UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR ROBOTIK
UND KOGNITIVE SYSTEME

# Traversability and Mapping of Obstacles in Uneven Terrain Based on a Robot's Design

*Befahrbarkeit und Kartierung von Hindernissen in unebenem Gelände basierend auf dem Aufbau eines Roboters*

**Bachelorarbeit**

verfasst am
**Institut für Robotik und Kognitive Systeme**

im Rahmen des Studiengangs
**Robotik und Autonome Systeme**
der Universität zu Lübeck

vorgelegt von
**Max-Ole Bastian von Waldow**

ausgegeben und betreut von
**Dr. Ngoc Thinh Nguyen**

mit Unterstützung von
**Prof. Dr. Georg Schildbach**

Lübeck, den 1. Februar 2022

## Eidesstattliche Erklärung

*Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.*

Max-Ole Bastian von Waldow

## Zusammenfassung

In unstrukturierten Umgebungen ist die Stabilität eines Roboters von dessen Design und Bewegung abhängig. In dieser Bachelorarbeit wird die Befahrbarkeit von unebenem Gelände bestimmt und an die Fähigkeiten eines mobilen Roboters angepasst. Wir modellieren gefährliche Situationen, die den Roboter immobilisieren oder vom Weg abbringen könnten. Dazu gehört Umkippen, scharfe Kurven, Motorüberlastung oder das Feststecken an Kanten. Aus der Konfiguration des Roboters bestimmen wir Grenzwerte für die befahrbare Steigung und Kantenhöhe. Hindernisse werden mittels dieser Werte erkannt und in einer lokalen Karte um den Roboter herum eingetragen.

Der vorgestellte Prozess soll modular und dadurch einfach einzubinden sein. Er kann in Verbindung mit einfacher bis zu fortgeschrittener Kontroll-Software eingesetzt werden. Einfache Software erfordert vorsichtige Annahmen während fortgeschrittene detailierter integriert werden kann und weniger einengende Karten ermöglicht. Wir testen unser System auf einem Skatepark und demonstrieren die Unterschiede durch mehr und weniger fortgeschrittene Kontroll-Software, wie auch die Performanz als Ganzes.

## Abstract

In unstructured environments, a robot's stability depends on its design and motion. In this thesis the traversability of rough terrain is determined and tailored to the abilities of a mobile robot. We model dangerous situations that could immobilize the robot or interfere with its desired path. These include tip-over, sharp turns, the lack of motor power or getting stuck on edges. The configuration of the robot is compiled to the generalized limit of the traversable inclination and edge height. Obstacles are detected through these limits and mapped inside a local region around a robot.

The presented process aims to be modular for easy adaptation. It is combined with control software that ranges from basic to more sophisticated. Simple controllers require conservative assumptions while more advanced software is integrated more tightly and allows for less restrictive maps. We employ the system in a field test at a skate park and illustrate the difference the type of control software makes and the performance as a whole.

# Contents

# 1

# Introduction

Automation has found its way into many parts of modern life. The adoption of robots mostly in production and research was spearheaded indoors. Man-made structured environments are well suited for robots and oftentimes we adapt the surrounding to better fit them. In rough terrain however robots need to adapt themselves.

## 1.1 Motivation

Navigating in difficult terrain is increasingly gaining importance as robots advance the fields of agriculture, extraterrestrial exploration as well as search and rescue. Automation has potential in facilitating labor-intensive weed control in agriculture [24]. By selectively removing weeds, there is less need to treat plants with pesticides. This in turn reduces the impact on insects and lessens environmental pollution in general [26]. When used for farming, the robots must navigate fields and rows of plants.

Space exploration is another growing field that sets high expectations for the robots used [22]. Once a robotic mission is sent, it must work reliably over an extended period of time and operate in harsh environments and unprepared terrain. Exploring risky terrain opens up further chances of gaining new insights. On the other hand, without the possibility of physical human intervention, an expensive extraterrestrial operation would come to an end if the robot were to fail, e.g. by getting stuck.

In addition, the continued work in rescue robotics promises to be of life-saving use after disasters. The German Rescue Robotics Center (DRZ) [14] researches the use of autonomous systems in such scenarios. Not only are the robots supposed to aid victims but also to protect and assist emergency forces with their work. Here as well the destroyed infrastructure requires robots that can navigate the ruins.

In these use cases it is essential that robots cope with unknown terrain. To ensure mission success, the robots must safely pass over challenging areas. Otherwise, they risk immobilization and damage. In order to prevent this, mobile robots need to map the environment and autonomously decide where they can drive and where they encounter obstacles. In rough terrain the regions that need to be classified as untraversable depend on the design of the robot. Design choices such as motor power, ground clearance, the position of wheels and center of gravity and more affect the ability to safely navigate.

## 1.2 Related Work and Contributions of this Thesis

In this thesis we present an approach to determine the traversability of the environment. The dynamic case of the robot tipping is looked on as well as other failure cases. The method considers the robot's design to find out what regions can be traversed. We provide safe driving zones in rough terrain, and the results are saved in a map for easy access by other software. The method aims to be as modular as possible which makes it easily adoptable in other mobile robotics projects. This has its own drawback of being less tightly integrated and requiring conservative assumptions about third party software. Our system is configurable for any robot with roughly similar design simply by changing some characteristic parameters. These are also analyzed in the process and the user is informed about what restricts the robot's ability the most to traverse uneven terrain, aiding in the design of a robust mobile platform.

### Comparison with Related Work

Mapping and path planning around obstacles in indoor environments are well studied topics. Established tools for these structured and 2-dimensional scenarios are readily available [15]. Fewer solutions exist for navigating rough terrain. Mobile robots need to be able to assess whether and how to traverse difficult regions.

Morales et al. [17] demonstrate an approach to improve the traversability of sloped terrain by a mobile robot with an attached robotic arm. They rotate the manipulator to shift the center of gravity of the whole robot in order to reduce the risk of tipping over. This is done assuming the static case with a safety margin to account for dynamic effects. Whereas we also take the robot's acceleration into consideration. This is required since our robot does not have a manipulator nor the ability to move its center of gravity. In their work the center of gravity is adjusted based on data from an onboard inertia measurement unit that reads how the mobile base is tilted. In contrast, our approach utilizes a map of the robot's vicinity which allows us to predict such states of the robot and react to slopes prior to being at that position.

The path planning approach by Hertle and Dornhege [12] respects the robot's configuration and finds a suited path through rough terrain. They consider many aspects of traversability regarding ramps and steps that we also investigate in this thesis and substantiate with real life experiments. Their method uses a variant of A* search, that checks for valid movement options when expanding the graph further. The process requires a pre-built 3-dimensional map while our method simultaneously maps the environment and determines its traversability. They integrate the stability of the robot well with a path planner. Whereas we try to separate those as much as possible to keep a modular structure.

Another path planning method is presented by Nguyen et al. [19]. Their approach uses optimized B-splines to generate smooth paths within polytopes. A 2-dimensional map of obstacles and free regions is required to extract polytopes in which it is safe to drive. The traversability of the terrain is not yet considered, but our work focuses on this mapping aspect and could be combined with their path planning approach. With this extension other methods can easily be transformed to work in rough terrain.

## 1.3 Outline of this Thesis

In the next part, Chapter 2, we present the robotic platform along with the ranged sensor utilized in this thesis. We also talk about the software framework and the third party packages used. In Chapter 3 we take a look at how to describe untraversable regions in rough terrain. There we introduce mathematical models and perform experiments to figure out the limits of what obstacles our robot can overcome. Then in Chapter 4, with the help of what we learned from the tests, we implement a process to calculate traversable regions and to save them in a map. We also present alternative versions to use with modified robot control software. Chapter 5 contains the evaluation of our method. It is combined with third party path planning software and the results of a final test are compared. Concluding this work in Chapter 6 we summarize our findings and give an outlook on what remains to be done.

# 2

# Fundamentals

Here some fundamental definitions are given that we rely on in this thesis. We introduce the hardware and the software packages and why we choose to use them for this project.

In terms of notation, we use the `typewriter font` throughout the thesis for filenames, function names, package names and data layers within our implementation. We denote a vector as $\vec{x}$, and the length of this vector $x$ using the same letter without the arrow on top.

## 2.1 Robotic Platform

In this thesis a robot with four wheels is assumed for the calculations and experiments proposed, however many concepts could be generalized for other robot designs as well. The mobile base we use as an example for such a robot, is a Jackal unmanned ground vehicle (UGV) built by Clearpath [4]. It is differentially driven and thus uses skid steering to turn. This means that the wheels are fixed in place and slide across the ground as the robot rotates. The Jackal is designed to be a rugged robotics research platform. It is well suited to be used outdoors which is why it is optimal for our case.

A ZED camera [21] is used as a sensor to perceive the environment in 3D and in detail. The sensor is a front facing stereo camera that is integrated with ROS through the `zed_wrapper` package. This software takes the synchronized videos from the left and right camera and is able to create depth images (images with depth information) and pointclouds (a collection of ranged measurements in 3-dimensional space) among other things. It also uses this information to perform visual odometry. By doing so the camera can provide the estimated pose of the mobile robot with its covariance.

The ZED camera is mounted on the Jackal along with a LIDAR (Light Detection and Ranging) sensor that we do not use in this thesis. The Jackal is additionally modified with an Nvidia Jetson. This computing unit aids the LIDAR with extra resources. These components add to and shift the weight of the robot, which we have to take into consideration. There is a protective foam placed around the LIDAR, which is visible in images of experiments. Our robotic platform with its extensions is shown in Figure 2.1.


Figure 2.1: The Jackal UGV is used as the robotic platform. It is shown with the ZED camera (1), an Nvidia Jetson inside its housing (2) and a LIDAR (3) that is covered by a protective foam during experiments.

## 2.2 Robot Operating System Framework

The Robot Operating System (ROS) [23] manages the software of the robot and is the backbone of the project. We use the ROS version Melodic Morenia that is installed on Ubuntu version 18.04. The framework combines many third party tools, and we can add our own software. ROS is a modular system with a standardized interface for software run with it. This allows us to easily put different software packages together and split tasks between them. We embrace the concept of modularity in this thesis and ROS helps us to do so.

Executables run with ROS are called nodes and ROS provides the basis for communication between them. Nodes can communicate through topics, where one or more nodes can publish a message to a ROS topic and other nodes can subscribe to topics to receive the messages. Communication through services is possible as well. In that case a node may request information from another node that advertises a service and then answers the request.

ROS also simplifies launching nodes with different configurations, which are commonly written in the human-readable YAML format. YAML is a recursive acronym for "YAML Ain't Markup Language". Configurations can be loaded onto a parameter server within ROS from where nodes have access to them.

## 2.3 Mapping Software

Mapping is the process of gathering information about the environment and then being able to provide it to other software tools in a well packaged format. The purpose of the mapping process discussed here is to provide information about the traversability of different regions to other modules. For example a path planner could use this information to stay inside the safe driving zones and avoid obstacles. The usage of the map as part of localization is not part of this thesis. Rather the focus lies on interpreting a height map under the aspect of traversability.

### Elevation Mapping

The `elevation_mapping` package [7, 8] is used to map the environment. It uses range measurements and the robot motion, both of which we provide with the ZED camera. The software integrates the sensor data into a map considering uncertainties in the distance measurements and the robot's pose estimate which degrades over time. We do not make use of the package's probabilistic capabilities in this thesis and only use the best estimate of the terrain.

By default, the map generated is robot-centric. The region covered by the map follows the robot as it moves and always contains its local environment. This means that it is less suited for cartographing a large space and more suited for analysis of the robot's near vicinity. The authors, Fankhauser et al., of the `elevation_mapping` package use the local information to determine where to step safely with a legged robot, and similarly we use the robot-centric map to determine where to drive safely. The map size can however be increased such that the differences between a robot-centric and a global map become less relevant.

### Grid Map

The `grid_map` library [9] is a mapping framework specialized on rough terrain. The grid map data structure consists of a 2-dimensional grid with multiple layers. This means that for each cell in the grid a variety of values can be saved where every value is part of a layer. An arbitrary amount of further layers can be added, each containing another type of information. While each layer is independent of the others, the same index in each layer corresponds to the same cell. The data structure is illustrated in Figure 2.2.

The `elevation_mapping` package requires this library and uses such a layer, the `elevation` layer, to write the height of the environment into each cell, resulting in an 2.5-dimensional map. Such a map only keeps a single elevation value per cell and cannot represent overhangs like a 3-dimensional map could. Due to the efficiency benefit that comes with handling a lower dimensional space, it is a good trade off for ground based vehicles like the Jackal.

The grid map is implemented with a 2-dimensional circular buffer that allows for efficient movement of data within the map as the robot moves, without having to copy the data in memory. The library also includes multiple options to visualize the data. It is distributed together with the `grid_map_visualization` package that helps show

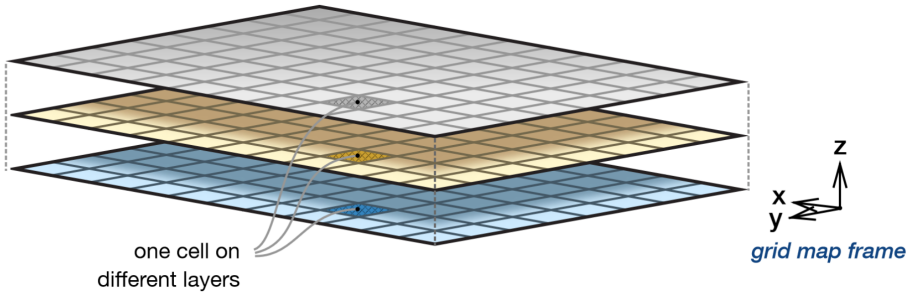



Figure 2.2: The grid map data structure with 3 layers. A single cell has an independent value in each layer. The image is taken from the grid map publication [9].

individual layers by converting them into other common data structures. We use this feature to export individual layers as an occupancy grid [18, 6], a simple 2-dimensional grid that is often used as a map in mobile robotics to keep track of the probability of a cell being occupied by an obstacle. The conversion does not use the occupancy mapping algorithm to populate the map, but sets the values depending on a grid map layer and a threshold.

### Filter Chain

A grid map features the ability to be processed with a number of filters by the `grid_map_filters` package that builds upon ROS filters [20]. The filters allow us to efficiently process many grid map cells at once. Each filter accepts one or multiple data layers as an input and writes its output into another layer. Linking these filters together, using the output of one filter as the input for another, creates the whole filter chain. The filter chain is conveniently configured in a YAML file without the need for custom code. In Chapter 4.2 we create a filter chain to convert the `elevation` layer from the `elevation_mapping` package into one or more layers that hold information about the traversability for each cell.

# 3

# Methods

In this Chapter we break down untraversable regions of the environment into the general features of slopes and edges. Those are then dealt with as separate sub-problems and finally merged into a final traversability representation. We present each sub-problem with a mathematical model and search for the threshold at which the region becomes untraversable. We perform various experiments to verify the theoretical models, and we discuss the findings.

## 3.1 Exploring Instability

In order to find out how to generalize obstacles or untraversable regions we look into situations that can be dangerous for a mobile robot. An unsafe environment might immobilize the robot or cause it to move differently than the control software commands. For example the robot could slip or be diverted by an obstacle that the control software does not know about. And the robot might get immobilized e.g. if it gets stuck or is tipped over. We call those undesired states unstable and the environment that leads to such states is called unsafe.

In an indoor environment such a state might be that the robot is hitting a wall. In most cases indoor obstacles are vertical and block the otherwise traversable floor. Outdoors in unstructured terrain however, many obstacles may not simply present themselves as objects blocking a path. Rather the line between safe and unsafe terrain is blurred and depends on the design of the robot in question.

The design includes properties of the robot that affect its ability to traverse rough terrain. First the motor power must be enough to control the wheels. When the wheels are turning, friction plays a major role to transform the rotation of the wheels into movement of the robot. To ensure this friction, the wheels must contact the ground and thus tip-over immobilizes the robot. Tipping depends on the center of gravity and the position of the robot's wheels. The motor power becomes relevant on sloped ground when the robot must move its own weight. Slopes also demand enough friction to keep the robot from sliding due to gravity. Additionally, tip-over is more common on inclined terrain. Similarly, edges require the robot to lift its weight and friction is needed for the wheels to grip the edge. A robot's geometry also constraints the edges that can be overcome. We

see that slopes and edges are the types of terrain that we need to consider when it comes to traversability.

### Robot behavior affecting the traversability

Our goal is to keep the whole system as modular as possible. That way other existing third party ROS modules can easily make use of our work. On the other hand, modularization has the drawback that our method is less closely integrated with other parts of the software. Most importantly we have no knowledge of the inner workings of the software controlling the robot. We refer to such third party software as a controller. The ability to traverse an obstacle partly depends on the movement of the robot which we cannot predict, and thus a dilemma arises. The controller e.g. requires the position of obstacles in the map, but these would change depending on the motion that the controller has planned in each part of the map. To circumvent this issue, we need to assume the worst case for the robot's behavior when calculating the traversability of the environment. By using conservative thresholds for obstacles, we create a map that can guarantee safe locations no matter the robot's behavior.

## 3.2 Traversability of Slopes

Unlike indoor environments the outside terrain rarely consists of perfectly level ground. A mobile robot has to be able to determine which inclination is safe to drive on. Slight slopes resemble level ground close enough to be traversable however the change to an untraversable slope can be gradual. We choose to consider the slope of the terrain to determine traversability because it is a highly general feature of obstacles. This way we cover walls, hills, ditches, holes and more, all with a single angle to find out if they can be overcome. By using this abstract feature, the system is more robust to unpredicted obstacle types. We determine the limit of a traversable inclination by modelling the following unstable scenarios. If a slope is too steep, the robot could tip over, slip off or stall because its motors lack the power to handle the weight of the robot.

### Tipping

If a robot is tilted too much because it resides on a slope, it will tip over. In the static case where the robot does not move, a measure of stability can be the distance of the center of gravity projected downwards to the edge of the supporting polygon, which is the convex hull of the contact points of the robot with the ground [16]. For the Jackal this is a rectangle with the bottom of the wheels as vertices. Tipping begins as soon as the center of gravity lies above an outer contact point with the ground. For mobile robots however we need to consider the dynamic case of the robot accelerating. In this part we introduce formulas to find the steepest safely traversable slope, we calculate the maximal centripetal acceleration and we determine the center of gravity of the Jackal.

As we do not know how the robot will move, we assume the most unstable case of the robot accelerating uphill with its maximum acceleration. We simplify the situation by viewing the robot from the side and in 2 dimensions as illustrated in Figure 3.1. We

assume the state in which the robot is just beginning to tip, and we regard the moments around the tipping point $T$ at the contact point of the slope with inclination $\alpha$ and a wheel at the lower end of the robot. There are no forces acting in other contact points because the wheels there would lift off the ground as the robot tips. Now the condition for tipping is met if the moment due to the inertial force $\vec{F}$ is greater or equal to the moment induced by the gravitational force $\vec{G}$. Note that it is not guaranteed that the robot falls over completely once it begins to tip. It could just rock back standing upright if the acceleration or slope decrease in time. The robot might recover from tipping, but we do not consider rocking of the robot as being stable. It is risky, and the robot would perform involuntary movement.

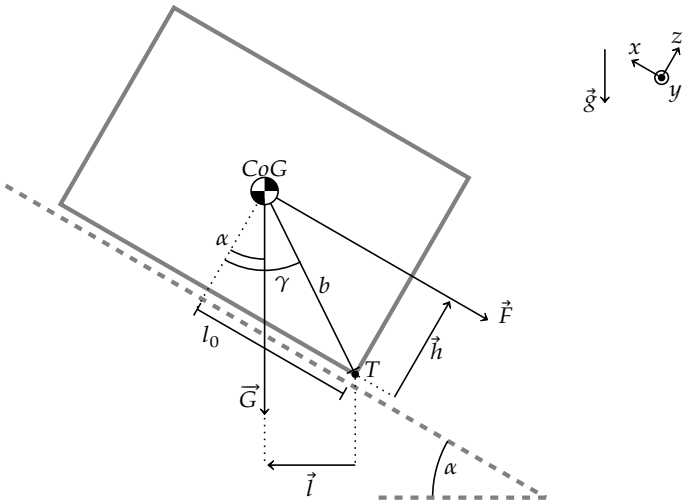



Figure 3.1: Side view of the Jackal at an incline. It shows the dynamics of tipping over at a slope.

From the equality of moments $\overrightarrow{M_i}^{(T)}$ around the point $T$ at the limit of tipping we get:

$$0 \stackrel{!}{=} \sum_{i \in \mathbb{N}} \overrightarrow{M_i}^{(T)} = \vec{l} \times \vec{G} - \vec{h} \times \vec{F} \quad \Big| \text{ with } \vec{l} \perp \vec{G}\,,\ \vec{h} \perp \vec{F} \quad \Big| \text{ and } F = ma\,,\ G = mg \tag{3.1a}$$

$$\Longrightarrow \quad ah = gl \tag{3.1b}$$

$$\Longleftrightarrow \quad \frac{ah}{g} = b \sin(\gamma - \alpha) \tag{3.1c}$$

$$\text{Where} \quad \gamma = \arctan\left(\frac{l_0}{h}\right) \tag{3.1d}$$

$$\text{and} \quad b = \sqrt{h^2 + l_0^2} \tag{3.1e}$$

Here $h$ denotes the height of the center of gravity $CoG$ from the ground and $b$ is its distance from the tipping point $T$. The gravitational acceleration is indicated by $\vec{g}$. The robot accelerates with the acceleration $\vec{a}$ in the direction opposing the inertial force $\vec{F}$. The lever arm of the gravitational force $\vec{G}$ has the length $l$ and $l_0$ is the length $l$ if the incline is zero. It is the distance of the center of gravity to the tipping point along the axis of acceleration,

and it can be measured easily along the body of the robot. The maximal slope $\alpha$ that is traversable is calculated by rearranging (3.1c) as

$$\alpha = \gamma - \arcsin\left(\frac{ah}{gb}\right) \tag{3.2}$$

### Centrifugal acceleration in tight turns

The Jackal can only accelerate forwards or backwards. Yet, it is possible that the robot tips to one of its sides. A sharp turn would facilitate this, so instead of setting the acceleration in the formula above to zero, we calculate the centripetal acceleration. As we do not know what movement the robot will perform, we need to assume the most unstable trajectory possible. In the worst case, the robot drives downhill at its maximal velocity and then sharply turns to face uphill. In the curve the robot experiences the highest centripetal acceleration while additionally being tilted to its side by the slope. Figure 3.2 illustrates the free body image of a robot turning as viewed from the top. The outer wheels need to turn the fastest in the curve, so they drive at the robot's maximal velocity $v_{max}$ while the inner wheels need to slow down or even rotate backwards for turning. We are interested in the acceleration $a_{tip}$ that would induce tipping to the robot's side. $a_{tip}$ is a projection of the centripetal acceleration $a_{CoG}$ at the center of gravity ◔ onto the axis perpendicular to the driving direction.

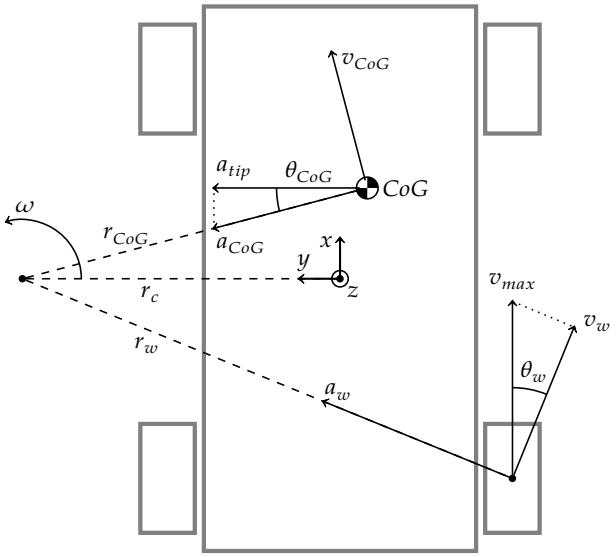



Figure 3.2: Top view of the Jackal turning. It shows the tipping acceleration $a_{tip}$ as part of the centripetal acceleration $a_{CoG}$ which depends on the curve radius $r_c$ and the maximal velocity $v_{max}$ of the drive.

The position of the center of gravity $\left(CoG_x, CoG_y\right)$ and the position of the wheel $\left(Wheel_x, Wheel_y\right)$ are constant. Note that we could also have chosen the other outer wheel as the front and back drive of the Jackal operate symmetrically. In case of a robot with Ackermann steering, one would have to choose the fixed outer wheel and the coordinate system needs to be moved such that the center of the curve still lies on the $y$-axis.

We can express many non-fixed variables above with the curve radius to the robot center $r_c$. Using trigonometry we derive formulas for the lengths of the radii $r_w$ and $r_{CoG}$, the distance from the center of the curve to the outer driven wheel and the center of gravity respectively. Similarly, there is a closed form for the angles $\theta_{CoG}$ and $\theta_w$.

$$r_{CoG} = \sqrt{\left(r_c - CoG_y\right)^2 + \left(CoG_x\right)^2} \tag{3.3a}$$

$$r_w = \sqrt{\left(r_c - Wheel_y\right)^2 + \left(CoG_x\right)^2} \tag{3.3b}$$

$$\theta_{CoG} = \arctan\left(\frac{\text{abs}\left(CoG_x\right)}{\text{abs}\left(r_c - CoG_y\right)}\right) \tag{3.3c}$$

$$\theta_w = \arctan\left(\frac{\text{abs}\left(Wheel_x\right)}{\text{abs}\left(r_c - Wheel_y\right)}\right) \tag{3.3d}$$

We can find a formulation of the angular velocity $\omega$ from the radius and the velocity of both the wheel and the center of gravity. For $a_{CoG}$ we use the formula of the centripetal acceleration.

$$\omega = \omega_w = \frac{v_w}{r_w} \tag{3.4a}$$

$$= \omega_{CoG} = \frac{v_{CoG}}{r_{CoG}} \tag{3.4b}$$

$$a_{CoG} = \frac{v_{CoG}^2}{r_{CoG}} \tag{3.4c}$$

Finally, we insert the mathematical terms into the projection $a_{tip}$ of the acceleration $a_{CoG}$.

$$a_{tip} = a_{CoG}\cos\left(\theta_{CoG}\right) \tag{3.5a}$$

$$= \frac{v_{CoG}^2}{r_{CoG}}\cos\left(\theta_{CoG}\right) \tag{3.5b}$$

$$= \omega_{CoG}^2 r_{CoG}\cos\left(\theta_{CoG}\right) \tag{3.5c}$$

$$= \frac{v_w^2}{r_w^2} r_{CoG}\cos\left(\theta_{CoG}\right) \tag{3.5d}$$

$$= v_{max}^2 \frac{r_{CoG}}{r_w^2}\cos^2\left(\theta_w\right)\cos\left(\theta_{CoG}\right) \tag{3.5e}$$

We now have the maximal centripetal acceleration as a function of the turn radius $r_c$ and constants from the robot configuration. To find out what radius results in the worst trajectory which we need to consider, we solve a simple non-linear optimization problem. The only constraint is $r_c \geq 0$. A negative radius would be a curve in the other direction. We assume turning in either direction to be equal, but we would need to consider an opposite wheel. We solve the problem in Matlab using the `fmincon` optimizer. Alternatively, as there is only a single variable, we can just try a range of values above zero and choose the highest acceleration. Both yield identical results when tested. The effect of the radius on the centripetal acceleration is depicted in the plot 3.3. Starting with a straight path, as

the curve gets tighter, $a_{tip}$ grows because the angular velocity increases. Beyond a certain point $a_{tip}$ drops again, which is because the velocity of the center of gravity decreases as the inner wheels slow down further to keep turning at that tight radius. The Jackal has the ability to turn on the spot. If a robot only has limited turning capability and cannot drive a curve with the radius for the worst trajectory, then its minimal turning radius can be used to calculate the centripetal acceleration instead of the $r_c$ from the optimization.

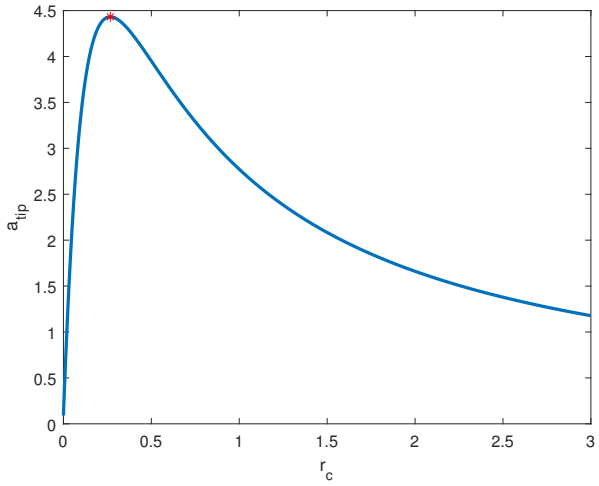



Figure 3.3: Optimization problem of obtaining the maximal centripetal acceleration $a_{tip}$ by driving a curve of radius $r_c$

## Center of gravity calculation

One last step required for the calculation of the tipping limit is to determine the position of the center of gravity. Oftentimes there is a digital model of the robot available that specifies the mass of different components and from which one can obtain the center of gravity. Because of the accessories added to the Jackal, its standard digital model only provides a rough estimation. We need to measure the position ourselves. A widely used method is to place an object on three or more load cells and to calculate the center of gravity in the plane from the difference in weight on each scale [1]. By reorienting the object and repeating the process the axes on which the center of gravity must lie can be intersected to gain the 3-dimensional position.

For our experiment we only have access to a single scale. We place it successively under each wheel of the Jackal while the other three wheels are hoisted up to the same height. It becomes apparent quickly that this method is flawed. Slight repositioning of the scale leads to vastly different measurements for the same wheel. We suspect this issue comes from the four contact points instead of three. The simple kitchen scale we use compresses by up to 2 mm under weight, which is enough to redistribute the weight. This could be mitigated by building a platform for the Jackal that then has three contact points for the measurement, instead we opt for another method.

If an object is hanged from a point, by definition the center of gravity will always be

directly below that point. By using two points after another the axes can be intersected to obtain the center of gravity. Since hanging the Jackal above the ground risks damaging the robot, we reverse the setup. If an object rests on an edge and is balanced above it, the center of gravity will always be directly above that edge. For each measurement we find a plane in which the center of gravity must lie. This needs to be repeated at least three times until the intersection of the planes result in a 3-dimensional point. The Jackal conveniently has a rectangular metal base. We hoist the robot up and tip it such that it balances on one edge at a time. We let the Jackal rest on its body instead of tipping it around the tires because the metal does not deform as the tires would, resulting in more accurate measurements. This setup is shown in Figure 3.4. When the Jackal is balanced at its equilibrium point, we measure the angle by which it is tilted. We found the most reliable way to do so is to measure the lengths of a triangle that spans from the edge to a mark on the ground to the lifted edge of the Jackal and to then get the angle from the law of cosines. The resulting position of the center of gravity is higher up than estimated with the digital model which is due to the accessories added on top. The position is also shifted to the front and a bit to the right of the geometric center of the Jackal. The precision is the lowest along the forwards and backwards axis. The redundant measurements deviate by 1.74 cm in that direction. This method of measurement is quite crude but precise enough for our purposes where we are fairly conservative.

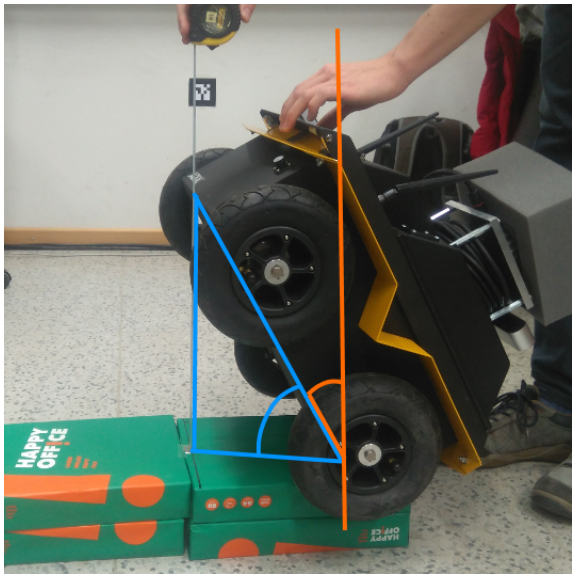



Figure 3.4: Center of gravity measurement. The plane in which the center of gravity must lie is illustrated in orange. It includes the edge the Jackal is balanced above and is oriented by the orange angle which is calculated by the side lengths of the blue triangle. The robot is tipped around all four sides of its base and a plane is determined for each.

### Slippage

The Jackal is especially prone to slippage because of its skid steering. If the robot turns, the wheels must slide across the ground. The static friction then turns into kinetic friction,

which is usually less, and thus the Jackal loses friction when turning. On an incline this leads to the Jackal slipping down the slope. We consider this case unstable as the robot moves in a direction other than the one desired.

Assuming that the amount of slippage is related to the inclination, we try to find the maximal slope at which the slippage is tolerable. Measurements of friction are difficult to take [13], especially when the friction is predicted for spots in the map where the robot was not already at [25]. Instead, we try to find the maximal slope empirically by performing an experiment as illustrated in Figure 3.5. A ramp is built from a reinforced board of wood that leans on and is strapped to a staircase railing. The ramp is easy to reposition for taking measurements at different angles. On the downside the skid steering induces vibrations in the wooden board. The robot is set to drive 80 cm up this ramp. It then turns using skid steering to face 90° to the left, then it turns by 180° to face to the right and then turns by left 90° to face back upwards. Finally, the amount that the Jackal dropped is measured on either side and compared to measurements at different inclinations. The results of this procedure are listed in Table 3.6.

Figure 3.5: The slippage experiment is set up with a wooden board positioned at different inclinations. The Jackal drives on the ramp in a preprogrammed manner.

We observe that the right side of the robot tends to drop more, which may be due to the programmed movement pattern. From this gathered data we find no correlation between the angle of the slope and the drift due to skid steering. Unlike expected, the robot does not drop more as the slope increases. Therefore, we ignore slippage when exploring the safe inclination limit of a slope. A probable explanation for this behavior could be that the overall friction loss by skid steering is reduced as the weight of the tilted Jackal is not distributed equally between the wheels. At increasing inclinations this would reduce the slippage while at the same time the normal force $F_N$ decreases and leads to more slippage. These two effects seem to cancel out for the most part.

Furthermore, this setup is useful to validate formula (3.2) from the segment about the tip-over limit of the Jackal. We set the Jackal's acceleration in software to a range of values

Table 3.6: Results of the slippage experiment, depicting the distance the Jackal slipped at different inclinations after performing a preprogrammed motion. The overlaid bar diagram corresponds to this amount of drop. At 0° there is no drop, but significant amounts appear at even slight angles.

| Angle of slope [deg] | Distance slipped [mm] | |
|---|---|---|
| | right wheel | left wheel |
| 10 | 570 | 570 |
| 15 | 715 | 675 |
| 20 | 615 | 600 |
| | 705 | 685 |
| 25 | 900 | 900 |
| | 695 | 695 |
| 30 | 690 | 690 |
| 35 | 700 | 695 |

and let the robot accelerate uphill at these accelerations. The results of this experiment are gathered in Table 3.7. We see that the Jackal begins to fail at an acceleration of 0.9 $\frac{\mathrm{m}}{\mathrm{s}^2}$ at a 35° inclination. To check formula (3.2) we need $\gamma$ and the length $b$ with the formulas (3.1d) and (3.1e) respectively. We input the acceleration limit 0.9 $\frac{\mathrm{m}}{\mathrm{s}^2}$ for $a$, the height of the center of gravity from the ground which is 0.1707 m for $h$ and the distance of the center of gravity from the back wheels 0.1419 m for the length $l_0$, because the Jackal drives forwards and therefore tips around the back wheels. The final formula (3.2) gives us the angle at which the given parameters would lead to tip-over and as expected the result comes out to be 35.6906° which is close to the 35° from our experiment.

Table 3.7: Results of accelerating the Jackal up a ramp with different inclinations given in degrees at different accelerations in $\frac{\mathrm{m}}{\mathrm{s}^2}$. Pass signals a safe ascent while fail means that the robot started tipping. At the entry pass/fail the result is inconclusive. Both cases are present at equal amounts.

| Acceleration | Angle | |
|---|---|---|
| | 35 | 40 |
| 0.02 | pass | fail |
| 0.20 | pass | |
| 0.50 | pass | |
| 0.75 | pass | |
| 0.85 | pass | |
| 0.90 | pass/fail | |
| 0.95 | fail | |
| 1.00 | fail | |

### Lack of Motor Power

The steeper a slope gets, the more the robot has to fight gravity to safely traverse it. A robot could lack the power to drive upwards or to brake when driving downwards. In these cases, given the fixed mass $m$ and wheel radius $r$ of the robot, the motor cannot provide the wheels with the torque $\vec{T}$ necessary to accelerate or decelerate the wheels. We solve the formula for the angle $\alpha$ to get the slope limit for the motor capability.

$$\vec{T} = \vec{r} \times \vec{F} \quad \Big| \text{ with } \vec{r} \perp \vec{F} \text{ and } F = mg \sin\alpha \tag{3.6a}$$

$$\iff \quad T = rmg \sin\alpha \tag{3.6b}$$

$$\iff \quad \alpha = \arcsin\left(\frac{T}{rmg}\right) \tag{3.6c}$$

The data sheet of the Jackal [5] does not provide the wheel torque. In case the torque is not known, we can calculate it from the drive power that the manufacturer of the robot provides instead. To do so, we get the force in the unit Newton [N] from the definition of the unit Watt [W] and its SI units meter [m] and second [s]. We see that we can express the force $F$ through the maximal power $P$ of the drive and the velocity $v$ at that wattage, the robot's maximal speed.

$$1[\mathrm{W}] = 1\frac{[\mathrm{N}][\mathrm{m}]}{[\mathrm{s}]} \tag{3.7a}$$

$$\iff \quad 1[\mathrm{N}] = 1\frac{[\mathrm{W}][\mathrm{s}]}{[\mathrm{m}]} \tag{3.7b}$$

$$\implies \quad T = rF = r\frac{P}{v} \tag{3.7c}$$

With a wheel radius of $r = 9.8$ cm, a maximal speed of 2.0 $\frac{\mathrm{m}}{\mathrm{s}}$ and 500 W of drive power, the torque comes out to be 24.5 Nm. We weigh the Jackal including its accessories and determine its mass to be 24 kg. With this torque and the relatively low mass, the argument of the arcsin in equation (3.6c) is above 1. This means we cannot determine an angle limit, because the Jackal would be powerful enough to traverse even a vertical slope if only the lack motor power were an issue. The Jackal is built for rough terrain and has a transmission that reduces the wheel speed in favor of torque. In general the wheel torque imposes a limit on the traversable slope, however with the Jackal torque is not the critical factor. It loses traction far before the motor fails to control the wheels.

## 3.3 Traversability of Edges

Edges resemble steep slopes. Oftentimes they stretch over a wide region like e.g. a sidewalk curb does, however e.g. a small rock on otherwise flat ground has edges at its perimeter too. We consider a slope an edge if its inclination is higher than what the robot can traverse as determined in the sections before. The difference between a slope and an edge is that we do not further regard its inclination but its height instead. Regardless of the slope of an edge the robot is able to traverse it if the edge is small enough. E.g. a small

cable lying on the floor can be driven over without much hassle even though there is an abrupt height difference from the floor to the cable and thus a steep slope.

In an experiment we try to find the limit of what edge height the Jackal can traverse. Another wooden board is placed to be level and elevated compared to the floor. It can be hoisted up to different heights and a wall alongside helps with sturdiness. In the tests the Jackal slowly drives towards the edge with a speed of 0.05 $\frac{m}{s}$ to mitigate any dynamic effects. We vary the approach angle and the Jackal repeats every test, driving forwards and backwards, because the center of gravity does not coincide with the geometric center. The approach angle is taken from the axis orthogonal to the edge to the direction in which the Jackal moves, so 0° is head on and 90° would be parallel to the edge. The results of the experiment are shown in Table 3.8 and they are discussed next.

Driving down over an edge is less of a problem than driving upwards. The Jackal never failed to drive off an edge in this experiment, however this case should still be considered in general. Other robots might fall over after dropping from an edge, or they risk damaging sensors or other equipment. Depending on the task the robot is expected to perform, one could also consider limiting the height the robot is allowed to drop in order to not jerk it while it is e.g. carrying something or taking measurements.

The Jackal fails to climb an edge for different reasons, some special cases are illustrated in Figure 3.9. With most failure cases at 0° the wheels touching the edge first can climb the curb, but the hind wheels get stuck. This can be seen in Figure 3.9a. In the spot marked red the wheel cannot grip the edge enough to lift the back of the robot. Driving forwards helps as the center of gravity then places more weight on the front wheels that climb the curb more easily than the hind wheels. The frontal weight also increases the friction at the front tires that can then pull the back wheels against the edge more easily. There is a similar occurrence at the edge height of 4 cm and 60° approach angle where the Jackal passed when approaching the edge from the right but not when coming from the left side. We associate this with the center of gravity being a bit to the right of the robot's geometric center. This gives climbing the edge an advantage if the right side of Jackal climbs the edge first, just as driving forwards works better than backwards. The lateral center of gravity offset is less than the longitudinal offset, which could be why we only noticed this effect once compared the many differences in driving forwards versus backwards.

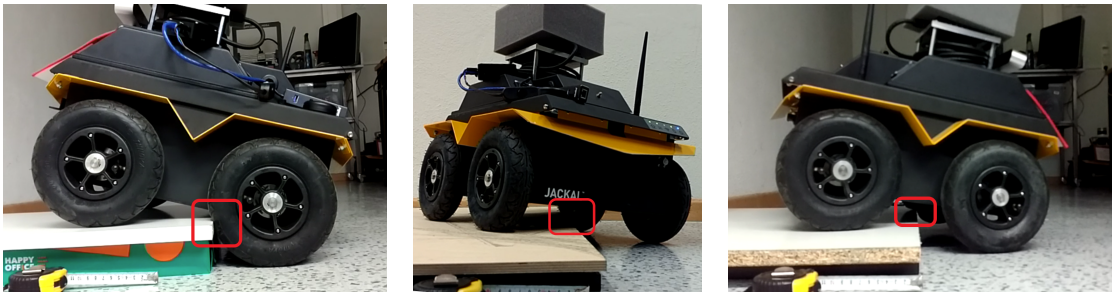
A slight approach angle can help the hind wheels overcome the edge. This can be seen when comparing the results of an approach angle of 20° to one of 0° for heights 5 cm and 5.5 cm and in Figure 3.9c where the single marked wheel manages to lift off the ground. With an angle of 45° and above one hind wheel reaches the slope before the diagonally opposing front wheel does. At sufficient height differences the front wheel then gets stuck and turns the robot to face parallel to the edge. This case is shown in Figure 3.9b with the marked wheel being dragged along the edge. Only edges with the height of 3.0 cm and below are traversed regardless of approach angle or driving direction. Hence, this is the limit when categorizing edges into traversable and non-traversable, and we refer to the value as the edge limit.

We additionally require that the edge limit is below the robot's ground clearance, otherwise the limit is replaced by it. This way the robot's body does not scrape the edge and the robot cannot get stuck on an edge that does not continue at the same height, like

Table 3.8: The effect of edge height, approach angle and driving direction on the ability of the Jackal to traverse an edge upwards.

| Height [cm] | Approach angle [deg] | Forwards | Backwards |
|---|---|---|---|
| 3 | 0 | pass[1)] | pass[1)] |
| | 30 | pass[1)] | pass[1)] |
| | 45 | pass | pass |
| | 60 | pass | pass |
| | 70 | pass | pass |
| 3.5 | 0 | pass[1)] | pass |
| | 30 | pass | pass |
| | 45 | pass[1)] | pass[1)] |
| | 60 | pass | pass |
| | 70 | fail | pass |
| 4.0 | 0 | pass[1)] | pass |
| | 30 | pass[1)] | pass[1)] |
| | 45 | pass | pass |
| | 60 | fail | passed twice |
| | 60 [3)] | fail | fail |
| 4.5 | 0 | pass | pass |
| | 30 | pass | pass |
| | 45 | pass | fail |
| | 60 | fail | fail |
| 5.0 | 0 | pass | pass[4)] |
| | 20 | pass | pass |
| | 30 | pass | pass |
| | 40 | pass | pass |
| | 45 | fail | fail |
| | 60 | fail | fail |
| 5.5 | 0 | pass | passed 3 times failed once |
| | 20 | pass | pass |
| | 45 | failed 3 times passed once | fail |
| | 60 | fail | fail |
| 6.1 | 0 | pass[4)] | fail |
| | 30 | pass[2)] | fail |
| | 45 | fail | fail |
| | 60 | fail | fail |
| 7.1 | 0 | fail | fail |
| 12.1 | 0 | fail | fail |

[1)] Result was induced from other results.
[2)] Trajectory was diverted to 0 deg.
[3)] This time coming from left side.
[4)] With some struggle. Barely passed.



(a) Edge height: 7.1 cm
Approach angle: 0°
Direction: forwards
Result: Failure

(b) Edge height: 4.5 cm
Approach angle: 60°
Direction: forwards
Result: Failure

(c) Edge height: 5.0 cm
Approach angle: 20°
Direction: backwards
Result: Success

Figure 3.9: Special cases when trying to climb an edge.

a bump. This condition is met for the 3.0 cm limit and the Jackal which has a 6.35 cm distance from its chassis to the ground.

Clearly the approach angle plays a major role in the traversability of an edge. With the modular technique in this thesis where we separate the mapping and traversability determination from the controller, we have no feedback from the controller. We cannot know how the robot will be oriented at different points in the map because the robot might have turned while getting there. The case of driving forwards or backwards can be interpreted as an approach angle above 90°. The same issue is present here as well. Without feedback from the controller, we cannot know in what direction the robot will drive at certain spots of the map. As a result we need to use the limit for the worst case approach angle even though this is restrictive. Many edges that could be traversed with the angle of how the software controlling the robot plans to approach the edge need to be considered an obstacle because during mapping we do not have knowledge of the path the robot will take.

It is possible to lessen this effect by considering the current orientation of the robot to the edges in the map. We might assume that the robot roughly maintains its orientation in its near vicinity. This means that it needs to move some distance relative to its current position in order to turn relative to its current orientation. In a small region around the robot, we could then estimate that a robot's orientation will be the same as it is at its current location. Within this region, we do not need to consider all approach angles towards an edge. We can improve our traversability analysis by taking the edge height together with the estimated approach angle into consideration. This assumption however degrades quickly with increasing distance from the robot. It is especially not suited for the Jackal as it can turn on the spot. The assumption does not hold even for edges that are directly next to the Jackal. Therefore, we do not further pursue this idea, but stick with the conservative edge limit.

## 3.4 Combining Limits

Both the slope feature and the edge feature determine traversable and non-traversable regions. While the slope is applicable for every region, the edge aspect only makes sense where the slope already is high. Additionally, the two given problem sets of regarding the slope and the presents of an edge in a region overlap. An edge that is too high and therefore untraversable would already be marked as an obstacle if one considers the steep inclination in the slope problem.

We now need a way to merge the results of both matters. To do so we make use of the subsumption architecture by Brooks [2, 3]. This architecture comes from the field of reactive mobile robotics where it is common that multiple modules exist that each represent a basic behavior like homing or wall following. These behaviors are layered and given different priorities. They run in parallel and are finally merged through an arbiter structure. Behaviors with a higher competence level can suppress or inhibit the output of behaviors at lower levels.

The subsumption architecture is useful for our case as it too decomposes the task into sub-problems. We can refer to the traversability determination at edges and slopes as behaviors that we call the edge behavior and the slope behavior respectively. Furthermore, it is a fast and reliable approach when there are few behaviors, two in our case, and it requires no state to be kept between updates. We partially implement the subsumption architecture by employing the arbitration method to combine the results of the edge and slope behaviors. Unlike in reactive robotics, the behaviors do not directly control actuators nor do they get the raw input data but a curated elevation map instead.

The feature of an edge can be seen as a special case of a steep slope. That is why we choose the slope behavior to run at a lower level. It is the default case and the edge behavior only comes into play at steep slopes. If the slope behavior does not determine a region to be untraversable, then the edge does not either because we only consider steep slopes to be an edge. The case where a region is considered to be an obstacle by the edge behavior but not by the slope behavior cannot exist. Conversely, if the edge behavior determines a region to be untraversable then the slope behavior must also have calculated an obstacle there. The difference between the behaviors becomes relevant when there is a region that is an obstacle with respect to the slope behavior, but that is traversable when it comes to the edge behavior. Therefore, the edge behavior does not need to add obstacles to the ones determined by the slope behavior. Instead, it needs to be able to nullify a region containing a small edge, that the slope behavior falsely marks as an obstacle. The subsumption architecture allows us to model this merging step with an inhibitor node. Figure 3.10 illustrates the subsumption architecture employed to merge the slope and edge behavior.

Note that the elevation map is updated regularly with new data from the environment and the filter chain updates the traversability map accordingly every time. In each update step we process the entire map that contains both traversable and untraversable regions. This means that we do not fully inhibit the output of the slope behavior if the edge behavior finds a small edge. The behaviors are merged cell-wise instead. The edge behavior unmarks specific cells but not the entire layer that the slope behavior calculated.

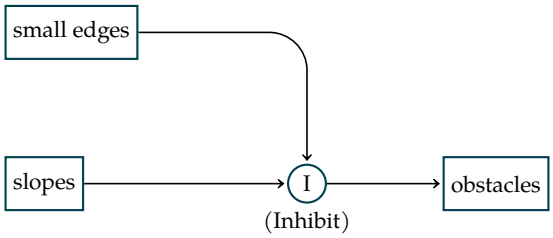



Figure 3.10: The subsumption architecture is used to merge the slope and edge behavior. The obstacles are determined from the slope by default, but this can be inhibited if the steep slope comes from only a small, traversable edge.

If the height map is blurred or recorded with reduced resolution, then such a small edge would only be represented by a slight incline that would not even trigger the region to be marked an obstacle by the slope behavior. Therefore, the effect of the edge behavior that inhibits obstacles for small height differences could also be achieved by reducing the resolution of the map or by calculating the slope from a larger patch of the map. However, the approach of having an extra edge behavior leaves the possibility for further research open. With information where edges lie one could find out how they are oriented. This can be used to refine the edge limit by regarding the approach angle.

# 4

# Realization

We now have introduced all the concepts necessary to know what situations are a hazard to a robot and how to combine the results of different behaviors. What is left to do is to detect the different hazards from the elevation data in the grid map. In this Chapter we implement the theory presented and create the process from start to finish. The regions of the map need to be categorized into traversable and non-traversable parts separately by the edge and the slope behavior, before the output of both behaviors is merged with the help of the subsumption architecture.

The process consists of two steps. There is a compilation step to initialize any values and setup files for the second part. In this first step we recognize the formulas derived in the 3rd Chapter. We use these formulas to compile values for the limits of what can be traversed. After that, the runtime step performs the work of converting the elevation data into the output map. It receives new information about the environment and publishes periodic updates on the traversability. This step is where the edge and the slope behavior become active and apply the compiled limits.

Afterwards we discuss the drawback of the proposed realization being restrictive due to conservative assumptions. We present alternative methods to mitigate this issue.

## 4.1 Implementing the Compilation Step

The first step is executed at compile time and only needed once for each robot specification. We separate this part from the second step, that must run in real time, to reduce the computation power needed. It also reduces the complexity of the run time step which allows us to implement it as a filter chain. During the compilation part we calculate traversability limits from information about how the robot is built. The procedure is illustrated in Figure 4.1.

The models in the Figure are implemented as a python script called `compile_robot_limitations.py`. It requires the configuration file `jackal_properties.yaml` which contains properties of the robot that are relevant to its traversability. These include the center of gravity, the maximal velocity and the maximal acceleration in each direction as well as the basic geometry of the robot and its motor power or wheel torque. This is also the place where we can input the result of the experiment from Section 3.3 that we performed to


Figure 4.1: The compilation step calculates traversability limits from the properties of the robot.

find the traversability limit of an edge. Even though not relevant for the Jackal, the inclination limit corresponding to the slippage, that we tried to find in Section 3.2, can be entered here as well. The code still includes the slippage for the sake of being able to apply the work to robots other than the Jackal. As stated before, we do not consider the slippage in the case of the Jackal which is why we enter a value of 90°. This effectively removes the slippage from further considerations because all slopes are below that threshold.

The python script reads the configuration file and applies the calculations mentioned in Chapter 3 to generate the limits of what slopes and what edges are traversable. For the case of the maximal safe inclination the limits due to tipping, slippage, and lack of wheel torque are regarded separately. Each produce a limit on the traversable inclination. It is dangerous for the robot to exceed even a single limit, so the final slope limit is the minimum of these limits for specific failure cases at a slope. That way we cover all of them.

The limit of the slippage is taken directly from the YAML file without further calculations. The calculation for the case of the robot tipping over is more extensive and this is the critical factor for the Jackal. Inside the function `max_inclination` the calculation for the case of tipping around an edge of the robot's body is repeated for all the four directions that the robot could tip to. The acceleration $a$ that is used in formula (3.2) is the maximum of the acceleration that the robot can achieve by turning its wheels and the maximal centripetal acceleration in that same direction. Given the desired direction and the robot configuration, the function `max_dynamic_accel` maximizes the term (3.5e) and determines the dynamic acceleration that the robot is exerted to when performing a tight turn at its highest velocity. It requires the maximal velocity in a direction perpendicular to the desired acceleration as well as the position of the wheels and the center of gravity. When calculating the limit due to the wheel torque from formula (3.6c), we provide two ways of adding the information in the YAML file. One can either specify the total wheel torque or the motor power. If the power is given, then the script automatically calculates the maximal torque using formula (3.7c) with the power, maximal velocity and the wheel radius. All of this happens in a function called `max_incline_torque`.

The limit for the edge behavior again is taken directly from the YAML file. No calculations are needed here, the value is the one we found during the edge experiments in part 3.3.

Finally, the function `write_limits_filter_chain` is called that creates the filter chain for the next step. It loads a filter chain template from the file `myGridmapFilters_postprocessing_template.yaml`. Then it replaces placeholders with the limits that were compiled and saves the result to `myGridmapFilters_postprocessing.yaml`.

During this step, the script prints out useful information about the stability of the robot. The script informs the user about whether the threat of tipping over, slippage or

the lack of motor power is the most restrictive for the highest safely traversable inclination. It also lets the user know what side the robot is most likely to tip over to. If no torque is provided, then the user is told what its value is as calculated from the drive's wattage. The centripetal acceleration from the calculations is presented along with the radius of the worst case trajectory. This brief analysis can be used to improve the robot's design in terms of traversability. It hints to what change in center of gravity or stance is needed the most to improve the stability. One could also limit the speed, acceleration or turning radius, but hindering the robot like this might not be necessary as is discussed in Section 4.3.

## 4.2 Implementing the Runtime Step

The second step in the process happens during runtime. Here we get live data as a grid map from the `elevation_mapping` package and use a filter chain to convert the `elevation` layer into the traversability output. The procedure is illustrated in Figure 4.2.

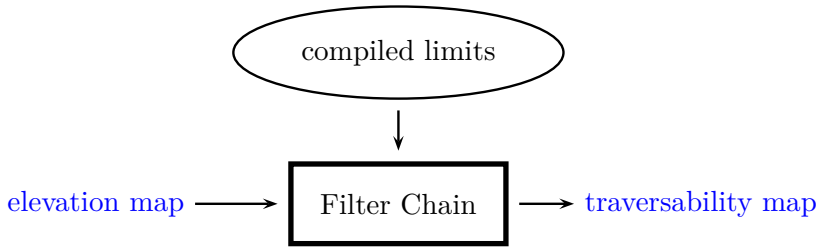



Figure 4.2: The runtime step applies the traversability limits that were pre-compiled to the elevation data. The output is a map showing what regions are traversable. Both the elevation map and traversability map are illustrated in blue to show that they contain live data.

The filter chain is illustrated in Figure 4.3. Note that this Figure is slightly simplified for visualization purposes. The implementation of each `ThresholdFilter` actually consists of a `DuplicationFilter`, that creates a new layer from the input layer by duplicating it, and one `ThresholdFilter`, that sets every value below a threshold to a constant, and another that sets all the values above that threshold. Additionally, the `SlidingWindowMathExpressionFilter` requires a `BufferNormalizerFilter` in front of it, due to the underlying use of a circular buffer.

In the following we also add images when explaining the corresponding layers that are created by the filter chain. Unless stated otherwise the layers in these images are taken from the data of our evaluation experiment in Chapter 5 where the scene is elaborated on. In the images of the grid map layers there is a Jackal for reference of scale. Note that the map stays centered on the robot during the experiment. The images here however are taken from a saved map in which the simulated Jackal can move around freely without affecting it. The elevation data dictates the shape of each of the surface plots, they differ in color which is set from data of individual layers of the grid map. A color bar defines the values the colors represent in each image.

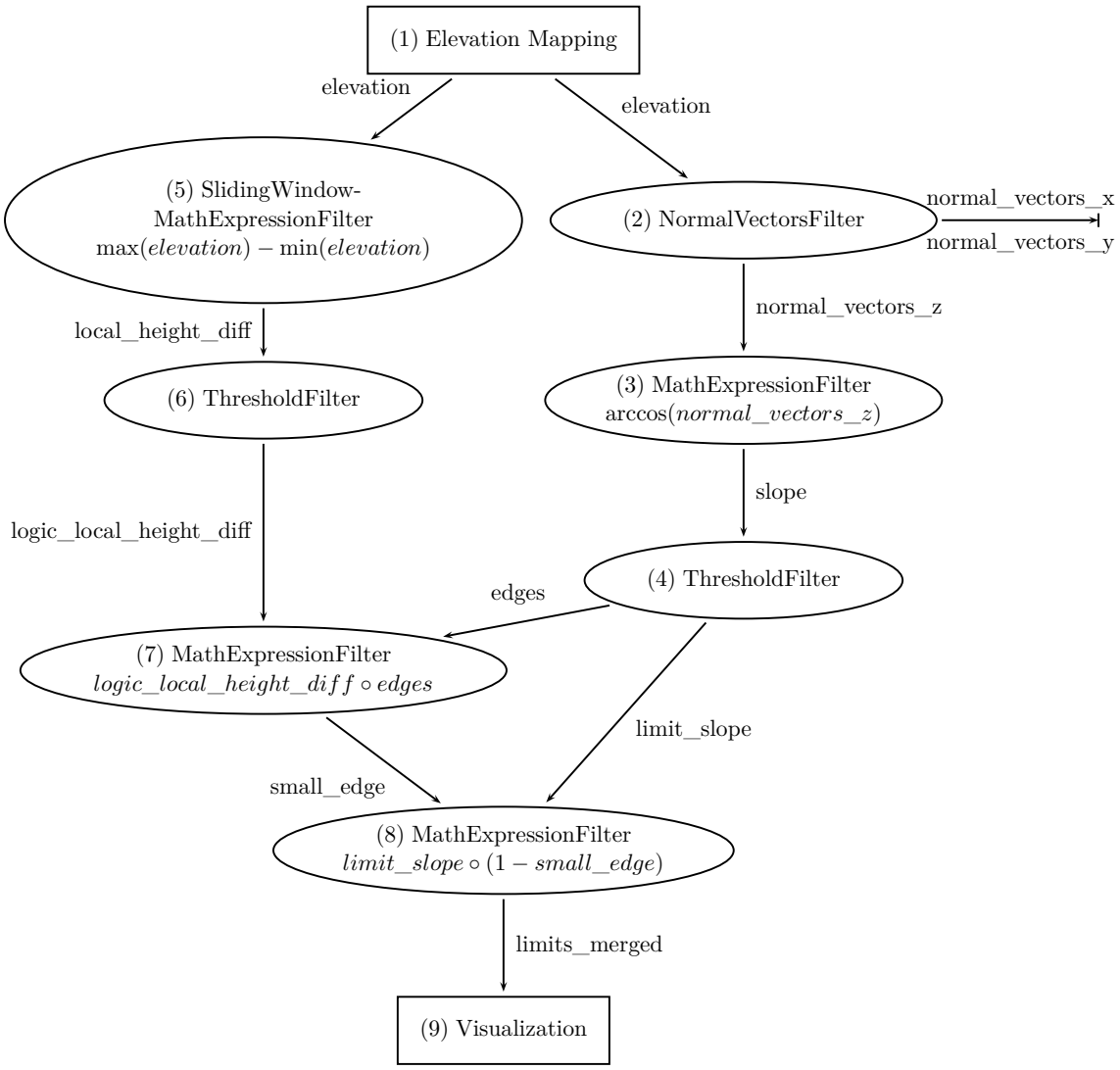



Figure 4.3: The filter chain which implements the run time step. Boxes represent nodes interacting with the filter chain. Ellipses are Filters and edges the data layers of the grid map.

### Slope behavior inside filter chain

In the beginning the slope and the edge behaviors emerge and both get the elevation data from the `elevation_mapping` package 4.3(1). We first discuss the slope behavior. This behavior must first extract the slope from the elevation data and then apply the limit we calculated in the first step.

A `NormalVectorsFilter` 4.3(2) is applied, and as the name implies it generates vectors that are normal to the surface formed by the `elevation` layer. Each cell then has a corresponding normalized 3-dimensional vector so 3 values need to be saved for each cell. Therefore, the filter populates 3 new layers: `normal_vectors_x`, `normal_vectors_y` and `normal_vectors_z` that each hold one of the vectors components. We only use the $z$-component, shown in Figure 4.4, to obtain the angle of the slope, the other layers are byproducts.

The angle of the slope can then be calculated with the `MathExpressionFilter` 4.3(3). This filter allows us to specify a math formula that is then performed element wise on the input layer. If we imagine the right triangle in Figure [TODO MAKE FIGURE], the output of the `MathExpressionFilter`, the `slope` layer depicted in Figure 4.5, is then equal to $\arccos(\mathit{normal\_vectors\_z})$.

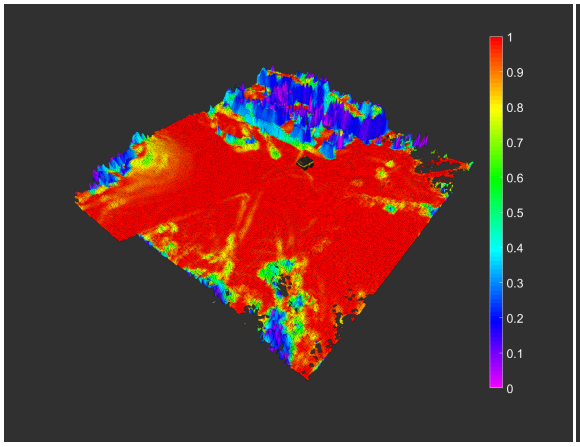



Figure 4.4: The `normal_vector_z` layer. The vector is normalized and the $z$-component is in the range of 0 to 1.

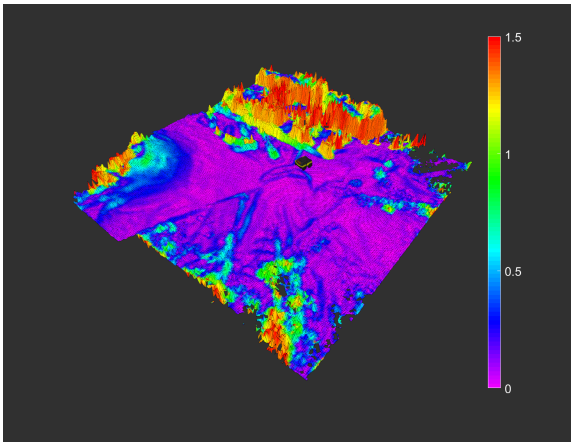



Figure 4.5: The `slope` layer. The slope is given in radians.

We then threshold the `slope` layer with the inclination limit that was compiled before with the filter 4.3(4). Every value above the threshold is set to 1 and every one below is set to 0. The resulting `limit_slope` layer concludes the slope behavior and can be seen in Figure 4.6. All regions that contain a 1 are marked untraversable and vice versa. This layer also doubles as the `edges` layer because we consider a region an edge if the slope there is too high to be traversed.

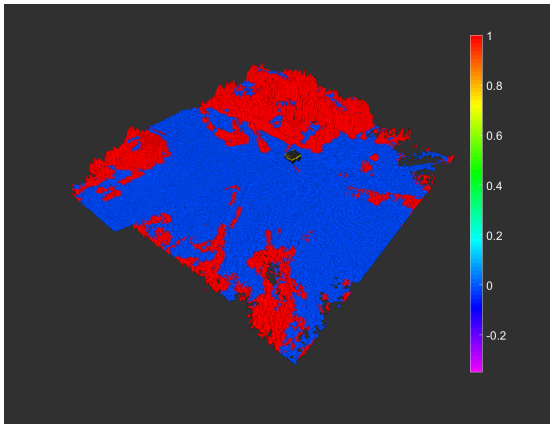



Figure 4.6: The `limit_slope` and `edges` layer.
A cell either contains a 0 (blue) or a 1 (red).

### Edge behavior inside filter chain

The edge behavior must find the height of edges and classify them as being small or large. To get the height of an edge we need the difference in elevation of a point at the bottom and a point at the top of a potential edge. This is implemented with the help of the `SlidingWindowMathExpressionFilter` 4.3(5). This filter performs a convolution on the `elevation` layer. Instead of a constant kernel we subtract the minimal from the maximal value within a quadratic window. The result is written to the center cell of the window in the output layer and the window then slides over to the next cell. The resulting layer is called `local_height_diff` and each cell contains the largest height difference found in a local region around that cell. This can be seen in Figure 4.8 where the local regions are apparent as square patches.

We choose the window to be 37.559 cm in size, the greatest side length of the Jackal's footprint as determined by the supporting polygon [16]. A window too large would reduce the accuracy of the height of the actual edge as it includes distant cells. In our edge experiments the Jackal is only supposed to climb a single edge. With a window smaller than the Jackal's footprint we risk missing an edge outside the window that the Jackal still has to traverse. Also, the window size $c$ must be chosen such that the edge behavior is not triggered at slopes at the inclination limit $\alpha_{lim}$. The situation is illustrated in Figure 4.7. With a window size of $c_{err} < c_{lim}$, the edge behavior would nullify the slope. A window size above $c_{lim}$ keeps the behavior from detecting false positives. We must set $c$ according to equation (4.1) to ensure that the height difference due to the slope is larger than the edge limit $e_{lim}$. Otherwise, the edge behavior would find traversable edges everywhere along an untraversable slope. The choice above of using the footprint size satisfies this constraint.

$$c > c_{lim} = \frac{e_{lim}}{\tan \alpha_{lim}} \tag{4.1}$$

We classify small and large edges from the `local_height_diff` layer by simply thresholding it in 4.3(6). The limit is the maximal traversable edge height that we determined

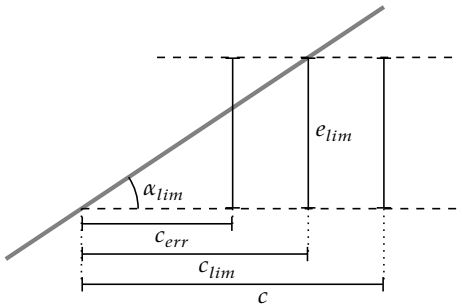



Figure 4.7: Side view of a slope with an inclination of the slope limit $\alpha_{lim}$.

earlier, and the output is saved in the `logic_local_height_diff` layer as in Figure 4.9. In accordance with the subsumption architecture we want the output to be active when the slope behavior is supposed to be inhibited. We choose to set the cells below the threshold to 1 representing a small edge that can be traversed even though the slope behavior considers it an obstacle. The cells above the threshold are set to 0 representing a large and untraversable edge.

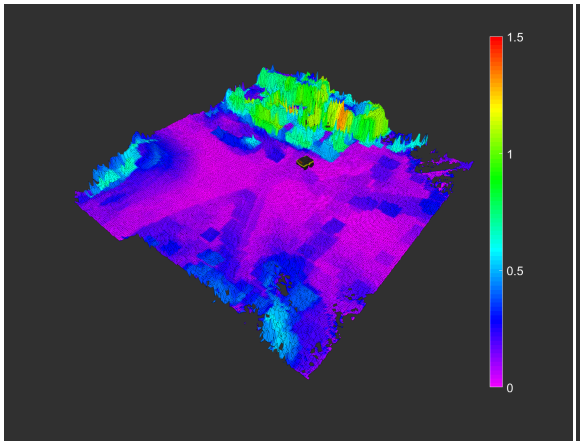



Figure 4.8: `local_height_diff` layer
The height difference is given in meters and calculated for a square of length 37.559 $cm$.

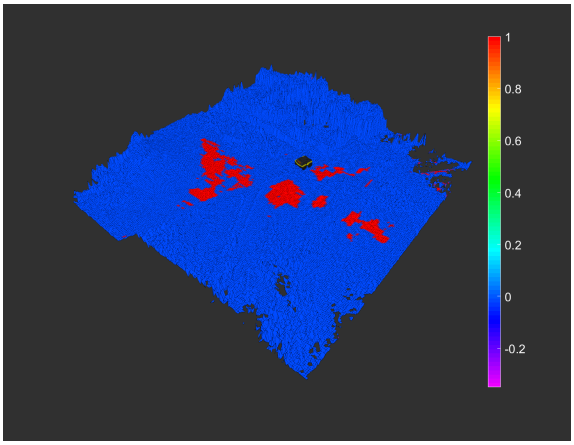
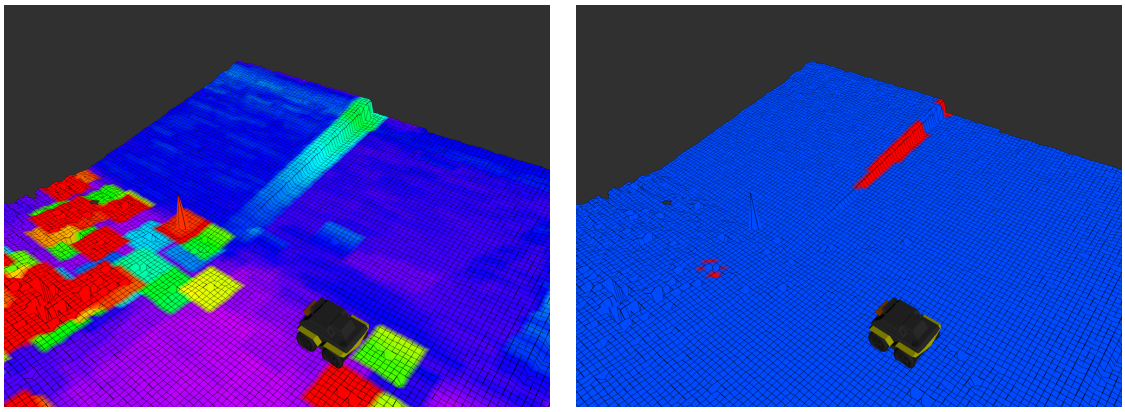



Figure 4.9: `logic_local_height_diff` layer
A cell either contains a 0 (blue) for a height difference larger than the edge limit or a 1 (red) for a small difference.

Through the height difference alone we already get a good approximation of where the edges lie and by thresholding which ones can be traversed. However, it can be noticed that the frontiers of all the edges are shifted by about half the patch size to occupy regions that do not contain edges. This is caused by the sliding window of the convolution that has to extend over multiple cells. Because of the width of the window, a window centered on a cell, that lies close to an edge but is not part of it, might just reach into the edge. The center cell then gets a value as if it were part of the edge.

We counteract this effect by using our initial definition of an edge: The slope steepness and not the height difference. The `edges` layer that is a byproduct of the slope behavior

already contains this information, so we do not have to calculate it again. It contains a 1 where there is and a 0 where there is no edge. We can take advantage of this and simply multiply this layer element wise with the `logic_local_height_diff` layer by employing the `MathExpressionFilter` 4.3(7). All regions where edges extend too far get zeroed out and the rest does not get changed. The result is called the `small_edge` layer because it holds a 1 where a traversable edge is and a 0 otherwise. The map used to visualize previous layers does not contain many small edges. In Figure 4.10 we instead show grid map layers from a simulated world that has an edge between two ramps of different inclination. We also configure the filter chain for an imaginary robot that has an edge limit of 25 *cm* to better visualize small edges. The `small_edge` layer can be seen in 4.10(a) and the `local_height_diff` layer of that map is shown as reference in Figure 4.10(b). The region in the bottom left of the images is level ground and is connected to two ramps towards the top right. There are a lot of unwanted sensor errors in the level region. These result in patches with large height differences. Section 5.2 mentions such errors in greater detail.



(a) `local_height_diff` layer
The height difference is given in meters and calculated for a square of length 37.559 *cm*.

(b) `small_edge` layer
A cell either contains a 0 (blue) for a height difference larger than the edge limit or a 1 (red) for a smaller difference.

Figure 4.10: A scene with 2 ramps and an imaginary robot with an edge limit of 25 *cm*.

### Merging and converting output of the behaviors

In the end of the filter chain the final layers of both behaviors are merged. The arbiter of the subsumption architecture is implemented with another `MathExpressionFilter` and the expression $limit\_slope \circ (1 - small\_edge)$ as seen in 4.3(8). This multiplies a cell from the `limit_slope` layer with 0 if there is a small edge at that position, hence inhibiting the output of the slope behavior for that cell. Otherwise, the factor is 1 and the edge behavior does not intervene but lets the output of the `slope` layer through unhindered. The arbiter saves the result to the `limits_merged` layer that now contains the traversability as a value of 1 or 0. This layer is illustrated in Figure 4.11, and we switch back to the scene from before.

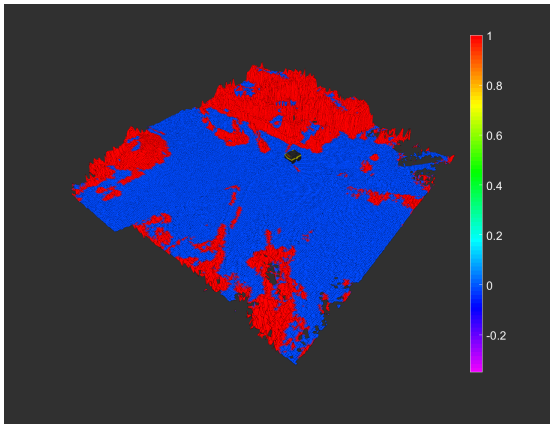


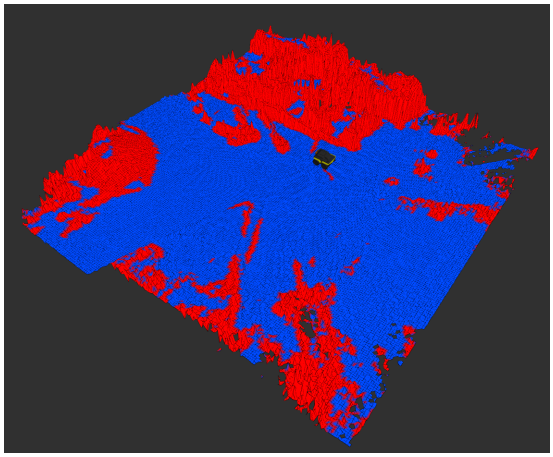

Figure 4.11: The `limits_merged` layer.
A cell either contains a 0 (blue) or a 1 (red).

Finally, we use the `grid_map_visualization` package under number 4.3(9) to convert the `limits_merged` layer to an occupancy_grid. The regions marked with a 1 are transformed to be obstacles and the regions containing only 0 valued cells are marked free. Now that we have completed the calculations and saved the result as a widespread format, many third party tools are able to make use of this map customized to the specific robot.

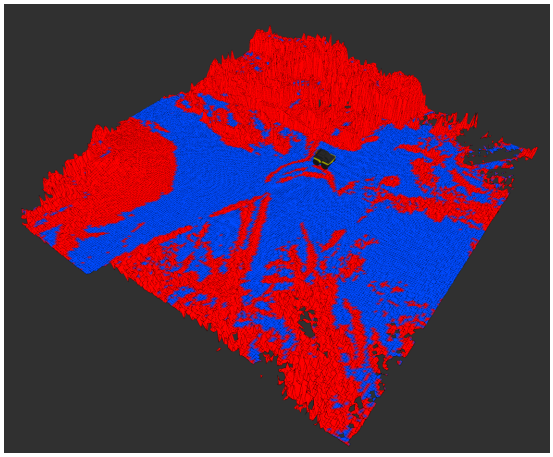
## 4.3 Alternative of Outputting Control Limits Instead of Obstacles

The process introduced so far is based on the principle of always assuming the worst case scenario. The map we generate is reduced to contain the simple information of a region being traversable or not in a widespread format. This however results in the calculated obstacles being exaggerated. The map generated can be restrictive up to the point that little traversable space is left to drive around. Actual obstacles then loose importance and are overshadowed by the possibility of bad behavior by the controller. This mapping approach then becomes unpractical because the map produced becomes too restrictive when the worst control is assumed. Figure 4.12 compares maps of the same environment with the only difference being the maximal velocity the robot is capable of. It shows how restrictive the map becomes. The maps are depicted for fictional robots that are built like the Jackal but have higher top velocities up to 3.1 $\frac{\mathrm{m}}{\mathrm{s}}$. If we assume a tight turn at an even higher velocity, something a badly designed controller might do, then a robot this capable would tip over regardless of the environment. We have to mark regions as obstacles even though they could be traversed if the robot were to move correctly.

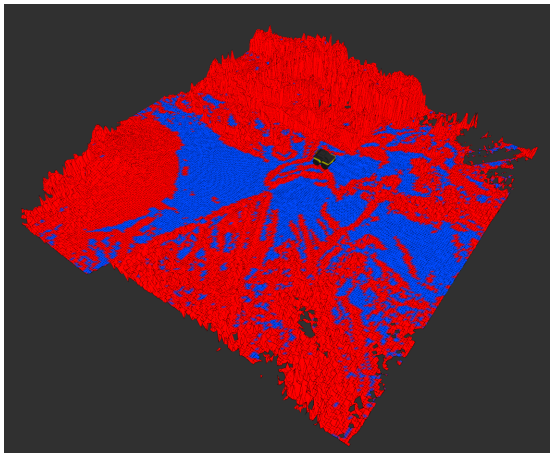
A simple way to mitigate this issue is by reducing the capability of the robot in software. Most controllers accept global limits that cap the speed, acceleration etc. These limits can be reduced to force safe movement of the robot even on the worst trajectory, thus allowing more regions on the map to be traversed. The drawback with this workaround is that the capability of the robot is reduced only to keep up the assumption of worst control. The robot is forced to move carefully globally, even where it is not necessary.



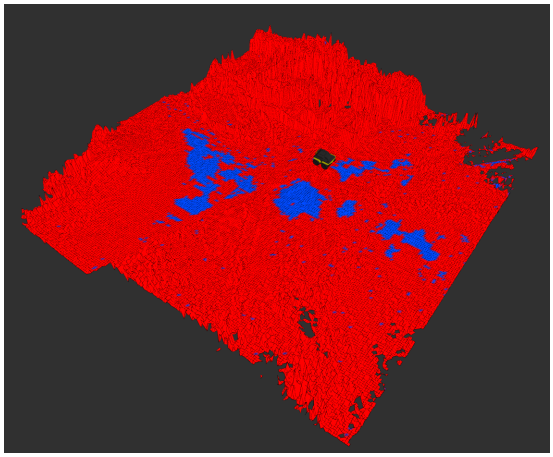
(a) The maximal velocity is 2.5 $\frac{\text{m}}{\text{s}}$.
The slope limit is 19.2°.



(b) The maximal velocity is 2.9 $\frac{\text{m}}{\text{s}}$.
The slope limit is 7.9°.



(c) The maximal velocity is 3.0 $\frac{\text{m}}{\text{s}}$.
The slope limit is 4.6°.



(d) The maximal velocity is 3.1 $\frac{\text{m}}{\text{s}}$.
The slope limit is 0.9°.

Figure 4.12: The `limits_merged` layer is depicted for imaginary robots that differ in their top speed. The map becomes more restrictive as the robots become capable of higher velocities. Blue cells are marked as traversable and red cells are marked as obstacles.

In this part we propose another method to counter the tendency of the map to become too restrictive. Having to take the worst possible controller into account is the cause of this issue. We now change our assumptions about the controller that uses the map we create. There still is no feedback from the controller, however we now require the controller to obey certain non-fixed control limits that we provide. Instead of assuming the worst control, our process can limit the control as needed. By imposing limits on the movement of the robot we can get it to move safely where it is necessary, and we can lift these regulations where we do not expect the robot to be in danger. This procedure is illustrated in Figure 4.13. The non-conservative limits are generated by a modified compilation step that now assumes the best instead of the worst control because we influence the control output. Additionally, the modified filter chain uses the robot properties to evaluate the necessary control limits.

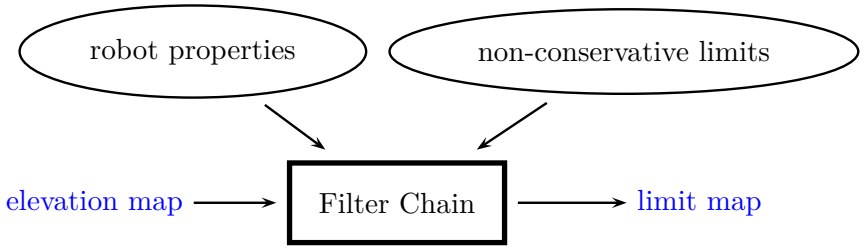



Figure 4.13: The approach of creating limits for the control output to the robot's drive. The modified filter chain requires properties of the robot and non-conservative limits that were pre-compiled assuming best control. The elevation map and the limit map contain live data and are shown in blue.

Note that these output limits only depend on the slope in the environment as we do not link the traversability of an edge to speed or acceleration. We can formulate an acceleration limit by rearranging the formula (3.2) to solve for $a$ as in equation (4.2a). If we rearrange formula (3.2) assuming maximal acceleration, then we can get the limits of the height of the center of gravity as equation (4.2b). Or we can solve for $b$ using equation (4.2c) to get the length between the center of gravity and the tipping point from Figure 3.1. These are irrelevant to the Jackal as it cannot influence these parameters. The mobile robot with manipulator arm however is able to change its center of gravity. With minor adaptions other types of robots may therefore choose to comply to these limits instead of the acceleration. We keep our focus on the acceleration limit.

$$a = \frac{gb}{h} \sin(\gamma - \alpha) \tag{4.2a}$$

$$h = \frac{gb}{a} \sin(\gamma - \alpha) \tag{4.2b}$$

$$b = \frac{ah}{g \sin(\gamma - \alpha)} \tag{4.2c}$$

The new map only marks obstacles where the robot can not possibly traverse, regardless of its behavior. We define such obstacles as fundamental obstacles. These regions are unsafe even if the robot is not subject to any movement, or for other robots if they are already configure to the safest $h$ and $b$ values. The other regions without fundamental obstacles are traversable as long as the limits are obeyed. Regions that were considered obstacles before now become accessible and the map becomes less restrictive. By doing so we are able to create a mapping system that is not only specific to the particular robot but also to much of its behavior.

From here on forth we refer to controllers that cannot obey varying control limits as simple controllers, in order to distinguish them from the other controllers introduced in this Chapter. A controller that is able to periodically receive these control limits and to obey them is called a moderate controller. Soon we also get to know another type that we name advanced controllers.

If the controller only limits its current output based on its current location in the map, we already achieve our goal of making the map less restrictive whilst still being customized to the specific robot configuration. This sort of moderate controller can easily be produced from a simple controller by overwriting its control outputs. We can create the node `control_output_cap.py` to look up the control limit that should be active currently from our map and to then make sure that the control output does not exceed these thresholds. The velocity and timestamp from the previous command are kept in order to determine the next velocity limit by integrating with the given acceleration limit. The limits coming from the grid map only affect the robot's $x$- or $y$-axis, because these are the directions it would tip towards. If one of the coordinates of the velocity command exceeds a limit, our script scales down the robot's linear and angular velocity uniformly as a means to not affect the direction of movement. When no limit is reached, the node passes the incoming commands on without change. The control output of the simple controller is rerouted through this node by remapping the ROS topic of the output, making the simple controller a moderate controller. With this combination, the robot would plan a path around the fundamental obstacles and afterwards adjust its control output to safely traverse the path. The limits generated from the formulas and experiments represent the most critical limits. In practice the node should not bound the control output to be just below these limits. Instead, we add a safety margin reduce risk and to account for any unmodelled effects.

The moderate controller does not know of the control limits it will be subjected to beforehand. The path it chooses around the fundamental obstacles probably is not optimal. For example, it could plan a short path that comes close to a fundamental obstacle, without knowing that it might have to slow down considerably in its vicinity due to a steady increase in inclination. A more advanced controller would ideally also use the information about all control limits in the map for its path planning. With this knowledge, it can optimize a trajectory including the motion of the robot instead of just planning a path. The resulting trajectory of the advanced controller may not be the shortest, but it can be faster because the control limits are considered. A simple controller would need more in depth modification to become an advanced controller. It would also be tied more closely to our mapping approach. Therefore, such a controller is less suited for the modular world of ROS.

Even this approach of the advanced controller is a bit restrictive. The orientation of obstacles relative to the robot is not accounted for. We still need to be conservative about the approach angle of an edge and the tipping direction. Unfavorable trajectories like a sharp turn are not eliminated either.

# 5

# Evaluation

In this Chapter we compare our approaches to each other as well as to a standard mapping method using the `slam_gmapping` package. We explain the setup of the scene and address errors in the map data. We demonstrate the shortcomings of the standard approach when it comes to rough terrain, and we show that our approach becomes less restrictive when combined with a moderate controller.

## 5.1 Setup of the Experiment

For a concluding experiment we take the Jackal to a skate park. We test our method there, because we have all kinds of slopes at our disposal in a small space, such that we can fit many different situations in our local map.

### Scene setup

Figure 5.1b shows an image of the scene. There is an island at position (1) of our map that has an elevation that is about 72 cm higher than the surrounding. The slope at its border starts out at 0° and increases towards the center to a maximum of around 40°. There also is a double ramp with a railing at position (2). We place a rucksack, folders, a cardboard box and a bottle in the scene as further obstacles. The rucksack (3) should be an obstacle for the Jackal. There is a small folder that has a maximal height of 2.5 cm and that should not pose an obstacle to the Jackal. It is positioned in (4) and happened to be just outside the mapped area. Then there is a larger similar folder with an edge of 5.5 cm just at the edge of the map in (5). In our experiments we determined this edge to not be traversable. We place a thin cardboard box that is only 2.5 cm high, a height the Jackal can easily pass. This is marked as (6). We weighed the box down with a bottle, which the Jackal should not drive over. The Jackal could actually move or crush many of these obstacles. But it does not understand the materials of the objects in the scene. The Jackal regards all surfaces in the world equally and as rigid and sturdy. This could be a field of further research. For example the robot could traverse high grass that visually seems to be an obstacle to the cameras we use.

The Jackal is mainly driven around manually to capture the scene. We focus on one

side of the island and our additional brought obstacles. The Jackal did not inspect the double ramp up close. Some portions are driven by control software that uses the different maps we create. In those cases we provide a target position and the robot attempts to find a path and to follow it to the goal. These are short duration movements during which the mostly already mapped scene is updated with new measurements.

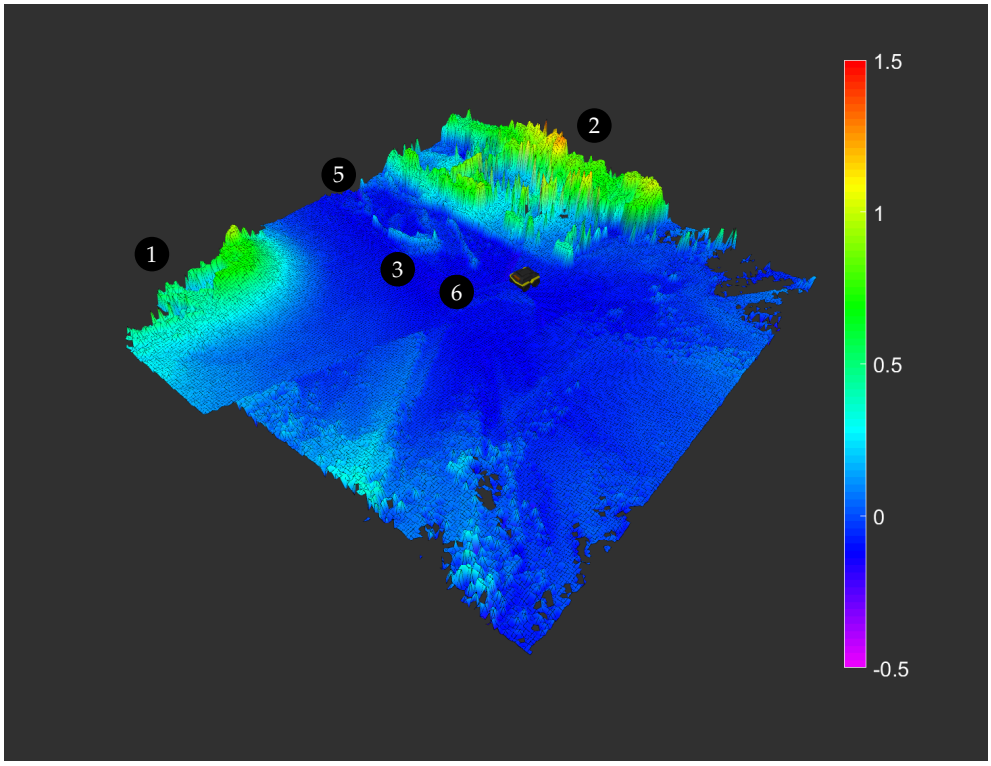
The result of mapping the elevation is illustrated in Figure 5.1a with the objects marked as before. The map is a square of 10 by 10 meters. Both the worst control and the control limiting approach are illustrated from the same height data so that they can be compared well. Other grid map layers of this experiment are depicted in Section 4.2 where the creation of those layers is presented.

### Software setup

There are three different software setups that we experiment with. We use our conservative mapping method from our first approach and feed the map to a package called `move_base`. The package has the ability to process sensor information and to register and clear obstacles in its own maps. We deactivate these options that would influence the map and rely solely on information from our map. The utility contains a global and a local path planner. We can send a goal position to the package and the global planner plots a path avoiding the obstacles in our map. It enlarges the obstacles to account for the robot's dimensions. The local planner receives this path and the odometry from the ZED camera. It then sends velocity commands to the Jackal in an effort to follow the global path as closely as possible. `move_base` additionally includes recovery behaviors that take effect if the robot appears to be stuck. In our case this happens when the `elevation_mapping` package updates the map and sensor errors appear close to the robot. The recovery behavior clears some obstacles from the maps on which the local and global planner operate and it rotates the robot in place to update the map with new sensor information. In our configuration the recovery attempts are not helpful. Any sensor errors within the `elevation_mapping` cannot be cleared from within `move_base` and fast rotations only seem to produce more sensor errors.

For a second test we configure a moderate controller for the control limiting approach. `move_base` is set up as before, but we intercept its velocity commands and route them through the `control_output_cap.py` node which in turn passes the now limited control commands to the Jackal's drive.

In order to compare our method we also perform a standard mapping using the `slam_gmapping` package [10, 11]. This common software only handles 2-dimensional data. We still use the same ZED camera, but we need to compress the pointcloud using the `pointcloud_to_laserscan` node. This tool does just as the name implies and projects the measurements of our sensor into a plane that we choose to be 36.35 cm above the ground. The resulting map is shown in Figure 5.2(a). We did reposition the rucksack and other obstacles slightly as shown in Figure 5.2(b). The map focuses more on the island and other structures of the skate park are not in range.




(a) `elevation` layer
The elevation is given in meters.


(b) The skate park scene with obstacles set up

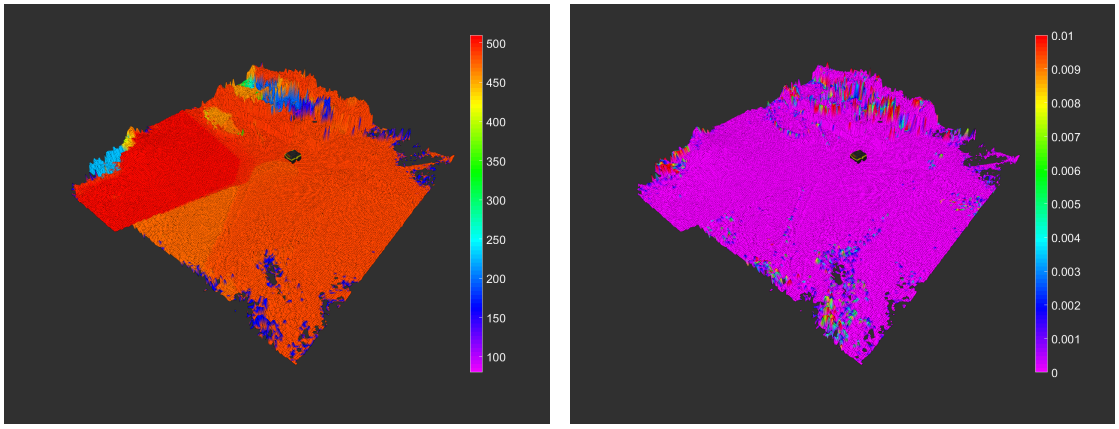
Figure 5.1: The resulting elevation in the grid map (a) next to an image of the real environment (b). We refer to (1) as the island and (2) is a double ramp. There is a rucksack at position (3), objects (4) and (5) are folders and (6) is a thin cardboard box with a bottle on top.

(a) `slam_gmapping` occupancy grid (b) Obstacle setup

Figure 5.2: The resulting map of the `slam_gmapping` package (a) next to an image of the real environment (b). Black cells in the map represent obstacles, white cells free space and gray areas are unknown.

## 5.2 Discussion of the Results

There are quite a lot of errors in the elevation map. The double ramp is covered only very briefly with the robot's sensor and from a far distance. The island is the focus of the mapping, so we expect spotty data for the double ramp. Additionally, we anticipate the railing made of small diameter metal bars to be hard to detect. There are further errors in the distance to the bottom and right side and also some errors on the inside of where the island should be. We also noticed that sudden moves of the robot falsely create a height difference at the border of the camera's field of view. Figure 5.3 depicts the `time` and `variance` layer that the `elevation_mapping` package generates besides the elevation we use. The `time` layer in Figure 5.3(a) holds the information about how many seconds into the mapping the last update was made for a given cell. The larger the seconds value is, the more recent the area was covered by the sensor. The quality of the map correlates with how thoroughly measurements are taken. We see that regions with many sensor errors were not covered in the last minutes. On the other hand, the surface of the island with little errors has recent updates as we focused our mapping there. The `variance` layer in Figure 5.3(b) contains the variance for the elevation data. Regions with many sensor errors have a higher uncertainty, which is expected. The environment is mostly made from plain concrete. Better textured surfaces could mitigate some of the issues we faced. The better option would be to add filters to smooth the data and to infill missing parts based on surrounding cells. At the moment no post-processing is done to the elevation data besides our traversability calculations.




(a) The `time` layer containing the time of the last update with sensor information in seconds since start of the mapping.

(b) The `variance` layer containing the variance of the elevation data.

Figure 5.3: The thoroughness of the sensor coverage and variance correlates to the amount of sensor errors.

### Comparing the conservative to the control limiting approach

When comparing the results of our first method of assuming the worst control with our method of limiting the control output, one can see the difference in obstacles. The occupancy grid of obstacles from the first method is shown in Figure 5.4 next to the occupancy grid of the fundamental obstacles. If we only regard the island region of the both maps, we see a clear increase in free space. The island obstacle consists of 1728 occupied cells for the case of worst control while there are only 990 such cells in the same region for the fundamental obstacle. Therefore, the fundamental obstacle takes up only 57.29% of the space that the obstacle would otherwise, leaving more room to maneuver the robot.

Figure 5.5 shows the control limits that are needed to keep the robot safe in the respective parts of the map. Note that the fundamental obstacles are specified separately in the occupancy grid in Figure 5.4(b), they are not excluded in the grid map, and those regions have unrealistic limits. For example the absolute of a negative acceleration would be how fast the robot has to accelerate downhill to not tip over. Figure 5.5(a) shows these acceleration limits that we subject the Jackal to. Figures 5.5(b) and 5.5(c) respectively depict the limits of the height of the center of gravity and the limits of the length $b$ between the center of gravity and the tipping point. The Jackal cannot utilize these limits and we do not have access to robots that could.

### Comparison to `slam_gmapping` method

We now compare our approaches to the standard `slam_gmapping` method. It suffers from sensor errors similarly to our method. The rucksack is picked up well, but there are also false obstacles present throughout the map. The occupancy grid created with the standard method does not closely resemble the maps created by our methods if we look at the island part that is present in all maps. Parts of the island are missing and it is di-

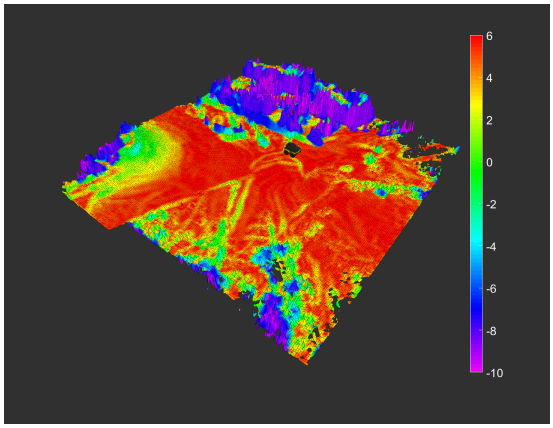
(a) Occupancy grid of conservative obstacles

(b) Occupancy grid of fundamental obstacles
Conservative obstacles are overlayed for reference.

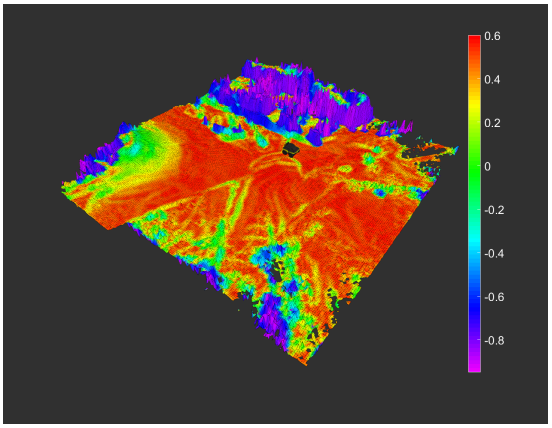
Figure 5.4: Comparing the occupancy grids of conservative obstacles and fundamental obstacles

vided into parts. This could be caused by sensor errors because the range of the sensor data that is used does cover the part of the island where the robot is in Figure 5.6b. Then again `slam_gmapping` assumes 2-dimensional ground and as the robot tilts upwards on the slope so does the plane in which the data points lie. This could interfere with the localization part of `slam_gmapping`. The robot would pick up the slope as an obstacle from far away because of its height but as it gets closer and starts to tilt upwards, the sensor picks up a part of the slope that is higher up and further away than the portion of the slope it saw earlier. Figure 5.6a shows the Jackal navigating to a goal position next to the island while the map is being build by `slam_gmapping`. The map has no obstacles registered between the robot's start and goal position so the path is a straight line. This path however takes the robot up near the top of the island. We need to halt the Jackal manually to prevent it from tipping over.

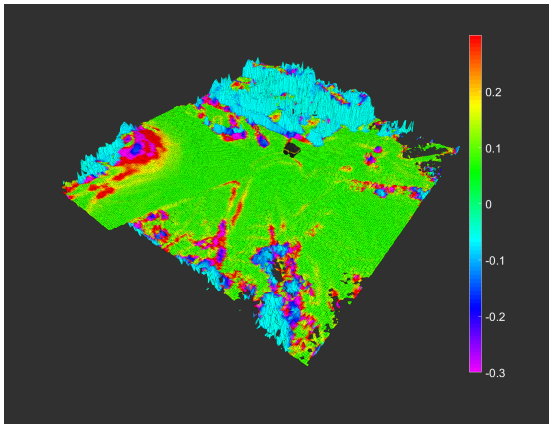
We conclude that this standard method is not an ideal choice in uneven terrain. Other methods that are able to process 3-dimensional information should be at an advantage. However, such methods still lack the understanding of what the robot can traverse based on its design. In contrast, the approach proposed in this thesis uses the sensor data to build a more realistic height map. The post-processing extracts drivable regions. If these are too restrictive a moderate controller can be employed which uses the less restrictive fundamental obstacle representation. Figure 5.7 shows the path generated by our conservative method. In the map the Jackal is just left of the island, and it is given a goal position further around the island. Both the start and goal position are similar to the ones from Figure 5.6a. This map differs from the ones presented above as it is taken at a different point in time. The map is shifted more towards the island and the map does not include the obstacles we place. The sensor errors differ as well. Instead of moving close to the




(a) Limiting the robot's maximal acceleration in $\frac{m}{s^2}$




(b) Limiting the height of the center of gravity in meters




(c) Limiting the length $b$ in meters between the center of gravity and the tipping point as seen in Figure 3.1

Figure 5.5: Grid map layers of control limits. The robot has to comply to either one to be able to safely traverse that region.

(a) Path near the island obstacle in the map built with `slam_gmapping`. Manual controls widgets are placed around the robot.

(b) Dangerous Jackal position at roughly two thirds of the planned path.

Figure 5.6: The Jackal encounters a risky pose when navigating in the `slam_gmapping` grid.

obstacle as is the case in Figure 5.6b, we see that the path curves around the island. When navigating in the conservative map of our method the island is avoided successfully.

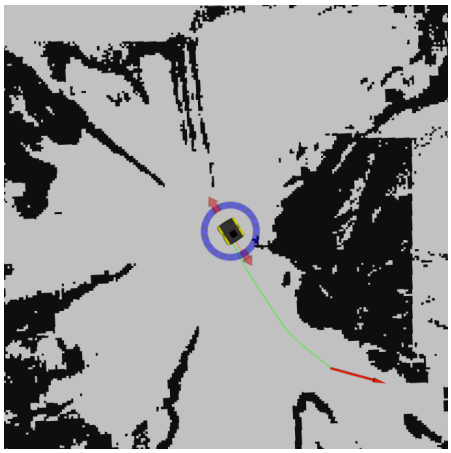


Figure 5.7: Path generated around the island by `move_base` using the conservative map of our method.

# 6

# Conclusion

## 6.1 Summary of this Thesis

In this thesis we explored what dangers a mobile robot faces in rough terrain. We split the aspect of traversability into sub-problems with the general features of slopes and edges.

We found the angle limit for the slopes to be safely traversable. Our models regard the case of the robot tipping and the motors not being powerful enough. The calculations include the robot's center of gravity and weight, its geometry, its wheel torque or drive wattage and its maximal velocity and acceleration. The center of gravity was experimentally determined, and we validate the tip-over formula on a ramp. A further experiment was performed on the ramp with the aim of including slippage in the slope limit. The Jackal robot we used slips mainly due to its skid steering. The slippage we saw does not seem to correlate with the ramp inclination and was not accounted for further. This might be different for non-skid-steering robots.

We also found the limit of the height of edges that the robot can overcome. This value was found experimentally. We observed that the approach angle towards the edge significantly influences the traversable edge height.

The goal of the thesis was to map the traversability of regions around the robot. We aimed to make this happen separately from path and motion planning to retain modularity. It becomes apparent that traversability greatly depends on the orientation of obstacles relative to the robot and the robot's speed and acceleration. In the mapping step however the trajectory the mobile robot plans to take is not yet determined. This turns out to be a major drawback of modularity. With the intention to guarantee safe driving zones in the terrain, we are forced to make conservative assumptions about the future trajectory. As a result actual obstacles are exaggerated and our map becomes restrictive. To mitigate this issue, we propose modified control software. Instead of only supplying a map, we also generate control limits to influence the trajectory. By doing so we can relax some conservative assumptions regarding the robot's behavior in the environment.

When evaluating our methods we found that the extension of providing control limits produces a less restrictive map than our conservative approach with little to no loss in modularity. Both successfully captured and marked dangerous terrain. We integrated our methods with third party software to then navigated the environment using our map.

We compared our approaches to a standard 2-dimensional mapping approach, and discovered that the latter is not well suited for use in rough terrain.

## 6.2 Future Work

The main focus of additional works to improve our methods should be on reducing the assumptions needed regarding the robot's trajectory. With a better estimation the traversability calculations can be tailored specifically to the actual situation the robot would face on its path. Due to lack of time, we are not able to further explore and build upon the proposed solutions.

Our method generates only a small local map of the robot's vicinity. Ideally our work is used as a local map for local trajectory planning within a large standard map. The basic global trajectory can be planned in the large, stationary map. The local robot-centric map would then have access to the rough trajectory that the robot plans to follow. With this knowledge, the obstacles in the local map can then be adjusted to the roughly planned movement.

As an alternative the grid map could be extended to cover orientation dependent obstacles without knowing the trajectory. All orientations in a circle could be discretized into a handful of directions the robot can move in, such as north, east, south and west and angles in between. The idea is that our process creates a traversability map for each orientation. When path planning the maps need to be switched as the path curves. This could be represented as a graph where equivalent cells in maps of the next orientation increment are connected. As the robot orientation is known in each traversability map, edges can be classified with the approach angle and tipping can be calculated towards the actual important direction. This however requires a specialized path planner, that can interpret and search the graph, which is contrary to our modular approach.

With the Jackal, we did not find a correlation between inclination and slippage. Nonetheless, sliding is a big issue on sloped ground. Our slippage experiment could be repeated with other types of robots that do not rely on skid steering. One could also account for friction during the experiments and when calculating traversability.

Finally, the sensor errors in the elevation map need to be reduced. It would be useful to add post-processing to the `elevation` layer in the filter chain before the slope and edge behaviors are executed. Because we needed to detect a small conservative edge height we were reluctant to add smoothing operations.

## Access to the Software Developed

The software developed during this thesis proves to be adoptable to a variety of mobile robots. In this thesis we demonstrated the effects of changing some parameters to those of imaginary robots. One can easily adjust the code to a new robot by replacing the configurations file `jackal_properties.yaml`, especially if the robot's data sheet contains detailed information about how the robot is built. The software is available on GitHub[1].

[1] https://github.com/Max-Ole/Bachelor_Thesis

# Bibliography


[1] Boynton, R. Measuring weight and all three axes of the center of gravity of a rocket motor without having to re-position the motor. In: *Space Electronic, Inc., Berlin*, 2002.

[2] Brooks, R. A. A robust layered control system for a mobile robot. In: *IEEE journal on robotics and automation* 2(1):14–23, 1986.

[3] Brooks, R. A. New approaches to robotics. In: *Science* 253(5025):1227–1232, 1991. DOI: 10.1126/science.253.5025.1227.

[4] Chen, Y. F., Everett, M., Liu, M., and How, J. P. Socially aware motion planning with deep reinforcement learning. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 1343–1350.

[5] Clearpath Robotics *JACKAL_DATA_SHEET_2020*. URL: https://go.pardot.com/l/92812/2015-07-20/g7dr/92812/2574/JACKAL_DATA_SHEET_2020.pdf (visited on 01/02/2022).

[6] Elfes, A. *Occupancy grids: A probabilistic framework for robot perception and navigation*. Carnegie Mellon University, 1989.

[7] Fankhauser, P., Bloesch, M., Gehring, C., Hutter, M., and Siegwart, R. Robot-Centric Elevation Mapping with Uncertainty Estimates. In: *International Conference on Climbing and Walking Robots (CLAWAR)*. 2014.

[8] Fankhauser, P., Bloesch, M., and Hutter, M. Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization. In: *IEEE Robotics and Automation Letters (RA-L)* 3(4):3019–3026, 2018. DOI: 10.1109/LRA.2018.2849506.

[9] Fankhauser, P. and Hutter, M. A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation. In: *Robot Operating System (ROS) – The Complete Reference (Volume 1)*. Ed. by A. Koubaa. Springer, 2016. Chap. 5. ISBN: 978-3-319-26052-5. DOI: 10.1007/978-3-319-26054-9_5.

[10] Grisetti, G., Stachniss, C., and Burgard, W. Improved techniques for grid mapping with rao-blackwellized particle filters. In: *IEEE transactions on Robotics* 23(1):34–46, 2007.

[11] Grisetti, G., Stachniss, C., and Burgard, W. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, pp. 2432–2437.

[12] Hertle, A. and Dornhege, C. Efficient extensible path planning on 3D terrain using behavior modules. In: *2013 European Conference on Mobile Robots*. IEEE. 2013, pp. 94–99.

[13] Kim, C.-S., Hong, K.-S., Yoo, W.-S., and Park, Y.-W. Tire-road friction estimation for a wheel-driven field robot. In: *2008 SICE Annual Conference*. IEEE. 2008, pp. 782–787.

[14] Kruijff-Korbayová, I., Grafe, R., Heidemann, N., Berrang, A., Hussung, C., Willms, C., Fettke, P., Beul, M., Quenzel, J., Schleich, D., et al. German Rescue Robotics Center (DRZ): A Holistic Approach for Robotic Systems Assisting in Emergency Response. In: *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2021, pp. 138–145. DOI: 10.1109/SSRR53300.2021.9597869.

[15] Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K. The office marathon: Robust navigation in an indoor office environment. In: *2010 IEEE international conference on robotics and automation*. IEEE. 2010, pp. 300–307.

[16] McGhee, R. B. and Frank, A. A. On the stability properties of quadruped creeping gaits. In: *Mathematical Biosciences* 3:331–351, 1968.

[17] Morales, J., Martínez, J., Mandow, A., Serón, J., Garcia, A., and Pequeño-Boter, A. Center of Gravity Estimation and Control for a Field Mobile Robot with a Heavy Manipulator. In: *IEEE 2009 International Conference on Mechatronics, ICM 2009*, Apr. 2009. DOI: 10.1109/ICMECH.2009.4957112.

[18] Moravec, H. and Elfes, A. High resolution maps from wide angle sonar. In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. 1985, pp. 116–121. DOI: 10.1109/ROBOT.1985.1087316.

[19] Nguyen, N. T., Schilling, L., Angern, M. S., Hamann, H., Ernst, F., and Schildbach, G. B-spline path planner for safe navigation of mobile robots. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 339–345.

[20] Open Robotics *filters - ROS Wiki*. URL: http://wiki.ros.org/filters (visited on 01/03/2022).

[21] Ortiz, L. E., Cabrera, V. E., and Goncalves, L. M. Depth data error modeling of the ZED 3D vision sensor from stereolabs. In: *ELCVIA: electronic letters on computer vision and image analysis* 17(1):1–15, 2018.

[22] Parker, L. E., Jung, D., Huntsberger, T., and Pirjanian, P. Opportunistic adaptation in space-based robot colonies: Application to site preparation. In: *Proceedings of World Automation Congress*. 2000.

[23] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. ROS: an open-source Robot Operating System. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[24] Slaughter, D. C., Giles, D., and Downey, D. Autonomous robotic weed control systems: A review. In: *Computers and electronics in agriculture* 61(1):63–78, 2008.

[25] Ueckermann, A., Wang, D., Oeser, M, and Sreinauer, B Towards Contactless Skid Resistance Measurement. In: *Safer Roads International Conference*. 2014.

[26] Werf, H. M. van der Assessing the impact of pesticides on the environment. In: *Agriculture, Ecosystems & Environment* 60(2-3):81–96, 1996.