenging issue which has an attention of many robotics experts and novel systems are being developed particularly since the beginning of the century.

It is a complex problem in which both, hardware and software, play vital roles. Mechanical construction of a robot is as important as a control system. The overall succes then depends on synergy of the both. Following sections cover several approaches and advancements in building chassis of mobile robots at first. Then a major part of the state of art review focuses on intelligent systems for traversing rough terrain.

5.1 Chassis construction

Generally, when designing a mobile robot, choice of suitable hardware is a first step in the process. We should take into consideration characteristics of an environment in which we assume the robot will work as well as a puropose of the robot itself. This particularly holds for urban search and rescue (USAR) missions, in which we want to ensure high mobility in rough, uneven and unstructured terrain. Therefore, we should think about chassis first.

Usually, three types of mobile mechanisms for robots are distinguished—wheels, legs and tracks. Wheels work well especially on even surfaces, where they enable robot to move fluently, effectively and fast. However, they may lack ability to move across complicated terrain. On the contrary, legged robots have a huge potential for such situations, but it is very difficult to control them thanks to many degrees of freedom and their movement is slow so far. But this field is a subject of modern research [16] [17] and it is likely that there will be many legged robots with amazing agility in the future. For now, the most popular mechanism for robots used in USAR are tracks, which offer enough robustness to move across rugged terrain combined with fair speed. [18] [19]

Tracks are able to provide good traction for most of the time, even on muddy or sandy surfaces, but sometimes problems may occur, when contact with ground is minimal. Such situations may happen, for example, when it is needed to overcome steps, boulders, fallen branches, depressions, etc. Some enhancements may come in handy.

For example, Lee, Kang and Kim designed a robotic platform ROBHAZ-DT3 [18], which is composed of two pairs of tracks, which are interconnected using passive joint, see Fig. 5.1. Relative position of front and rear tracks is being passively changed according to shape of traversed surface. It does not only provide a better contact with the ground, but also center of gravity is shifting, giving better stability to the robot in comparison with single tracked platforms.

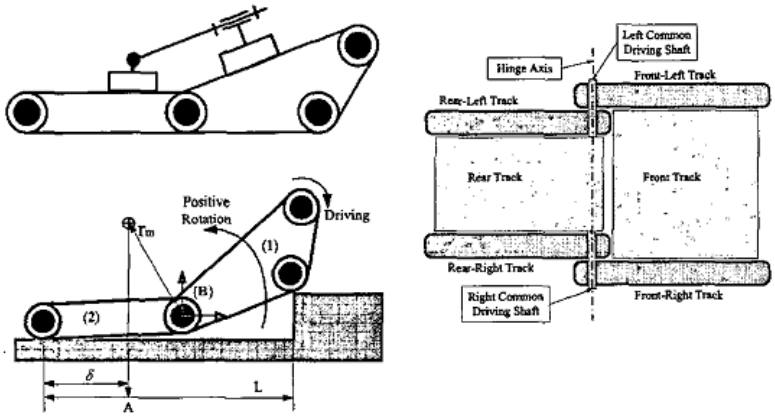

Fig. 5.1: Mobile robot platform ROBHAZ-DT3. [18]
Left – view from side. Right – view from above.

Passive machanisms, like the one mentioned above, have the advantage that they do not require additional control systems. On the other hand, they may get stuck, e. g. in some depression, and may not be able to escape. Actively controlled mechanisms represent an alternative, which brings new options for robot movement. [20]

Yamada et al. [21] got inspired by a wharf roach and designed a robot combining some passive and some active mechanisms. They refer to their design as blade-type crawler mechanism because flexible blades are mounted on tracks. These enhance robot´s ability to overcome some minor obstacles, see Fig. 5.2, and also serve as dumpers, when robot falls. In addition, robot is equiped with actively controlled antennas, which may even help robot to jump in some occasions like the animal.

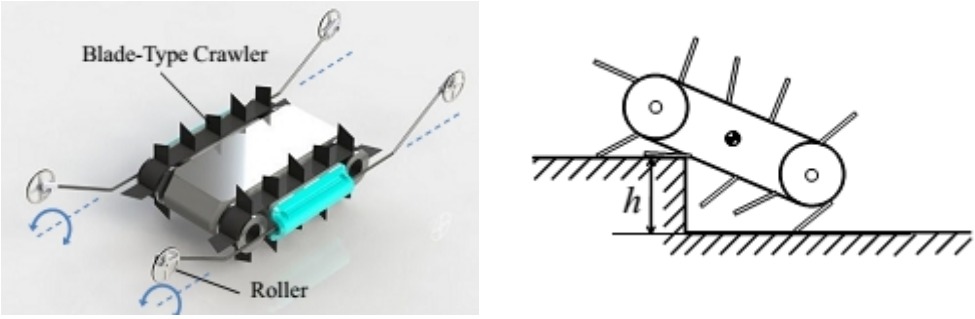

Fig. 5.2: Blade-type crawler mechanism inspired by a wharf roach. [21]

Guarnieri, Debenest et al. [19] introduced an interesting mechanism applied on robot Helios Carrier, which was equipped by actively controlled tail-like mechanism, see Fig. 5.3, which particularly served as support preventing robot from flipping over. They succesfully tested their design during climbing and descending stairs.

Fig. 5.3: Robot Helios Carrier with actively controlled tail-like mechanism. [19]
Left: Robot overview. Right: Climbing stairs without and with tail-like mechanism.

Finally, a popular approach, which may be considered as an extension of the mechanism introduced in the previous paragraph, is a design using two pairs of articulated side tracks—often reffered to as *flippers*. This type of chassis is widely used for search and rescue robots including UGV from NIFTi/TRADR project. [1] [20] [22] [23] [24] (Or there is an another popular platfrom Packbot produced by iRobot company which uses only front flippers. [25]) Attitude of each flipper is controlled by its own servomotor. One can also alter flippers´

compliance by limitng maximal current (and therefore limiting maximal generated torque). Flippers significantly enhance robot´s mobility as they allow to traverse larger set of heterogenous obstacles, from which some of them may be even impossible to overcome by other types of chassis. [26] This variability, however, demands more sofisticated control. It is the aim of this thesis to propose a solution, how to automatically control flippers for smooth overcoming of obstacles, but first, there will be introduced state of art on this topic in the following sections.

5.2 Artificial intelligence for autonomous traversal

There were several types of chassis for UGVs, which enable movement on more complicated terrain, presented in the previous section. Constructions with actively controlled elements naturally offer more options, but their control is more difficult. This brings higher demands on robot´s operator, especially when robot is being driven remotely and information of robot´s situation provided to operator on his workstation is limited. It is slowing down operator´s work and a risk of mistake is higher. Therefore one of the main goals of current research is to automatize robot´s movement and lower cognitive load of operator, who may consequently focus on higher level tasks required by the mission instead of paying extra attention to driving the robot. [2] [24]

If we want robot to have a certain level of autonomy, we cannot forget to develop intelligent software, that will be able foremost to process sensor readings (both proprioceptive and exteroceptive) and utilize them into a model of environment. Secondly, it should be able to evaluate negotiability of the terrain based on the model and eventually to choose proper configuration of adjustable chassis.

5.2.1 Planetary rovers

One of the first application domains of mobile robots in which a problematics of save terrain traversal was emphasised was usage of planetary rovers. [27] [28] Increased caution during movement in an unknown environment is necessary in their case beacuse of high cost of the mission and a fact that they have to get along without any assistance. If rover lost its stability and tipped over, it would mean an early end of mission and therefore bad succes.

This was the reason, why scientists started to think about how to minimize a risk of an accident.

Negotiability of terrain is evaluated as a part of an autonomous navigation and a path planning in the case of rovers like in work of Singh, Simmons et al. [27], who designed planning algorithm for Mars Rover. They encountered a problem that previous researches concerning path planning for UGVs had divided space only into free space and obstacles. While this binary division may be sufficent for indoor robots, it ceases to be efficient in a complex environment. One can find a wide variety of obstacles outdoors. Some of them may be impassable like large objects or deep pitfalls and it is necessary to avoid them, while some other obstacles may be overcome by robot. One just have to take into account higher pitch and roll when passing them which usually means higher risk and energy consumption, which should be incorporated into cost function for a planner.

Concretely, Mars Rover is equipped with a stereoscopic camera used to map terrain. The map is divided into cells whose size is approximately the same as robots. Points obtained by the camera which fall into one cell are used to fit a plane via least squares method. This allows, among others, to determine eventual pitch and roll of the rover and a roughness of terrain from deviations of points from the plane. They also compute a certainty of a cell which encompasses number of points in the cell and their distribution. For example, a cell with only few points in one corner will have less certainty value than other cell with a lot of points distributed uniformly in its area.

The plan itself is being build on two levels—global and local. D* algorithm is used for global planning to prepare an initial plan. The local planner than evaluates different directions of movement from the current position to find the most suitable one. It is using parametrs from the map, certainty of map cells and mechanical properties of rover.

Algortihm for autonomous navigation was furtherly extended in the work of Goldberger et al. [28] who use a grid of cells with higher resolution. They also introduced some other parameters to describe cells such as step hazard, roughness hazard, pitch hazard and border hazard (cell which is neighbouring with unknown cells has high border hazard). They subsequently utilize all this information to determine optimal heading of rover in order to get closer to the goal and choose safe trajectory at the same time.

5.2.2 Search and rescue UGVs

Mechanisms developed for planetary rovers may be indeed applied on other UGVs including those designated for USAR missions. Nevertheless, rescue robots cannot manage with some simplifying assumptions, that may have been done for Mars Rover. Thats because of more dangerous character of their missions. Environment in which they are supposed to work is more complex and it is more difficult to make its model. [22]

5.2.3 Planning approach to traversability

One popular approach for autonomous overcoming of obstacles is determination of traversability (and choice of suitable configuration of chassis if possible) already during path-planning; quite similarly like in the case of Mars Rover.

There is a fair number of papers that focus mainly on building traversability cost maps for robots with fixed chassis, i.e. for robots which cannot adapt their chassis according to terrain (for both, tracked and wheeled). [29] [30] [31] [32] These maps are then supposed to be an input for path planning algorithms which are responsible for generating safe path. Researchers are usually exploiting either onboard stereocameras or laser range finders to obtain point clouds which are furtherly transformed into maps (mostly by fitting planes using least squares fit, Ransac, etc.). Relative difficulty of driving through certain region is assessed by various methods. Usually detailed information on robot-terrain interaction is estimated such as pose of robot on surface (in the computed world model, map), its stability, traction/slippage, power consumption, etc.

Kim, Sun et al. [33] proposed online learning method for terrain classifier that should distinguish traversable and untraversable terrain by wheeled robot. Their system performs collection of data by sterocamera as robot moves. They extract features from the data describing geometry of terrain together with image textures. The classifier then learns which features are corresponing to traversable terrain and which are corresponding to untraversable terrain. Terrain classification then provides information to the planner. Their approach is similar to ours in some ways, but they are learning model only to distinguish traversable and unsafe areas, while we are trying to learn model to assess several different chassis configurations and choose the best one. Secondly we are trying to exploit reactive behaviour rather than planning.

Brunner et al. pursue the problem of terrain traversability for robots with adjustable chassis [26], whereas they try to design a general algorithm, which would not be limited to just one type of chassis. Their solution is built on creating quality model of terrain. In their opinion, difficult parts of the path should be known in advance (while we argue that this condition may not often be satisfied in USAR missions), so that one can avoid dangerous situations in advance. Secondly, model in memory is needed in situations when robot cannot rely on actual sensor readings due to high pitch (and therefore looking in bad direction) or some noise in sensor data.

Again, their planner works on two levels. Global planner is looking for an approximate path and also performs classification of terrain on easy areas and hard areas. Its output is a graph roughly describing path. In the following phase, easy segments of path are not considered as they are accesible in any configuration. Local planner is used just on difficult segments, which needs thorough analysis.

For now, they consider only static properties of robot in the article [26]. Including dynamic properties (inertia, dynamic stability, ...) is intended to be a subject of future work. For now, they check static stability which they quantify by mechanical work needed to tip over the robot along least stable axis at that moment. This is determined for each possible configuration and incorporated into cost function used in planning algorithm for given vertex. They validated their approach in ROS Gazebo simulator, but not on real robot and real environment.

Colas et al. [22] have chosen more specific approach. They work with concrete robotic platform Niftibot. (Note: They are collaborating on TRADR project, therefore they are experimenting and developing systems for the same platform as we do.) They emphasis thorough processing of sensor data and building precise model too. The model is then used to determine reachable positions (i.e. robot does not collide with obstacles, has enough support from the ground and pitch and roll are within prescribed limits) using tensor voting method at first. Then they determine if it is possible to move between two positions and in what configuration. This makes a graph which is used as an input for D* algorithm.

Functionality of their solution was verified in an experiment in flat terrain at first to compare their planner with regular planners for 2D environment. Then they performed an experiment in fully 3D terrain during succesful attempt to climb a staircase. The extension

of the planner from 2D (2.5D) to 3D is considered as their most significant contribution by the authors.

Safe climbing of stairs is generally a popular challenge, which is tackled by more robotics developers working on many platfroms (not only tracked robots, but also legged robots as well). There are just some examples: Mihankhah et al. [23] work with tracked platform Silver similar to Niftibot equiped with two 2D laser range finders (horizontal and vertical). The lasers are used to detect a staircase. It is done by extracting lines from laser data and choosing candidate lines for describing stairs. Robot is then subsequently navigated to the staircase and is aligned in front of the stairs ensuring that robot will start climbing from a good initial posture. Finally, they use fuzzy logic based controller to drive the robot upstairs using laser data on input assessing relative posture of the robot to the stairs and keeping the robot parallel to walls on the side of staircase.

Hesch, Mariottini et al. [34] pursued stair descending for a change. Nevertheless, basic idea was similar. They utilize monocular camera to analyze scene geometry (lines) using machine vision algorithms in order to detect descending staircase. After that, the robot is navigated to the staircase followed by aligning itself with the edge and finally climbing down. The main concern is to keep the robot in the center and avoid tipping over. Information from tri-axial gyroscope combined with line information from camera is exploited by extended Kalman filter and a PID controller which should steer the robot following safe trajectory.

Although climbing/descending stairs is a challenging problem, the stairs are still structured and regular terrain. The both mentioned articles have in common that the proposed controllers detect stairs as lines in laser or camera measurements, but terrain in USAR scenario is expected to be much more irregular, therefore more robust approaches are needed.

5.2.4 Reactive approach to traversability

There were several papers mentioned in previous sections which deal with terrain traversability and adjusting configuration of robot´s chassis already during a planning procedure. On the one hand, this approach may prevent driving robot into potentially dangerous situations, but on the other hand, it may be sometimes computationally demanding, slow and not very flexible, requiring complete information about environment and precise model.

Situations in which we do not have comprehensive knowledge of the environment (for example due to dust or smoke covering sensors) are not exception as well. This makes planning much more difficult, because we are unable to recognize what part of map is reachable and what part is not. In addition, thorough plan may not be even needed. Most of the times operator does not request robot to autonomously plan the whole path and then realize locomotion from actual position to distant goal position.

Usually, search and rescue robot is under human control during its deployment for most of the time. It is the operator who is choosing direction and speed of the robot´s movement. Active auxiliary articulated tracks, flippers, may be controlled manually when needed as well. However, it is more convenient when flippers adaptively change their position with respect to a shape of terrain underneath and in front of the robot. That is, we rather aim to develop a semi-autonomous system instead of fully autonomous one. The system is not supposed to replace an expert operator, but it should play supportive role and ease operator´s work.

Similar philosophy to the solution of the problem concerning adaptive traversability may be found in works of Okada et al. [24] or Ohno et al. [20]

Okada et al. develop a control system of flippers for a tracked robot Kenaf, see Fig. 5.4, which has a similar construction as our platform Niftibot. These authors used two laser range scanners mounted on both flanks above main tracks in their older paper. [35] The lasers provided information about terrain under both, left track and right track, see Fig 5.5. Then their control system updated actual flippers´ positions using the information. This solution turned out to be insufficient, especially in situations, when there was a narrow obstacle in front of robot, which fitted into free space between front flippers. Robot crashed into the obstacle without noticing it earlier in that cases.

The authors later published another paper [24], in which they introduced enhanced system with third laser range scanner, which detects terrain in front of the robot, see Fig. 5.5. We may deduce from their experince, among others, that if we want to effectively control flippers for safe traversal, we do not need only information about terrain´s shape right under robot´s tracks, but also in front of the robot. This is an important idea which is used in our work as well as we utilize digital elevation map (DEM) which describes terrain under the robot and in front of the robot.

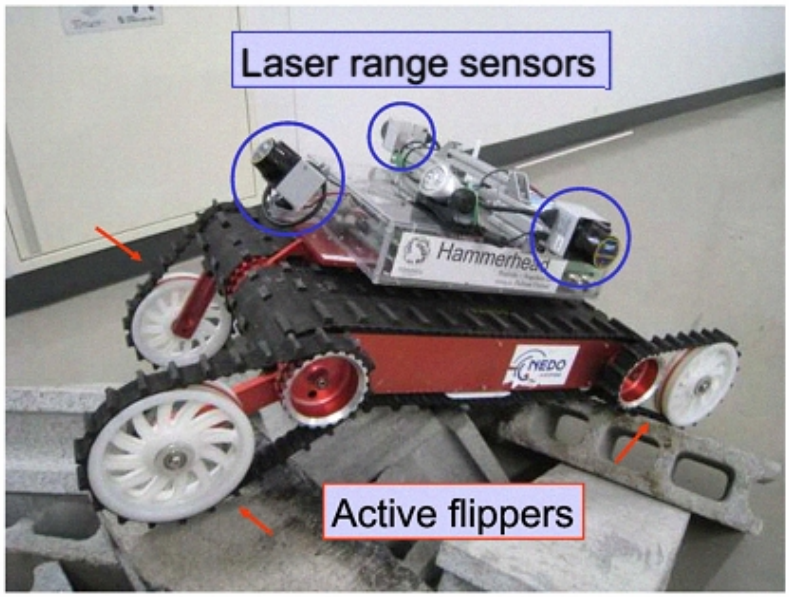

Fig. 5.4: Tracked robot Kenaf. [24]

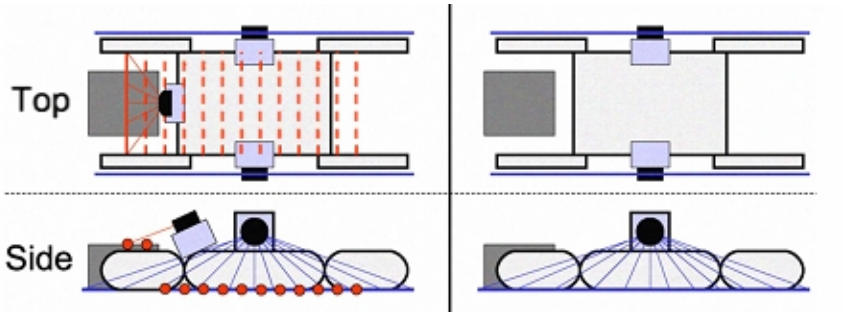

Fig. 5.5: Laser range scanners mounted on Kenaf. [24]
Left – New system described in [24] with a new additional sensor in front.
Right – Old system described in [35] using only two sensors on sides.

Algorithm which was applied in the paper is summarized in a schema in Fig. 5.6. At first, shape of the terrain under robot´s tracks is derived from laser scans and actual posture of the robot is measured by inertial measurement unit (IMU). These estimates are then used to assess a suitable configuration of flippers to overcome the terrain. A requested posture (if different from actual) is designated so that it is paralel with plane, which approximates terrain using a least squares method and so that the robot has enough support from the ground. Its reachability is also subsequently checked in a feedback loop, especially in the sense of stability. In the end, control signals are sent to servomotors in order to execute desired movement. A conventional PID regulator is used in this step.

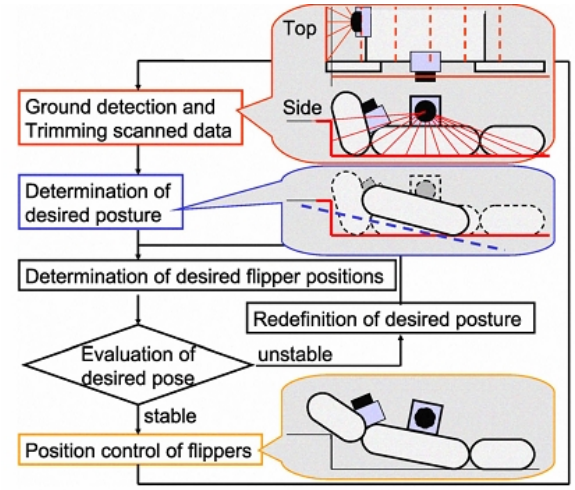

Fig. 5.6: Adaptive traversability algorithm introduced by Okada et al. [24]

Ohno et al. [20] also rather try to explore reactive behaviour based on actual data from sensors instead of building a complex model of the world and planning over this model. They use following sensor information: pitch of robot´s body, angles and torques of flippers, distance between front body and ground and distance between rear body and ground, see Fig. 5.7.

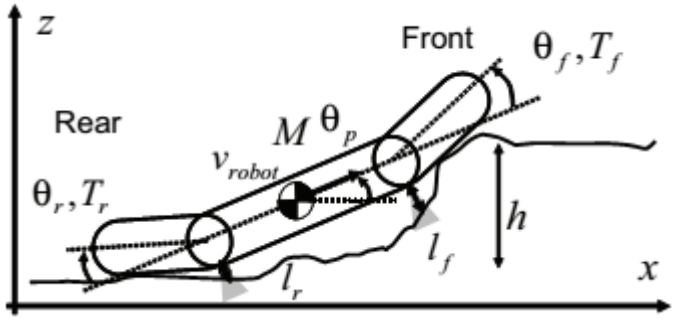

Fig. 5.7: Exploited sensor information in [20].

Pitch of robot´s body θ_p, angle of front flipper θ_f, angle of rear flipper θ_r, torque of front flipper T_f, torque of rear flipper T_r, distance of front (rear) body and ground l_f (l_r).

They also made some simplifying assumptions in their work. Firstly, they assumed only 2D situations—sagittal section of robot and terrain was considered (like in Fig. 5.7). This also

means that they did not take into account a roll of the robot. Secondly, they assumed that rubble and other terrain structures may have been approximated as steps in 2D.

Based on their assumtions, the authors designed motion sequences for getting smoothly over an upward step and analogically for getting past a downward step; see Fig. 5.8, which illustrates these sequences. They prepared several control rules for both, front and rear flippers, according to the sensor information at the moment. The rules were organized into two look-up tables. Several quantities are being continually observed, see Fig. 5.7: robot´s pitch, angles of flippers, distance between robot´s body and ground and contact between flippers and ground (which is assessed from robot´s inertia, posture of flippers and angular momentums generated by actuators). Appropriate control rules are chosen from the tables for front and rear flippers according to measured quantities. The rules themselves are implemented as PD controllers which drive flippers to optimal positions so that the movement of the robot is as smooth as possible.

They succesfully tested their system on some artificial obstacles made of wooden blocks.

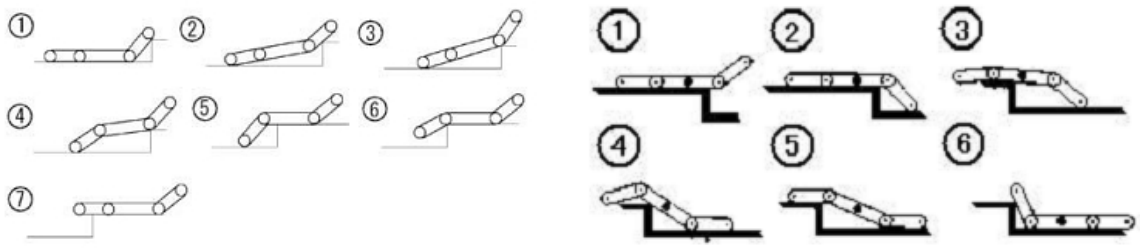

Fig. 5.8: Designed motion sequences for getting over upward step (left) and downward step (right). [20]

The advantage of proposed method is that it enables robot fluently cross steps of variable heights thanks to sophisticated control rules and PD controller optimized for such situations. Nevertheless, authors did not mention how their algorithm performed on general terrain shapes and we find utilizing 2D sections instead of 3D model as pretty strong assumption which is not valid in many situations. We also argue that such approach may be effective when climbing a single step, but the robot may have difficulties when driving more complex terrain.

6. Adaptive traversability

Mobile platforms which have adjustable chassis with actively controlled elements such as flippers have good capabilities to overcome various terrain features. However, they inevitably yield more degrees of freedom to control. It causes a high level of cognitive load of robot´s operator, thus making his task more difficult and error-prone. Therefore, semi-autonomous control system for adapting robot´s morphology to traverse terrain in an optimal way is being developed, so that operator may focus on other tasks than manually adjusting flippers´ positions. This behaviour is called adaptive traversability (AT). [1] [2].

The ultimate goal of this thesis is to take up current progress on semi-autonomous control system which has been done so far [1] [2] and further enhance its capabilities—mainly to extend the system by introducing active tactile exploration of not visible terrain by a robotic arm. Necessity of this extension has been suggested after previous experience. As the terrain was being scanned only by laser range finder before, situations in which robot was unable to detect objects in front of itself occurred. It may happen for example in case of reflective surfaces such as water and oil spills, in the presence of smoke or just in case of occluded view. Such situations are causing missing data in the model of environment, which is gradually being build from laser scans. As this model is then used by decision maker to choose optimal flipper configuration for terrain traversal, missing data may significantly spoil the decision. Mentioned situations are quite probable during USAR missions, so they deserve our attention.

6.1 Existing AT algorithm

6.1.1 Utilized features—Feature vector *F*

There are two types of features which are provided by robot´s sensors and may be used for decision making—proprioceptive and exteroceptive. The former group of features is provided by inner sensors. Namely following information is being utilized: pitch, roll, real speed of robot, speed requested by commands and currents in servomotors driving all 6 tracks (that includes main tracks as well as flippers).

Exteroceptive features describe terrain in close area around robot and are generated by 3D laser scanner. Concretely, we utilize terrain representation called Digital Elevation Map

(DEM). In our case, it is a discrete grid of size 2 m x 0.5 m which is divided into cells of size 0.1 m x 0.1 m, i.e. there are 20 x 5 cells, see Fig. 6.1. Each cell stores value of height of terrain at corresponding [*x, y*] coordinates. The area under the robot (orange in the figure) is cut from octomap (provided by octomap server) which is stored in the memory. The area in front of the robot is directly visible by laser scanner, therefore it is rather computed from dynamic point cloud than from octomap. (We experienced that octomap server does not perform well when dealing with dynamic environment as moving objects leave "trail" in the octomap. In other words, a moving object persists in the map even if it has already moved to a different place and it is spoiling the octomap.) Height in each cell at given [*x, y*] coordinates is computed as a mean value of *z*-coordinates of all points from the dynamic point cloud which fall into particular cell.

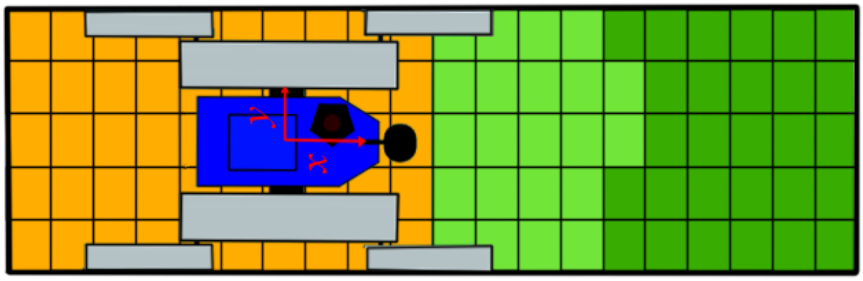

Fig. 6.1: Digital Elevation Map (DEM).

DEM is composed of 0.1 m x 0.1 m cells which form a grid of 20 lines and 5 columns (note that DEM is rotated by 90° degrees in the figure). Each cell stores value of height of terrain in particular place.

Values in orange colored cells are obtained from octomap provided by octomap server (one of ROS packages) while values in green colored cells are computed from dynamic point cloud. Light green cells are also measurable by Kinova JACO arm.

All features which may bring us potentially useful information are stored in the feature vector ***F***. It has following structure:

F = [100 x height values (each DEM cell), pitch, real forward speed of the robot, speed requested by command, 6 x currents in tracks, roll, 100 x counter (number of points used to compute height in corresponding DEM cell)]

6.1.2 Flipper configurations

Control of attitude of all four flippers and their compliance (which is obtained by limiting maximal allowed current to their corresponding servomotors) is generally a difficult 8-dimensional continuous problem. However, our experience suggest that only several configurations may be enough to succesfully drive robot across rough terrain. It is not necessary to adjust posture of flippers with millimeter precision so that they are accuratelly copying terrain under the robot. The robot just needs to know only few different flipper modes—each one for specific terrain structures that may be encountered, see Fig. 6.2. [1] [2]

This observation allows us to simplify the task. It may be limited just to a choice of a viable flipper configuration from a finite discrete set. Flipper configurations proposed in this thesis were designed during experiments at FEE, CTU. [1] [2] However, we are not the only ones who do something like that—Colas et al. proposed similar configurations in design of their path planner. (Their approach is also described in chapter 5.2.3) [22]

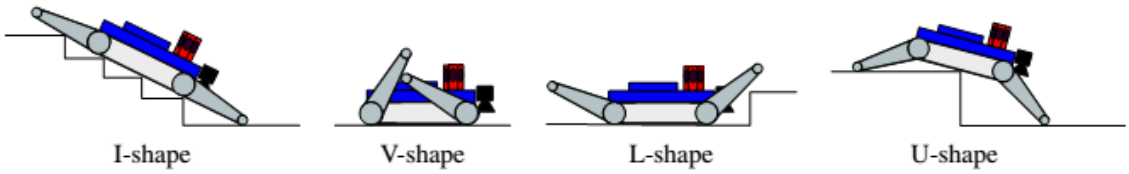

Fig. 6.2: Proposed flipper configurations. [1]
Each configuration corresponds to a different morfology of robot´s body and has its own characteristics. The configurations are depicted in typical situations for their usage.

I-shape (maximazes traction)

The flippers are parallel with the main tracks and they increase effective lenght of tracks. It consequently increases traction, which is the highest out of all configurations. I-shape mode is effective when driving robot on inclined plane and particularly when it has to climb up or climb down a staircase (and it is already past first step), because tracks make contact with as many steps´ edges as possible. On the other hand, the robot may have difficulties to get over some bumps or single steps in this mode.

V-shape (provides observability)

The flippers are folded. It is used when robot is moving on a flat surface, i.e. in situations when flippers are not actually needed for overcoming obstacles and traction provided by main tracks is sufficent for robot to move. Folding the flippers guarantees free view for laser scanner or cameras mounted on the robot. (The flippers otherwise slightly cover the view of sensors to some extent when being in other configurations, thus limiting information about surrounding environment which robot is able to obtain by sensors.)

L-shape (climbing up)

The front flippers are elevated. It is the best mode, when robot is approaching an obstacle (like stones, bumps, etc.) as it allows the robot to smoothly get on the obstacle.
As posture of tracks in this mode is similar to a tank (or other military vehicles), it is the most powerful configuration in the means of terrain traversability. Unfortunately elevated flippers interfere with rays of laser scanner, thus complicating building of model of environment. Therefore it is prefered to use this configuration only when it is really needed.

U-shape (climbing down)

The flippers are pushed downwards when this configuration is on. It the most suitable mode when robot is approaching some kind of depression or a step down. The front flippers are the first part of robot´s body to touch ground under the step and they are immidiately able to provide enough support for robot to prevent falling down or flipping over.

The idea is to change between configurations so that the movement of robot is as fluent as possible and eventual obstacles are safely traversed. Some obstacles may even require a combination of configurations which are being adaptively changed as robot moves across the obstacle. For example, when climbing a staircase, the robot must first use L-shape to get on first step. As robot´s pitch rises, configuration is changed to I-shape maximizing traction when climbing following steps. Finally, when robot gets on the upper edge of the staircase, U-shape is required in order to prevent flipping over.

6.1.3 Reinforcement learning framework for AT

The agent which is selecting optimal flipper configuration is based on reinforcement learning framework which has been theoretically described in chapter 4. The goal is to choose configuration $c \in C = \{'I', 'V', 'L', 'U'\}$ from a set presented in the previous section. The agent is evaluated by a reward function $r(\boldsymbol{F}, c, \boldsymbol{F}') : \mathbb{R}^n \times C \times \mathbb{R}^n \rightarrow \mathbb{R}$ which assigns real-valued reward for a transition from an initial state described by features $\boldsymbol{F}$ to a successor state described by features $\boldsymbol{F}'$ while using configuration c. [1]

We may define Q-function $Q_{AT}(\boldsymbol{F}, c) : (\mathbb{R}^n \times C) \rightarrow \mathbb{R}$ which estimates expected sum of discounted rewards following after the robot drives from state $\boldsymbol{F}$ using configuration c. The aim of learning is to estimate Q-function so that for each state $\boldsymbol{F}$ we may select configuration c^* with highest Q_{AT}-value: [1]

$$c^* = \arg\max_{c \in C} Q_{AT}(\boldsymbol{F}, c) \quad . \tag{6.1}$$

We also require that the best configuration c^* fulfils a condition that $Q_{AT}(\boldsymbol{F}, c^*)$ should be at least positive. Otherwise, none of configurations may be considered safe and it is not recommended to drive robot forward at all.

There were several problems which were encountered during collection of training samples for initial training of Q_{AT}-function. Firstly, driving a real robot is generally time consuming and any operator is able to record only a limited set of trajectories in reasonable time. Secondly, the operator cannot afford to jeopardize robot. (So far, the robot is too expensive to risk its destruction only to obtain some data samples.) That means, that there are not many examples from situations in which robot robot found itself in some critical circumstances.

The first of the problems mentioned in previous paragraph was partially bypassed by manual annotation of the data. While the configuration chosen by the operator when actually driving robot was denoted as optimal and was assigned a reward equal to 1.0, the other configurations were judged offline on the recorded data. Suboptimal configuration for each situation received reward 0.5, indifferent configuration (neither good nor bad) received reward 0.0 and unacceptable configuration received negative reward (i.e. penalty) – 1.0.

Thanks to that, we got useful training data even for configurations which were not used for driving robot at all, thus making our training dataset richer. Manual annotation also enabled us to partially bypass the second problem and add negative samples to dataset. As it was mentioned, some configurations may have received negative rewards in this way. Additionally, even if robot could not have been taken into situations which would endanger it, the operator could drive it near to such situations. For example, the operator could have taken robot near the edge of a cliff. Naturally, such recording episode had to end before the robot reached the edge and fell down. But last frames of recorded data may have been labeled with negative rewards as any of the configurations would lead to destruction of the robot if it really continued moving forward.

However this approach has its drawbacks. Mainly, as it is not much straightforward, such data collection and manual labeling may be hardly replicable. Secondly, when rewards were manually assigned to a configuration different from configuration actually used when driving robot during data recording, a fact that robot may find itself in a slightly different state was neglected. For example we did not take into account that robot would be in a different pitch angle if it switched flippers´ positions.

6.1.4 Decision trees

Q_{AT}-value $Q_{AT}(\boldsymbol{F}, c)$ for each pair feature vector $\boldsymbol{F}$ - configuration c is predicted by regression trees which are trained using recorded training datasets.

We operate with n-dimensional vectors of features $\boldsymbol{F} = (F_1 \dots F_n)$ Conditional probability $p(F_i|F_j)$, where $j = \{1 \dots n\} \setminus i$ of i-th feature is represented by a forrest of regression trees. We have a training dataset which consists of M training samples denoted by $\boldsymbol{F^1} \dots \boldsymbol{F^M}$. Each tree is learned by top-down greedy algorithm which selects the splitting variable j and split point s at each node in order to minimize variance in the left and in the right subtree. Following term is minized [2]:

$$\underset{(j,s)}{\operatorname{argmin}} \; |R_1| \cdot \underset{k \in R_1(j,s)}{\operatorname{var}}(\boldsymbol{F_i^k}) + |R_2| \cdot \underset{k \in R_2(j,s)}{\operatorname{var}}(\boldsymbol{F_i^k}) \quad , \tag{6.2}$$

where $R_1(j,s)$ denotes a subset of samples which fall into left subtree (value of splitting variable j is lesser than or eaqual to splitting value s) and $R_2(j,s)$ denotes the other subset

of samples which fall into right subtree (value of splitting variable j is greater than splitting value s). Those samples whose particular splitting feature is unknown are inserted to the both subsets, thus falling to the both subtrees. [2]

Conditional probability of the feature is computed as the mean conditional probability over all the leaves in the forrest reached by the respective sample. It is finally used to estimate probability distribution function of the Q_{AT}-function. $Q_{AT}(F, c)$ is obtained as mean value over this distribution. [2]

6.2 Enhancement of AT algorithm

6.2.1 High–level features F_H

Generally, one of characteristic properties of decision trees is, that learning algorithms have already embedded feature selection. That also includes trees used in this work, see previous section. The learning algorithm is searching feature vector for features which are able to effectively split learning dataset. In the end, it may happen that only a subset of features from the feature vector is actually used.

It was observed when experimenting on recorded datasets, that if all values from DEM are used as single input features, only few of them are actually used for making decisions while the others are ignored. Such behaviour may increase probability of overfitting. Therefore usage of high-level features was considered in order to make model more robust. Vector of high-level features will be furtherly denoted as $\boldsymbol{F_H}$.

Several combinations of high-level features were designed—five different models with different vectors $\boldsymbol{F_H}$ were constructed, see Appendix for detailed description. Q_{AT}-function $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}^{i}, c)$ has been learned for each of them separately, thus we obtained 5 different models for adaptive traversability. For example, model which is denoted as Model 4 has following construction of vector $\boldsymbol{F_H}$:

$\boldsymbol{F}_{\boldsymbol{H}}^{4}$ = [pitch and roll,
average height values in lines in DEM (there has to be at least 2 valid values in line),
number of NaN values in lines in DEM (line is perpendicular to x-axis),
standard deviation of averages in lines in DEM,
maximal gradient value along x-axis (same orientation as heading of robot) of DEM,
minimal gradient value along x-axis of DEM,
mean gradient value between lines in DEM]

Proposed high-level features particularly describe actual robot´s inclination and provide information about terrain in a compact and robust form. For example, using average height value in whole line of DEM should be less susceptible to noise than using value from a single bin of DEM.

6.2.2 Active tactile exploration of terrain by robotic arm

Kinova JACO robotic arm has been mounted on the top of Niftibot. Vojtěch Šalanský implemented an algorithm which enables the robot to use the arm for active tactile exploration within his thesis. [36] The basic principle is simple and is similar to the method used by blind people when they are navigating themselves. The arm is holding a stick which is used to tap on terrain. After coordinates $[x, y]$ of unknown part of terrain (for example coordinates of DEM cell) are specified, the arm moves the stick towards ground at the requested coordinates until it touches the ground. The touch is recognized when the currents in arm´s servomotors rise due to pushing against the ground. Height value z which is computed by solving direct kinematic task for the arm is then returned. [36]

This work is focused on utilization of arm in order to fill in missing height infromation in DEM. It consequently helps to determine if it is safe to move robot forward and what flipper configuration is optimal for such transition. The arm is able to reach up to cca. 50 cm in front of the robot. DEM cells which may be measured by arm are colored light green in Fig. 6.1. This particularly involves cells from lines 11–14. Despite arm´s reach is limited, it particularly enables us to explore terrain immediately in front of the robot and it may provide valuable information for making decisions.

6.2.3 Introduction of NaN Mode

The decision trees, see paragraph 6.1.4, are used to predict values of Q_{AT}-function $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, c)$ for all configurations $c \in C = \{'I','V','L','U'\}$ at the state described by feature vector $\boldsymbol{F_H}$. When performing experiments on the data where some of information about terrain was missing, a problem was encountered.

There are two common types of situations when a configuration (or all configurations) may receive a bad reward:

1) It is actually inappropriate to drive robot further in that configuraton.
2) There is too much incomplete information in DEM so that driving robot further may

not be denoted as safe in any configuration.

Therefore a new mode (a pseudo-configuration) was introduced for situations when there is too much missing information in DEM. It is reffered as NaN (not a number) mode, because this value is prevalent in feature vector $\boldsymbol{F}$ (which includes DEM) in that case. The extended set of modes, which may be chosen, is now $C' = C \cup \{'NaN'\} = \{'I','V','L','U','NaN'\}$.

The both situations presented above share that it is not safe to move forward. However, the later one may possibly obtain a better reward after missing information is completed. It was not needed to distinguish these two kinds of situations in time when robot was not equiped by the robotic arm for active tactile exploration, because it did not have any mechanism to fill in the missing information. However, now, when we have the opportunity to explore terrain by the arm, it is practical to be able to decide if further exploration will be beneficial or redundant; especially if we take into account the fact that exploration by the arm may be time demanding.

A new tree has been trained (and added to existing model) to predict $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, 'NaN')$ using a dataset with manually annotated training samples. The rewards for these data were labeled by values from range from -0.5 (meaning no need of tactile exploration) to 1.0 (almost all data in DEM are missing). So, the training procedure was similar to other configurations.

However, this mode is a bit specific in contrast with other modes and is treated differently. Firstly, it does not move flippers, but exploration by the robotic arm is initiated instead. Secondly, the rule (6.1) that configuration with highest $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, c)$ is the one that will be used does not explicitly apply on NaN mode. **It was empirically found out that it is more convenient to use value $\boldsymbol{Q_{AT}(F_H,'NaN')}$ as a safety measure in general.** When this value is higher than a specified treshold ε, i.e.

$$Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, 'NaN') > \epsilon \quad , \tag{6.3}$$

movement forward is not considered safe due to a lack of information in DEM in front of the robot. NaN mode is used—and therefore exploration by arm is enforced—whenever $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, 'NaN')$ is higher than ε (value $\varepsilon = 0.25$ is used implicitly) regardless other configurations (even if there is some configuration with higher Q_{AT}-value). The other

configurations are judged only after $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, 'NaN')$ becomes low, which means that the terrain in front of the robot is known well.

The new rule applied to a choice of viable flipper configuration $c*$ may be formulated in the following way:

$$c* = \arg\max_{c \in C} Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, c) \text{ subject to } Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, 'NaN') < \epsilon \qquad (6.4)$$

and $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, c*)$ must be at least positive.

Introduction of NaN mode also solves another problem which was occasionally observed. The implemented decision trees have a property that if value of the splitting feature is unknown, then input sample descents into both sub-trees. The output value is then computed as mean over all leaves, in which the input sample ended.

Initially it was expected, that predictor will predict poor values of $Q_{AT}(\boldsymbol{F}, c)$ for inputs $\boldsymbol{F}$ with too much incomplete data. However, it was observed that it may occasionally happen that it predicts a good value of $Q_{AT}(\boldsymbol{F}, c)$ because the few elements which are actually known may happen to be the ones which ultimately fall to branches of the tree giving high rewards.

Now, when there is too much unknown information about terrain, NaN mode is chosen even if some other mode may incidentally get a good reward in spite of little information.

However, we suppose that this drawback could be partially solved if there was a richer training dataset to learn the trees.

6.3 Testing of AT algorithm

6.3.1 Offline testing on recorded datasets

Decision trees have been learned to estimate value of Q_{AT}-function $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, c)$ for all modes $c \in C'$. Such trained model then chooses the most viable mode from C' for particular input feature vector $\boldsymbol{F}_{\boldsymbol{H}}$. As it has been mentioned in section 6.2.1, five models —five different Q_{AT}-functions—which differ in the way, how feature vector $\boldsymbol{F}_{\boldsymbol{H}}$ is computed, have been trained.

Performances of these five models were assessed offline on training dataset. As it is mentioned in 6.2.3, samples in this dataset had been manually annotated, telling what

configuration was optimal, what configuration was suboptimal or what configuration was inappropriate. The configuration chosen by the model was compared to the manual annotation in order to determine if the model made optimal, suboptimal or incorrect decision. The results are shown in graph, see Fig. 6.3.

In addition, second dataset was prepared for testing. This dataset contained some unique samples which did not appear in training data. It also contained some samples from training dataset in which values in DEM were synthetically pertubated. Height values in DEM cells were altered by a value from normal distribution with mean value 0 cm and standard deviation 4 cm. It was done to obtain richer dataset for testing. Again, the configuration chosen by the model was compared to manual annotation in order to determine if the model made optimal, suboptimal or incorrect decision. The results are shown in graph, see Fig. 6.3.

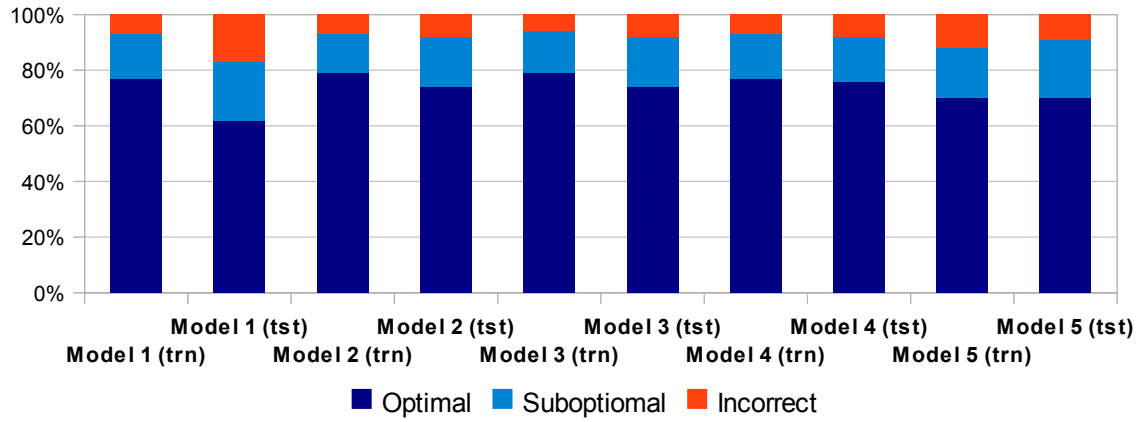

Fig. 6.3: Testing of models for adaptive traversability.
Models were tested on two datasets—training dataset (trn) and different testing dataset (tst). The graph shows percentage of optimal, suboptimal and incorrect decisions made by models on samples in particular datasets.

As you may see in Fig. 6.3, all five models provide similar results. The optimal action was chosen on approximately 75% of data samples, suboptimal action was chosen approximately on 15% of data samples and incorrect action was chosen on approximately 10% of data samples.

10% of incorrect decisions may seem quite high, but when the robot is actually driven, decision about configuration is done with frequency cca. 2 Hz. Hence incorrect decision is often corrected in less than a second. Fig 6.4 shows how many times (in %) the models

chose incorrect decision on three samples from testing dataset in a row. (Note that dataset contained samples in the order as they were recorded.) Repeated incorrect decision, which may actually endanger the robot, is made in less than 2.5%.

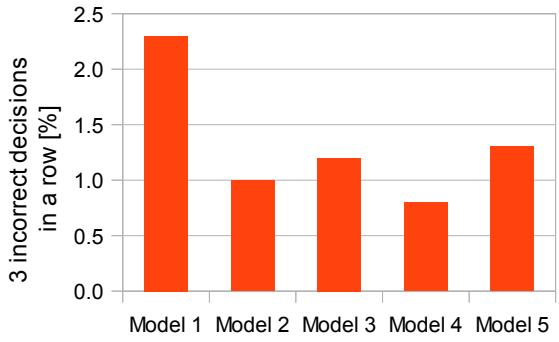

Fig. 6.4: Testing of models for adaptive traversability.
Percentage of cases, when model chose an incorrect mode 3 times in a row on samples from testing dataset.

Probably the worst performance was given by Model 1. There is an apparent drop in succes rate when comparing performance on training and testing data, thus suggesting that this model was slightly overfitted. The other four models behave more robustly and results of tests performed on training and testing data are similar.

The best model created so far is the one denoted as **Model 4**. It made only 7.8 % incorrect decisions on testing dataset and only in 0.8 % cases incorrect decision was repeated 3 times in a row or more). Therefore, this model is prefered when driving the robot.

6.3.2 Online testing of AT algorithm on Niftibot

The test of performance of the proposed adaptive traversability algorithm on Niftibot was performed when Nifitbot was driven across standard EU pallet. You may watch a video record AT_1.mp4 on enclosed DVD. The robot behaved as it was expected. It used L-shape when approaching the pallet in order to get onto it. When robot´s pitch rose, configuration was switched to U-shape to support the robot and thus prevent flipping over. While being on pallet, V-shape could be used for observability. Finally U-shape was used to climb down when the robot reached the opposite edge of the pallet.

7. Tactile exploration of DEM

Special mode for situations when there is missing information in digital elevation map (DEM) was introduced in the previous chapter. Decison tree which is predicting Q_{AT}-value $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ has been trained on manually annotated data. This Q_{AT}-value serves as a safety measure. The idea is that whenever Q_{AT}-value of NaN mode which depends on feature vector $\boldsymbol{F_H}$ (whose significant part is computed from DEM) is less than specified treshold ε ($\varepsilon = 0.25$ in implicit setting), robot´s movement forward is not considered safe because terrain in front of the robot is unknown and generally may have any shape. The missing part of map may mean that there is just a shallow puddle of water or there may be a deep pitfall. These two situations cannot be distinguished only using laser scans, but their distinction is crucial for robot safety.

Hence, tactile exploration of DEM by robotic arm is invoked. In the first case (a puddle), missing height values are being gradually filled in DEM as the arm touches terrain at various points. Finally, there will be enough known values in DEM that robot may determine that it is safe to continue and chooses the best flipper configuration to go across the puddle. In the second case, it may even happen that the pitfall is too deep that robotic arm is unable to reach its floor. No information is added to DEM and therefore we may conclude that going on will be dangerous and rather choose different direction of heading instead of going forward.

For illustration, an example of situation when exploration by arm is needed is depicted in Fig 7.1. $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ is high as DEM in front of the robot is practically empty and agent does not have enough information to decide whether to continue going forward and what flipper configuration should be used. Updated state after exploration, during which 10 DEM cells were measured by the arm, is depicted in Fig 7.2. At this time, decision about flipper configuration can be safely done, because value $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ has fallen under treshold $\varepsilon = 0.25$. (It became even negative.) As you may notice, V-shape flipper configuration obtained the highest Q_{AT}-value and therefore is chosen for moving forward.

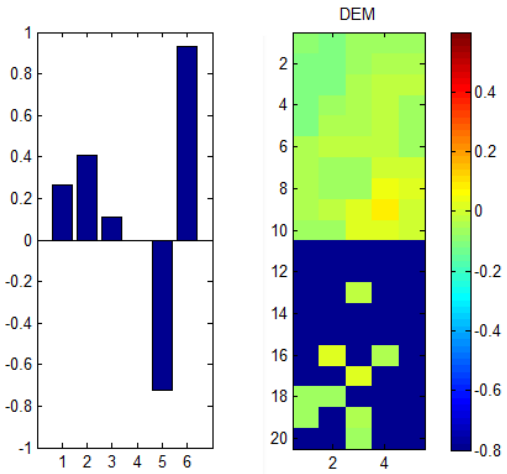

Fig. 7.1: Example of DEM in situation when major part of terrain is not visible for robot. Left: Q_{AT}-values for all modes [1-I, 2-V, 3-L, 5-U, 6-NaN]. NaN mode has Q_{AT}-value greater than 0.25 → exploration by arm is required.
(Note: Model 4 is used to estimate Q_{AT}-value, see chapter 6 for more information)
Right: DEM. Heights in cells (in meters) are coded by colors, dark blue ~ NaN. The robot finds itself in the upper half of DEM and it is heading down.

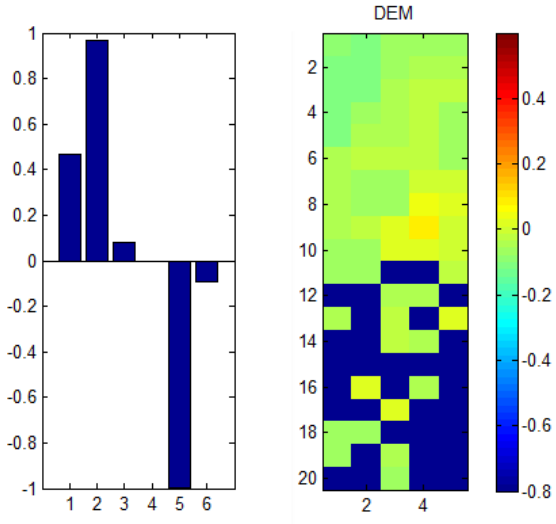

Fig. 7.2: Example of DEM after some of missing values were measured by arm.
Left: Q_{AT}-values for all modes [1-I, 2-V, 3-L, 5-U, 6-NaN]. NaN mode has Q_{AT}-values less than 0.25 → exploration by arm is no longer required. V-configuration has the greatest Q_{AT}-values → it is chosen to traverse terrain.
Right: DEM. Heights in cells (in meters) are coded by colors, dark blue ~ NaN. The robot finds itself in the upper half of DEM and it is heading down.

The presented concept is simple—the robot is supposed to use arm for exploration of terrain in front of itself until safe decision can be made, i.e. Q_{AT}-value of NaN mode becomes low. Naturally, it is advantageous to use arm as little as possible. In other words it is desired to find a strategy which would enable us to touch minimal number of cells in DEM that would be sufficent to make safe decision.

One reason for this solicitude is to minimize time spent on touching terrain. As we want to prevent any damage of arm which could be caused by rash movement, it is preffered to control arm so that it moves rather slowly. Therefore even measurement at only one cell of DEM takes some time. Second reason is that our arm is not equiped by actual tactile sensor. Contact with the ground has to be recognized indirectly from currents in arm´s servomotors when arm starts to push against ground. This contact recognition method burdens motors in joints and some joints may get overheated after some time. So, it is desired to touch ground at as few points as possible.

7.1 Reinforcement learning for tactile exploration

7.1.1 RL Framework

When looking for an optimal exploration strategy which would be quick and reliable, reinforcement learning approach was utilized one more time. For detailed information about reinforcement learning see chapter 4.

The two elements of framework which have to be properly defined are states and actions. As arm is used to explore DEM, it is natural to derive states and actions from DEM. Action is easier to define—action a is related to cell whose height is supposed to be measured. Therefore we choose action $a \in A$, where A is a set of all cells in DEM reachable by arm, see Fig. 7.3.

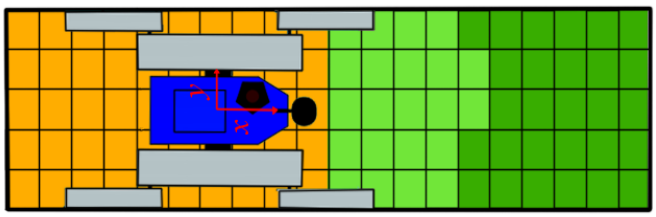

Fig. 7.3: Digital Elevation Map (DEM).

DEM is composed of 0.1 m x 0.1 m cells. Each cell stores value of height of terrain in particular place. Cells which may be reached by arm are colored light-green.

States are a bit more difficult to define. The very first idea was to denote what cells are already known and what cells are left to explore. But, as one cell may be in 2 states (known/unknown), it makes 2^{22} states (over 4 million states) in total if we take into account only those cells which may be touched. It would be considerably computationaly demanding to estimate Q-values for all states and it would require a lot of memory as well. Hence, it is good to replace such high-dimensional states with low-dimensional ones. An approach which showed up to be efficient is to watch how many cells are known in each line of DEM. One line may then find itself in 6 possible states (none cell is known, ..., all five cells are known). If we take into account lines 11-15, there are 6^5 (7776) states. This number of states is more acceptable. (Note: Keeping track, how many cells in particular lines of DEM are explored, is performed also because input high-level features $\boldsymbol{F_H}$ for adaptive traversability predictions $Q_{AT}(\boldsymbol{F_H}, c)$ are based on average values in lines of DEM, see section 7.3.1.)

7.1.2 Learning process

The task of exploring DEM naturally breaks down into episodes. We start with unknown terrain and then, step by step, we fill new measured values into DEM (and consequently feature vector $\boldsymbol{F_H}$, which serves as the input for decision about safety, is updated) until we have enough information about the terrain. If we want to make this process fast, we should adapt rewards to motivate learning agent to finish as soon as possible. We may define reward in a following way: We give a small negative reward $r(s, a, s') = -1$ if $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ in new state s' is still higher than the specified treshold $\varepsilon = 0.25$. Otherwise, we give high positive reward $r(s, a, s') = 25$ when $Q_{AT}(F_H, 'NaN')$ becomes less than ε and the episode is terminated. The ultimate goal of learning agent is to learn how to maximize obtained reward so it is supposed to get to terminal state quickly.

We may define a new Q-function $Q_{EX}(s, a)$ which estimates future expected sum of rewards starting from point when action a is used in state s. (Lower index EX is used to distinguish this function from Q_{AT}-function $Q_{AT}(F_H, c)$ used for adaptive traversability.) On-policy Monte Carlo Control learning algorithm adopted from book [13] was implemented to train $Q_{EX}(s, a)$, see Chapter 4.3 for detailed description.

The whole learning process to obtain $Q_{EX}(s, a)$ was performed offline on datasets recorded by the robot in advance. (Learning online while the robot is moving would be time

consuming and it could also endanger the robot.) The training dataset consisted of DEMs with full information. However, before each learning episode started, front part of DEM was artificially hidden, i.e. height values in cells were replaced with NaN values. Learning agent was then performing simulated "measurements of height" after which NaN values were, one by one, replaced back by actual height value until terminating condition was fulfilled. Over 2.5 million of learning episodes were performed in order to ensure that each state-action pair was tried enough times to ensure that estimated $Q_{EX}(s, a)$ is plausible.

7.2 Testing of strategy learned by Q–learning

Q_{EX}-values $Q_{EX}(s, a)$ for all possible state-action pairs were determined by reinforcement learning. The strategy for exploration is then to greedily choose action a^* at given state s:

$$a^*(s) = \arg\max_{a} Q_{EX}(s, a) \quad . \tag{7.1}$$

Performance of this strategy has been assessed offline on testing dataset. At first, front part of DEM was replaced with NaN values in the beginning of each testing episode, thus simulating terrain invisible to laser scanner. Each action *a*(s)* simulated a measurement by arm by replacing NaN value back with actual height value. Testing episode ended in the moment when $Q_{AT}(F_H, 'NaN')$ dropped bellow treshold $\varepsilon = 0.25$ and number of measurements which was needed to reach this state was remembered. This simulated experiment was performed on 1150 DEMs.

Similarly, a strategy which was choosing random actions was used on the same data (1150 DEMs and 50 random episodes on each of them), so that performance of both strategies, measured by number of actions needed to make a decision, may be compared.

Chapter 6.3.1 was dealing with high level features $\boldsymbol{F_H}$. Choice of high level features directly influences adaptive traversability Q_{AT}-function $Q_{AT}(\boldsymbol{F_H}, c)$ and hence differently build vectors $\boldsymbol{F_H}$ make different models. Five such models were designed (see their detailed specification in Appendix) and their correspondive functions $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}^{i}, c)$ were learned separately. This also means that $Q_{EX}(s, a)$ strategy had to be learned for each model

separately as well, because $Q_{EX}(s,a)$ model is build on $Q_{AT}(\boldsymbol{F_H}, c)$ model. However, learning algorithms were used always the same.

The performance of $Q_{EX}(s,a)$ strategy then could have been compared with random strategy on each model. Average numbers of measurements needed to make a decision about robot´s safety when using random measurements (actions) and Q-learning strategy are plotted in a graph, see Fig. 7.4.

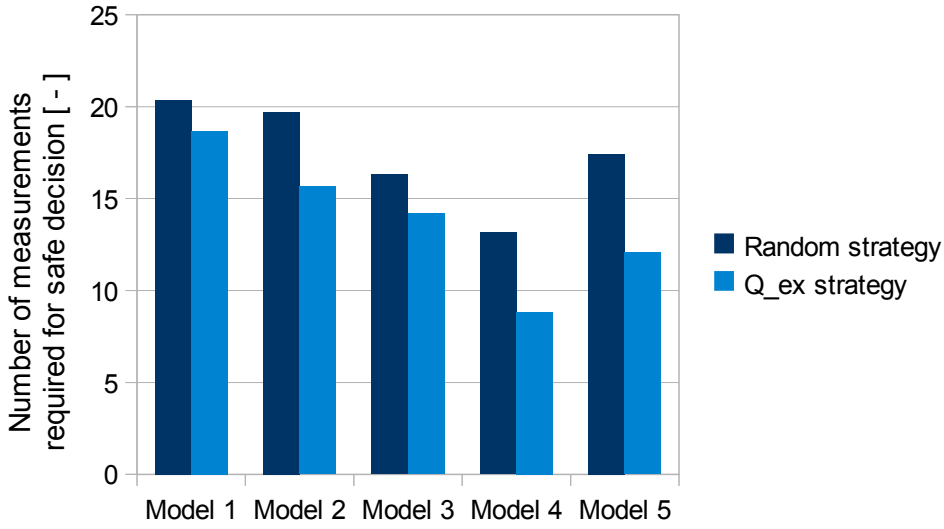

Fig. 7.4: Comparison of strategy learned by Q_{EX}-learning and random strategy.
Q-learning algorithm was used to learn a strategy on 5 different models (which differ in the way how vector $\boldsymbol{F_H}$ is constructed, see chapter 6.3.1). Figure shows average number of DEM cells which have to be measured by robotic arm until $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ becomes lower than $\varepsilon = 0.25$ by random sampling and Q_{EX}-learning based strategy.

Our results propose that using Q-learning strategy for exploration may speed up the process. It proved to be faster on all of five presented models. We observed that this strategy prefers to choose action which may potentially bring more valuable knowledge about terrain and which may enable robot to make safe decision sooner.

Q-learning strategy performed well particularly on Model 4. Only 8.8 measurements were needed in average and it was less cca. by 4 measurements than using random strategy.

8. Documentation

This thesis was mainly focused on a continuation in a development of a ROS package *'adaptive_traversability'* for UGV in TRADR project. This package´s purpose is to introduce a module for autonomous control of Niftibot´s flippers and to adapt them with respect to traversed terrain. Core of the package was implemented by Petr Zuzánek. [1] [2] Source codes of several nodes were modified during the work on this thesis.

Files AT_with_MATLAB_JACO.cpp, AT_with_MATLAB_JACO.h are then the most important contribution of this work (along with Matlab part). These files realize communicaton with Matlab, which was used to create and utilize models for AT, see chapter 6, and for arm exploration, see chapter 7. Matlab part of the project is implemented in files init_nodes.m, F_callback.m and F_choose_arm_action_callback.m.

If you are interested in the implemented '*adaptive_traversability*' package, please contact the author of this thesis, Jakub Mareš, or his supervisor, Karel Zimmermann, Ph.D..

8.1 Launching instructions

After Niftibot is turned on and all drivers are succesfully initialized, run following commands on the robot, the base station and in Matlab:

```
BASE$ roslaunch nifti_mapping_launchers mapAndNav.launch
BASE$ roslaunch adaptive_traversability AT_with_MATLAB_JACO_base.launch
ROBOT$ roslaunch jaco_moveit JACO_for_AT.launch
ROBOT$ export ROS_IP=192.168.2.xxx # ip adress of robot
ROBOT$ roslaunch adaptive_traversability AT_with_MATLAB_JACO.launch
MATLAB$ cd "path to 'at_rosmatlab' directory"
MATLAB$ init_nodes
```

To ensure proper communication between Matlab and roscore which is running on robot, modify following lines in init_nodes.m properly:

```
ROS_MASTER_IP = '192.168.2.xxx';    % ip adress of robot
ROS_MASTER_PORT = 11311;            % port
setenv('ROS_MASTER_URI','192.168.2.xxx:11311') % ip adress of robot:port
setenv('ROS_IP','192.168.2.yyy')    % ip adress of machine with Matlab
```

Press buttons '3' and '4' on joypad simultaneously to activate adaptive traversability mode. Press button '4' to deactivate.

Press button '5' to enable exploration by JACO robotic arm. Press button '6' to disable.

Brief description of launched nodes is covered in the following sections.

8.2 Base station nodes

Following launchers are primarly supposed to run on the base station, but they may be also run on the robot as well.

8.2.1 mapAndNav.launch

This launcher starts nodes from '*nifti_mapping*' package which are responsible for running laser scanner and generating useful information about geometry of robot´s surrounding environment.

8.2.2 AT_with_MATLAB_JACO_base.launch

This launcher starts several nodes which are used particularly for preparing map information which is later used by AT_MATLAB_JACO node to generate DEM and feature vector ***F***.

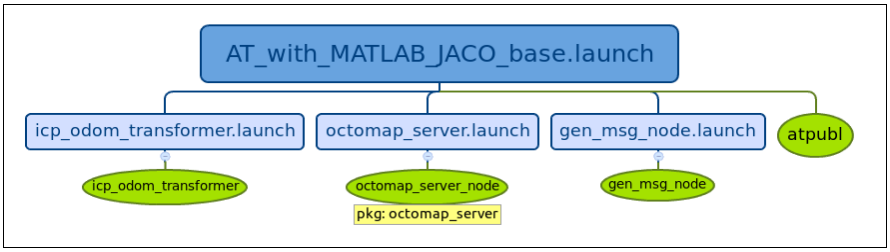

Fig. 8.1: AT_with_MATLAB_JACO_base launcher

Node: icp_odom_transformer

Source: adaptive_traversability/src/icp_odom_transformer.cpp

This node listens to /dynamic_point_cloud topic. It receives point cloud containing points whose coordinates are expressed in /base_link frame, i.e. it receives points generated by laser scanner in coordinate system connected to the robot. The node then transforms point cloud into fixed /map frame which is the global frame. Such transformed point cloud is published on topic /dynamic_point_cloud_filtered which is subscribed by octomap server.

Node: octomap_server_node

Package: octomap_server [37]

It is a node from octomap_server package which is one of ROS standard packages. It listens to the topic /dynamic_point_cloud_filtered where dynamic point clouds in global frame are published. These are gradually used for building an octomap, which is 3D occupancy grid. Octomap is published on /octomap_point_cloud_centers topic.

Node: gen_msg_node

Source: adaptive_traversability/src/gen_msg_node3.cpp

This node´s function is to subscribe to several topics in order to obtain information about robot´s position, velocity, odometry, currents in tracks, state of flippers and particularly point clouds. It is receiving dynamic point cloud (on topic /dynamic_point_cloud) and point cloud generated by octomap server (on topic /octomap_point_cloud_centers).

The node wraps up all information into floor message (floor_msgs.msg), which is defined within adaptive_traversability package. This message contains all the information needed to construct feature vector ***F***.

Node: at_publ

Source: adaptive_traversability/src/tf_child_publisher.cpp

This node is broadcasting 'tf' transformation between coordinate frames /base_link, which is connected to the robot, and /stab_base_link, which has the same origin, but it has compensated pitch and roll so that it is parallel to global frame.

8.3 Robot nodes

Following nodes (launchers) are intended to be run on the robot itself.

8.3.1 JACO_for_AT.launch

There is a nested launcher jabbing.launch which was implemented by V. Šalanský. [36] It starts action server for jaco_moveit, thus it enables to control arm from ROS and it launches jab_service.py node within which there is an implemented algorithm for touching ground at specified point.

Node: service_arm_touchDEM

Source: jaco_moveit/src/service_arm_touchDEM.py

This node utilizes jab_service.py to perform height measurement at specified DEM cell. It serves as a service server which uses user defined srv **jaco_moveit/JabDEM.srv**:

```
int8 row
int8 col
---
bool valid
float32 x
float32 y
float32 z
```

The request contains coordinates (row and column) of DEM cell which should be explored by the arm. The arm performs a measurement after which the service returns a response containing coordinates [*x, y, z*] of the measured point. Boolean variable *valid* denotes if measurement was succesful (therefore values in *x, y, z* are valid) or unsuccesful.

The service´s name used in ROS framework is **/JACO_touch_DEM**.

8.3.2 AT_with_MATLAB_JACO.launch

This launcher starts a node AT_MATLAB_JACO which is the core node for adaptive traversability. It communicates with Matlab node and is capable to utilize JACO robotic arm through service_arm_touchDEM.py node (by calling /JACO_touch_DEM service).

Node: AT_MATLAB_JACO

Source: adaptive_traversability/src/AT_with_MATLAB_JACO.cpp

Header: adaptive_traversability/include/AT_with_MATLAB_JACO.h

This node is the main node for adaptive traversability on Niftibot. It is subscribing for floor message and dynamic point cloud from gen_msg_node. When callback on floor message is triggered, inference process begins.

In the beggining, vector $\boldsymbol{F}$ is constructed. It contains 210 members: $\boldsymbol{F}$ = [100 x height values for each bin in DEM; pitch; speed_r (real speed of robot); speed_c (speed demanded by control); 6 x currents in tracks; roll; 100 x counter].

This vector is send to Matlab node on topic /model_request. Matlab node, at first, extracts high-level features $\boldsymbol{F_H}$ from $\boldsymbol{F}$. After that, value $Q_{AT}(\boldsymbol{F_H}, c)$ is estimated by regression trees for all configurations and send back to AT_MATLAB_JACO node on topic /model_response.

If $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ is higher than treshold $\varepsilon = 0.25$, vector $\boldsymbol{F}$ is send to Matlab again, this time on topic /arm_action_request. Matlab chooses the most viable action according to $Q_{EX}(s, a)$ which has been trained, see chapter 7 for more details, and is stored in a file RL_OUTPUT.mat. Coordinates (row and column) of DEM cell which should be explored by arm are send back on topic /arm_action_response. After that /JACO_touchDEM service, see section 8.3.1, is called. Robotic arm is used to measure height of terrain at particular DEM cell which is then added to DEM. This process is repeated in a cycle until $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ drops below treshold ε or there are no more DEM cells to explore or operator forces its end.

Finally the most suitable flipper configuration with the highest $Q_{AT}(\boldsymbol{F_H}, c)$ value is chosen for driving robot forwards. The whole process is recapitulated in a flow diagram, see Fig. 8.2.

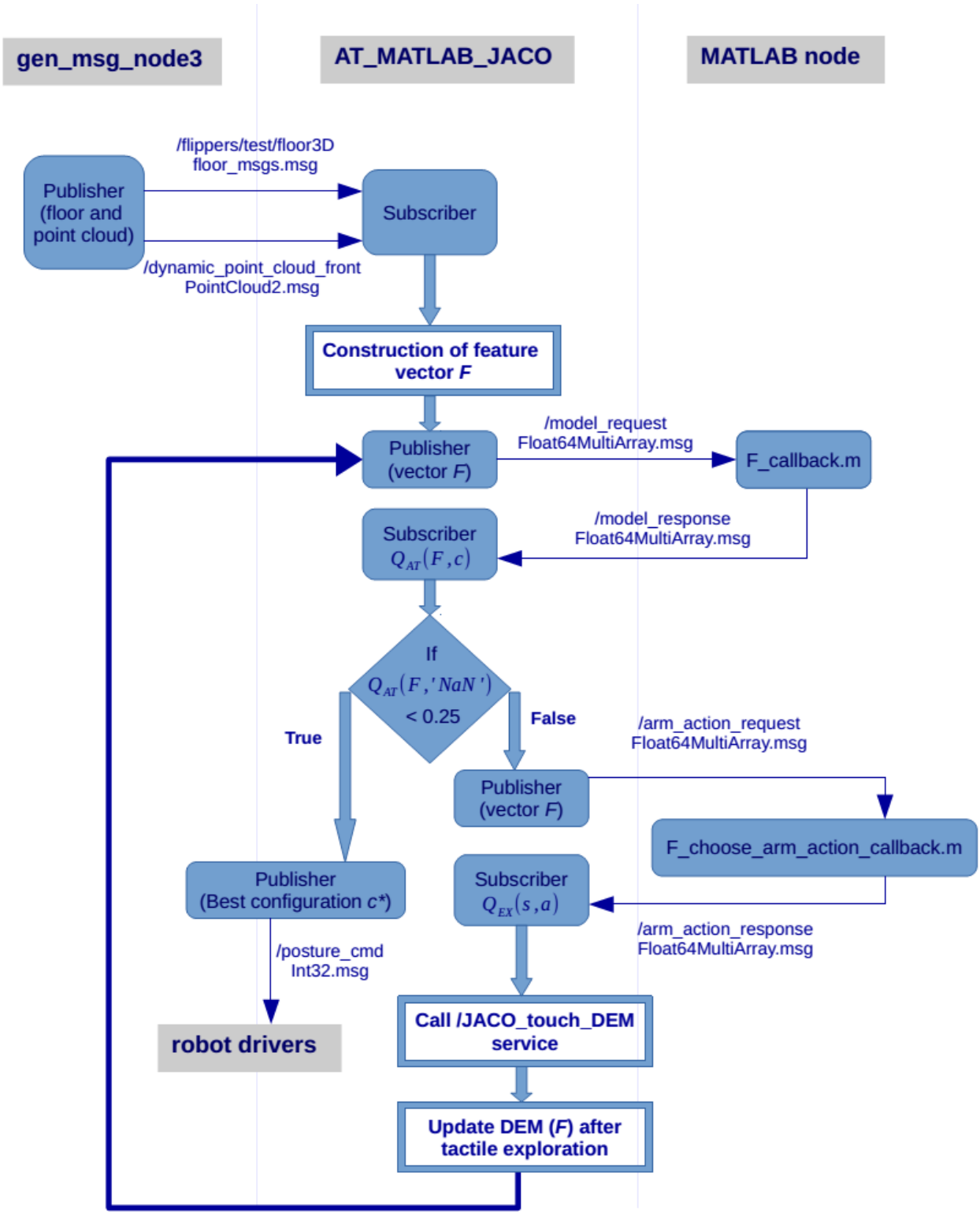

Fig. 8.2: Flow diagram of AT_MATLAB_JACO node.

8.4 Matlab node

8.4.1 init_nodes.m

This script connects Matlab to roscore running on the robot. Remember, that IP adress has to be correctly specified within the code for connection to be succesfully established, see section 8.1 for instructions! Then, it starts a rosmatlab node with two pairs subscriber-publisher, which are covered in the following sections.

8.4.2 F_callback.m

F_subscriber which is started by init_nodes.m subscribes to topic /model_request. A message containing vector $\boldsymbol{F}$ published by AT_MATLAB_JACO node on the topic is received this way. Callback function F_callback.m is triggered at that moment.

This Matlab function utilizes function getFeaturesFromF.m which extracts high-level features $\boldsymbol{F_H}$ from $\boldsymbol{F}$. Then it uses learned regression trees to estimate $Q_{AT}(\boldsymbol{F_H}, c)$ for all configurations. (The trees are stored in ./tree directory whereas major part was implemented by Petr Zuzánek in [2].) In the end Q_publisher publishes Q-values on topic /model_response.

DEM and Q-values for all configurations are also visualized in Matlab figures.

8.4.3 F_choose_arm_action_callback.m

F_action_subscriber which is started by init_nodes.m subscribes to topic /arm_action_-request. A message containing vector $\boldsymbol{F}$ published by AT_MATLAB_JACO node on the topic is received this way. Callback function F_choose_arm_action_callback.m is triggered at that moment.

It utilizes learned $Q_{EX}(s, a)$ function, which is stored in RL_OUTPUT.mat file, in order to determine what DEM cell should be explored by arm. The coordinates (row and column) are published on topic /arm_action_response by arm_action_publisher.

9. Experiments

Two experiments were performed to verify functionality of the implemented algorithm for adaptive traversability extended by possibility of tactile exploration by a robotic arm. The experiments took place under lab conditions. The aluminium foil served as a reflective layer which is hardly visible for laser scanner. It caused missing information in Digital Elevation Map (DEM) in front of the robot and consequently it disallowed robot to move forward because such action would be potentially dangerous. Robot had to use robotic arm which was holding a simple wooden stick to touch some points and add them to the existing map. Only then the robot could make a safe decision about moving forward and choose suitable flipper configuration.

9.1 Experiment on flat terrain

First experiment was captured on a video AT_with_JACO_1.mp4 which is accesible on enclosed DVD.

During the experiment, robot was driven on a flat floor in a laboratory. The operator was only giving commands to move forward or backward. Other actions were performed autonomously by the robot. There was an aluminium foil put on the floor which caused missing data in DEM, because it was not well detactable by laser scanner. When the robot approached the foil, exploration by robotic arm had to be invoked.

The whole movement of the robot may be viewed as divided into few sequences. At first, the robot explores some DEM cells by the JACO arm until movement forward is denoted safe, i.e. Q_{AT}-value of NaN configuration $Q_{AT}(F_H, 'NaN')$ drops under treshold $\varepsilon = 0.25$. Robot can move forward a bit and it gets again into a situation that there is an unknown terrain in front. $Q_{AT}(F_H, 'NaN')$ becomes again higher than ε and hence, exploration by the arm is invoked again and new points are added to DEM.

After several tactile measurements, the robot is able to get safely over aluminium foil. Without usage of the arm for tactile exploration, such obsatcles like the foil could not be traversed because of incomplete terrain data or it would be at least hazardous if operator forced the robot to continue to unknown terrain anyway.

Finally, the operator commanded robot to drive backward to return to the same position as in the beggining of the experiment. As the points which were obtained by touching the ground by the arm were stored in separated point cloud, they persisted in memory. When the same aluminium foil was seen again by the robot, the touched points were already present in DEM. When the robot was overcoming the foil for the second time, no further exploration by the arm was therefore not needed.

9.2 Experiment on a downward step

Second experiment was captured on a video AT_with_JACO_2.mp4 which is accesible on enclosed DVD.

In the beginning of this experiment robot found itself on an elevated platform and operator wanted to drive it downwards. However, an aluminium foil was located under the step and it caused that robot had not enough information about the terrain under the platfrom. Hence, tactile exploration by arm had to be used. When exploration was finished, robot correctly chose U-shape of the flippers which is the most suitable configuration for overcoming a descending step.

10. Discussion

Performed experiments suggest that our approach to adaptive traversability, which is exploiting information about pose of the robot and Digital Elevation Map (DEM) and is using decision trees to predict *Q*-values, is promising. The robot is able to adjust flippers in order to overcome obstacles in front of itself. The main contribution of this work is then extension of the process which is used to construct DEM. Formerly only data from laser scanner were used. It showed up that laser alone may be insufficient for certain scenarios, particularly it is unable to detect reflective surfaces or it performs poorly in the presence of smoke. As the both mentioned situations may be expected during USAR missions, it has been decided to equip the robot by JACO robotic arm.

The idea that the arm may be utilized for tactile exploration of terrain in front of the robot was proved to be practicable. The experiments with an aluminium foil have indicated that the robot is really able to update DEM in this way. This may help make safer decisions in case of incomplete data about terrain in front of the robot.

However, some issues were encountered during tests which may be subject of the future work. One of them is that robot sometimes (in slightly less than 10% cases, see section 6.3.1) chooses incorrect flipper configuration. It is not bad if we take into account that decisions are made with frequency cca. 2 Hz and an incorrect decision is often immediately followed by a correct one, so robot´s safety is not actually endangered. It just lead to a situation when robot starts switching flippers to the incorrect configuration and it is immediately followed by switching back to the optimal configuration before it even finishes switching to the incorrect one. This behaviour may be also seen on the video AT_with_JACO_1.mp4 on enclosed DVD. Addition of some heuristic which would prevent this behaviour may contribute to more fluent movement.

Secondly, movement of the arm is slow, therefore it takes some time to measure even one value in DEM. This setting is chosen deliberately for now in order to prevent rash movement which could lead to bumping with arm to surrounding objects. One also has to be cautious during tactile measurement by arm. It is the principle of this subtask, that one does not know in advance actual height of the terrain which has to be explored. Thus, at first,

arm points the stick at a greater height and it slowly moving down towards ground until a touch is recognized.

Recognition of the touch is also a bit problematic. Contact with the ground has to be identified indirectly from currents in arm´s joints when the arm starts to push against the ground. This contact recognition method burdens motors in joints and some joints may get overheated after some time. The held stick may also "slide" on the surface so that contact is not recognized. Usage of some kind of tactile sensor would be more effective and comfortable.

Thirdly, as the arm has been mounted on the robot, robot´s center of gravity has moved upward. Additionally, as the arm moves, the center of gravity moves as well. It has influence on stability of the robot and it may not be neglected in some critical situations. It is double-edged, because arm may do both—cause tipping over or prevent tipping over—it depends only on the way how it is controlled. Intelligent balancing may be subject of the future work.

Some drawbacks were discussed in previous paragraphs. However, we argue that researched potential of arm´s contribution may compensate second and third mentioned drawbacks as the arm allows the robot to explore more places than before. Tactile exploration by arm may be very useful, particularly if the hand was enhanced with direct tactile sensors which would make recognition of the touch much easier and more reliable.

In parallel with this project, Vojtěch Šalanský was implementing 3D terrain reconstruction algorithm in his diploma thesis. [36] He models conditional probability of height value in each DEM cell with respect to other cells whereas he uses method of maximum likelihood to describe how value in one cell is influenced by values from other cells. DEM is divided into two subsets—known and unkwnown cells. Two properties are used to describe unknown cells (which are candidates for tactile measurement by the arm)—*accuracy* (particular cell has high accuracy if it depends on already known cells) and *usefulness* (particluar cell has high usefulness if it may potentially bring valuable information on other unknown cells, i.e. values in other unknown cells highly depend on this one). The algorithm chooses unknown cell to be measured by the arm, which has high usefulness and low accuracy, so it helps to estimate values in all unknown cells (including itself) more accurately.

Model implemented in this thesis, which is used to choose cell which should be measured by the arm, was trained using reinforcement learning. It chooses such cell which may potentially lower $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, 'NaN')$ which is used as safety measure. (The lower this value is, DEM is better known.) The observed learned strategy is that cells for tactile measurement are chosen in such order to complete information in vector of computed high-level features $\boldsymbol{F}_{\boldsymbol{H}}$. In contrast with the algorithm implemented by V. Šalanský, it does not aim to estimate values in all DEM cells and to model the whole DEM. It just aims to collect enough information to make safe dicision if the robot may or may not move forward and what flipper configuration should be used.

11. Conclusion

This thesis was focused on solving adaptive traversability (AT) task for a tracked unmanned ground vehicle (Niftibot). The main goal was to propose a semi-autonomous system to control auxiliary articulated tracks of the robot, called flippers, so that the robot would be able to move fluently across terrain and to safely overcome obstacles on the way while forward speed and heading of the robot is controlled by its operator. The main contribution of such system is lowering cognitive load of the operator who is no longer required to adjust flippers manually.

The author took up previous progress on AT system which was based on *Q*-learning and decision trees. [2] Several flipper modes (configurations) were designed for various situations. The trees then estimated *Q*-value $Q_{AT}(\boldsymbol{F}, c)$. for all configurations $c \in C$ in particular state described by feature vector $\boldsymbol{F}$, which included Digital Elevation Map (DEM) of adjacent terrain. The mode with the highest *Q*-value was ultimately chosen for control. However, this system had difficulties when it had to deal with situations with incomplete terrain data. (It leaned only on laser scanner which barely detects reflective surfaces or performs badly in smoke leaving blank spaces in a model of environment.)

Therefore a new mode, denoted as NaN mode, was introduced to the system. This mode is invoked whenever there is too much missing information in DEM. All decision trees were re-learned on manually collected and annotated datasets in consideration of this extension. The learning process also included introduction of high-level features $\boldsymbol{F_H}$ extracted from the original feature vector $\boldsymbol{F}$ in an attempt to lower susceptibility to overfitting (see Chapter 6.2.1 and Appendix).

As a result of the learning process, new decision trees estimating *Q*-values have been obtained and it also included a tree which estimates *Q*-value of NaN mode $Q_{AT}(\boldsymbol{F_H}, 'NaN')$. This value serves as a safety measure determining whether the robot may move on or not. The value is highest when there is no information in DEM at all. On the contrary, it is negative when information in DEM is complete. When $Q_{AT}(\boldsymbol{F_H}, 'NaN')$ is higher than a specified treshold $\varepsilon = 0.25$, movement forward is forbidden, because there is little information in DEM and safety of robot cannot be guaranteed.

Kinova JACO robotic arm has been mounted on the top of the robot to deal with such situations. The robot is now able to touch terrain in front of itself and use terrain height

measured this way to update DEM. Concretely, an agent, which has been learned using reinforcement learning, selects an unknown DEM cell which may potentially bring useful information to the model. The arm touches terrain at coordinates of selected DEM cell, height value is retrieved and filled in DEM. $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, 'NaN')$ value is estimated again after the update and if it is still higher than ε, a new measurement has to be made. This means that arm is used to explore unknown terrain in front of the robot until there is enough information in DEM. Only after that the robot is allowed to choose optimal flipper configuration c^* with the highest estimated $Q_{AT}(\boldsymbol{F}_{\boldsymbol{H}}, c)$ and move forward.

Functionality of the implemented system was tested under lab conditions, see video-recordings on enclosed DVD. Artificial obstacles were used to simulate possible situations which may happen in real environment. It includes using aluminium foil which has a property that it reflects majority of laser rays, thus it is hardly detectable by laser scanner. The system performed quite well—the robot was able to touch terrain in front of itself. It made several measurements this way and finally chose viable flipper configuration to move forward across the aluminium foil.

Proposition of the future work includes testing of the implemented algorithm in a realistic environment. This process may also include collection of more training samples for learning of decision trees in order to improve the trained model. Better behaviour of the model is expected after that.

References

[1] Zimmermann, K.; Zuzanek, P.; Reinstein, M.; Hlavac, V., "Adaptive Traversability of unknown complex terrain with obstacles for mobile robots," *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp.5177,5182, May 31-June 7 2014

[2] Zimmermann, K.; Zuzanek, P.; Reinstein, M.; Petricek, T.; Hlavac, V., "Adaptive Traversability of Partially Occluded Obstacles," Robotics and Automation (ICRA), 2015 IEEE International Conference on; May 26-30 2015

[3] Mission. *TRADR: Long-Term Human-Robot Teaming for Robot-Assisted Disaster Response* [online]. [cit. 2015-05-01]. www: http://www.tradr-project.eu/?page_id=73

[4] Partners. *TRADR : Long-Term Human-Robot Teaming for Robot-Assisted Disaster Response* [online]. [cit. 2015-05-01]. www: http://www.tradr-project.eu/?page_id=30

[5] Fragoulopoulou, Z.; First TRADR EU FP7 Joint Exercise (TJEx). *Robohub* [online]. [cit. 2015-05-01]. www: http://robohub.org/first-tradr-eu-fp7-joint-exercise-tjex/

[6] *Bluebotics* [online]. [cit. 2015-05-01]. www: http://www.bluebotics.com/

[7] *Redmine, CIIRC, CTU*; internal documentation of TRADR project;
www: https://redmine.ciirc.cvut.cz/

[8] Ladybug3 1394b. *PointGrey* [online]. [cit. 2015-05-01].
www: http://www.ptgrey.com/ladybug3-360-degree-firewire-spherical-camera-systems

[9] *Kinova Robotics* [online]. [cit. 2015-05-01].www: http://kinovarobotics.com/

[10] O´Kane, J., M.; "*A Gentle Introduction to ROS,*" Version 2.1.1, Independently published; ISBN 978-14-92143-23-9; Oct. 2013; www: http://www.cse.sc.edu/~jokane/agitr/

[11] *ROS* [online]. [cit. 2015-05-03]. www: http://www.ros.org/

[12] Browsing packages for indigo. *ROS* [online]. [cit. 2015-05-03].
www: http://www.ros.org/browse/list.php

[13] Sutton, R., S.; Barto, A., G; *Reinforcement Learning: An Introduction*. Cambridge, Mass.: MIT Press, 1998, xviii, 322 p. ISBN 0262193981.

[14] Russell, S., J.; Norvig P.; Davis, E.; *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River: Prentice Hall, c2010, xviii, 1132 p. ISBN 0136042597.

[15] Watkins, C.; Dayan, P.; Technical Note: Q-learning; *Machine Learning*; Vol. 8, Issue 3-4; pp. 279-292; May 1992

[16] Byoung-Ho Kim, "Centroid-based analysis of quadruped-robot walking balance," *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pp.1-6, 22-26 June 2009

[17] Komatsu, H.; Endo, G.; Hodoshima, R.; Hirose, S.; Fukushima, E.F., "Basic consideration about optimal control of a quadruped walking robot during slope walking motion," *Advanced Robotics and its Social Impacts (ARSO), 2013 IEEE Workshop on*, pp. 224-230, 7-9 Nov. 2013

[18] Woosub Lee; Sungchul Kang; Munsang Kim; Mignon Park, "ROBHAZ-DT3: teleoperated mobile platform with passively adaptive double-track for hazardous environment applications," *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* , vol.1, pp.33-38, 28 Sept.-2 Oct. 2004

[19] Guarnieri, M.; Debenest, P.; Inoh, T.; Takita, K.; Masuda, H.; Kurazume, R.; Fukushima, E.; Hirose, S., "HELIOS carrier: Tail-like mechanism and control algorithm for stable motion in unknown environments," *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp.1851-1856, 12-17 May 2009

[20] Ohno, K.; Morimura, S.; Tadokoro, S.; Koyanagi, E.; Yoshida, T., "Semi-autonomous control system of rescue crawler robot having flippers for getting Over unknown-Steps," *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp.3012-3018, Oct. 29-Nov. 2 2007

[21] Yamada, Y.; Endo, G.; Fukushima, E.F., "Blade-type crawler vehicle bio-inspired by a wharf roach," *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp.806-812, May 31 2014-June 7 2014

[22] Colas, F.; Mahesh, S.; Pomerlau, F.; Liu, M.; Sirgwart, R., "3d path planning and execution for search and rescue ground robots," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 722-727. Nov. 2013

[23] Mihankhah, E.; Kalantari, A.; Aboosaeedan, E.; Taghirad, H.D.; Ali, S.; Moosavian, A., "Autonomous staircase detection and stair climbing for a tracked mobile robot using fuzzy controller," *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pp.1980-1985, 22-25 Feb. 2009

[24] Okada, Y.; Keiji Nagatani,; Kazuya Yoshida,; Yoshida, T.; Koyanagi, E., "Shared autonomy system for tracked vehicles to traverse rough terrain based on continuous three-dimensional terrain scanning," *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp.357,362, 18-22 Oct. 2010

[25] IRobot 510 PackBot®: the Multi-mission Robot. *IRobot* [online]. [cit. 2015-05-11].
www: http://www.irobot.com/for-defense-and-security/robots/510-packbot.aspx#PublicSafety

[26] Brunner, M.; Brüggemann, B.; Schulz, D., "Towards autonomously traversing complex obstacles with mobile robots with adjustable chassis," *Carpathian Control Conference (ICCC), 2012 13th International*, pp.63-68, May 2012

[27] Singhv, S.; et al., "Recent progress in local and global traversability for planetary rovers," In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. IEEE, 2000. pp. 1194-1200.

[28] Goldberg, S.B.; Maimone, M.W.; Matthies, L., "Stereo vision and rover navigation software for planetary exploration," *Aerospace Conference Proceedings, 2002. IEEE* , vol.5, pp. 2025-2036, 2002

[29] Shirkhodaie, A.; Amrani, R.; Tunstel, E., "Soft computing for visual terrain perception and traversability assessment by planetary robotic systems," *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol.2, pp.1848-1855, Vol. 2, Oct. 2005

[30] Chunxin Qiu; Xiaorui Zhu; Liping Liu, "Robot Terrain Inclination model extracted from laser scanner data for outdoor environment," *Mechatronics and Automation (ICMA), 2012 International Conference on*, pp.1727-1731, 5-8 Aug. 2012

[31] Braun Tim; Bitsch Henning; Berns Karsten; "Visual Terrain Traversability Estimation Using a Combined Slope/Elevation Model," *AI, KI 2008, 31st Annual German Conference on*, pp. 177-184 , September 23-26, 2008

[32] Martin, S.; Murphy, L.; Corke, P.; "Building Large Scale Traversability Maps Using Vehicle Experience," *Springer Tracts in Advance Robotics*, Vol. 88, pp. 891-905, 2013

[33] Dongshin Kim; Jie Sun; Sang Min Oh; Rehg, J.M.; Bobick, A.F., "Traversability classification using unsupervised on-line visual learning for outdoor robot navigation," *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp.518,525, 15-19 May 2006

[34] Hesch, J.A.; Mariottini, G.L.; Roumeliotis, S.I., "Descending-stair detection, approach, and traversal with an autonomous tracked vehicle," *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp.5525-5531, 18-22 Oct. 2010

[35] Okada, Y.; Keiji Nagatani,; Kazuya Yoshida,, "Semi-autonomous operation of tracked vehicles on rough terrain using autonomous control of active flippers," *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp.2815-2820, 10-15 Oct. 2009

[36] Šalanský, V.; Kontaktní průzkum terénu pro mobilního robota; Diplomová práce; Fakulta elektrotechnická, Čvut v Praze; květen 2015

[37] *Octomap_server: Package Summary* [online]. [cit. 2015-05-01].
www: http://wiki.ros.org/octomap_server

Appendix

A.1 Extraction of high–level features F_H from vector F

Vector $\boldsymbol{F}$ was introduced in chapter 6.1.1. It contains 210 members:

$\boldsymbol{F}$ = [100 x height values for each bin in DEM; pitch; speed_r (real speed of robot); speed_c (speed demanded by control); 6 x currents in tracks; roll; 100 x counter].

As it was mentioned in chapter 6.2.1, it was decided to use high-level features as input for decision trees. Five different models, which differ in the way how high-level feature vector $\boldsymbol{F_H}$ is constructed from $\boldsymbol{F}$, were made. This section provides information about construction of $\boldsymbol{F_H}$, namely it presents source codes of Matlab functions getFeaturesFromF.m, which are responsible for this process.

There are five files model_X.m which contain Matlab implementation of function getFeaturesFromF(F) on enclosed DVD. These files differ, each of them creates own collection of high-level features which are extracted from vector $\boldsymbol{F}$. Source code of getFeaturesFromF(F) for Model 4 (model_4.m on DVD) is presented as an example in the following section.

A.1.1 Model 4

```
function [features] = getFeaturesFromF(F)
    DEM = F2DEM(F);

    PITCH = F(101);
    ROLL = abs(F(110));

    LIN_AVGS = zeros(8,1); index = 0;
    for line = 9:16
        index = index + 1;
        DEM_line = DEM(line,:);
        if (sum(isnan(DEM_line)) > 3)
            LIN_AVGS(index) = NaN;
        else
            LIN_AVGS(index) = nanmean(DEM_line);
        end
    end
    LIN_AVGS_STD = nanstd(LIN_AVGS);
    NAN_INFO = isnan(LIN_AVGS(3:6));

    lin_averages = LIN_AVGS(isfinite(LIN_AVGS(3:end)));
    gradient = diff(lin_averages);
    GRAD_MAX = max(gradient);
    if(isempty(GRAD_MAX))
        GRAD_MAX = NaN;
        GRAD_MAX_nan = 1;
    else
        GRAD_MAX_nan = 0;
    end
    GRAD_MIN = min(gradient);
    if(isempty(GRAD_MIN))
        GRAD_MIN = nan;
        GRAD_MIN_nan = 1;
    else
        GRAD_MIN_nan = 0;
    end
    GRAD_MEAN = mean(gradient);

    features = [PITCH;
        ROLL;
        LIN_AVGS;
        LIN_AVGS_STD;
        NAN_INFO;
        GRAD_MAX;
        GRAD_MAX_nan;
        GRAD_MIN;
        GRAD_MIN_nan;
        GRAD_MEAN;
        ];
end
```