Whole-Body Planning for Obstacle Traversal with Autonomous Mobile Ground Robots

Ganzkörperbewegungsplan zur Überwindung von Hindernissen mit autonomen mobilen Bodenrobotern

Master-Thesis von Martin Oehler

November 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Department Computer Science
Simulation, Systems Optimization and Robotics Group

Whole-Body Planning for Obstacle Traversal with Autonomous Mobile Ground Robots
Ganzkörperbewegungsplan zur Überwindung von Hindernissen mit autonomen mobilen Bodenrobotern

Vorgelegte Master-Thesis von Martin Oehler

1. Gutachten: Prof. Dr. Oskar von Stryk
2. Gutachten: Dr.-Ing Stefan Kohlbrecher

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Martin Sven Oehler, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

Advances in the design of mobile robotics systems enable the application in new tasks like disaster response, inspection and logistics. Recently, autonomous robots have been a major focus of research. Compared to teleoperated machines, they work faster and more efficiently especially in environments with degraded connectivity. Without human supervision, the underlying algorithms need to be robust against unexpected circumstances to prevent damage to the robotic system and the environment.

A common challenge for autonomous robots is the traversal of obstacles. To continue its task, the robot has to cross the obstacle without tipover instabilities. So far, research on prevention of vehicle tipover is mostly limited to simple systems with few degrees of freedom (DOF).

In this thesis, a novel whole-body motion planning approach is proposed. By using a model of the world, the joint configuration is optimized for stability along a given path. The proposed method evaluates whether a safe traversal is possible and generates a motion plan that allows the robot to cross generic obstacles without tipover. Collisions are prevented by modeling them as constraints of the optimization.

This approach is evaluated on a tracked vehicle with adjustable flippers and a five DOF manipulator arm. The proposed method leverages the flippers to improve stability by maximizing ground support and the arm to shift the center of mass. Additionally, the platform features various sensors to perceive its environment.

Performance of the whole-body motion planning is evaluated in simulation and on the real robot. In multiple scenarios, it is shown that the approach effectively prevents tipover and increases robot stability.

Zusammenfassung

Fortschritte in der Entwicklung mobiler Robotersysteme ermöglichen den Einsatz in neuen Aufgabengebieten wie Katastrophenschutz, Inspektion und Logistik. Seit K urzem sind autonome Roboter ein Schwerpunkt der Forschung. Im Vergleich zu ferngesteuerten Maschinen arbeiten sie schneller und effizienter, insbesondere in Umgebungen mit eingeschränkter Konnektivität. Ohne menschliche Aufsicht müssen die zugrunde liegenden Algorithmen jedoch robust gegen unerwartete Umstände sein, um Schäden am Robotersystem und der Umwelt zu vermeiden.

Eine besondere Herausforderung für autonome Roboter ist die Überwindung von Hindernissen. Um seine Aufgabe fortsetzen zu können, muss der Roboter in der Lage sein, das Hindernis sicher zu überqueren. Bisher beschränkt sich die Forschung zu diesem Thema meist auf einfache Systeme mit wenigen Freiheitsgraden.

In dieser Arbeit wird ein neuer Ansatz zur Ganzkörperbewegungsplanung vorgeschlagen. Durch die Verwendung eines Weltmodells wird die Gelenkkonfiguration auf Stabilität entlang des Pfades optimiert. Die vorgeschlagene Methode bewertet, ob eine sichere Überwindung möglich ist und generiert einen Bewegungsplan, der es dem Roboter erlaubt, generische Hindernisse ohne Umkippen zu überqueren. Kollisionen werden verhindert, indem sie als Nebenbedingungen der Optimierung modelliert werden.

Dieser Ansatz wird an einem Kettenfahrzeug mit verstellbaren Flippern und einem fünfgelenkigen Manipulatorarm ausgewertet. Die vorgeschlagene Methode nutzt die Flipper, um den Bodenkontakt zu maximieren, und den Arm, um den Massenschwerpunkt zu verschieben. Zusätzlich verfügt die Plattform über verschiedene Sensoren, um die Umgebung wahrzunehmen.

Die Performance der Ganzkörperbewegungsplanung wird in der Simulation und am realen Roboter ausgewertet. In mehreren Szenarien wird gezeigt, dass der Ansatz Umkippen verhindert und die Stabilität des Roboters erhöht.

Contents

1 Introduction

1.1 Motivation

For a long time, the usage of robots in industrial settings was limited to production in caged-off areas with no human interaction. Only recently, effort has been made to make robots mobile and extend their use cases.

Mobile robotic systems can be used for a variety of new tasks like disaster response, inspection and logistics. Compared to their stationary counterparts, they face a lot of new challenges. Their surroundings are generally unstructured and previously unknown. This makes common tasks like navigation and manipulation difficult.

Teleoperated robots have successfully been used in the past. The nuclear power plant disaster in Fukushima brought the need for remotely-operated machines. Due to the high radiation exposure, humans were unable to work for an extended period of time without risking their health. Instead, necessary tasks like the removal of debris were performed by excavator robots (Figure 1).

These systems put a lot of responsibility on the operator. Just by using the provided sensor equipment, he has to assess the situation and carefully control the robot. This limits working speed and is also prone to errors.

Figure 1: *BROKK 330* used by TEPCO to remove debris at the Fukushima Daiichi site.

These issues are alleviated by semi-autonomous robots. They take load of the operator by automating typical tasks. Controlling the robot now happens on a higher level by issuing abstract commands. This increases reliability and working speed.

The next step is full automation. Autonomous robots that no longer require an operator have compelling advantages. With no human operator in the loop, they generally work faster, more efficient and also cheaper. This is especially true if the site of operation is remote and connectivity is degraded. But without an operator monitoring the execution, robot behavior has to be robust against unexpected circumstances. In a changing environment, the robot must be able to perform its task safely to prevent damage to itself or the surroundings.

1.2 ARGOS Challenge

The ARGOS Challenge (Autonomous Robot for Gas and Oil Sites)[1] arranged by Total was the first robotics challenge in the oil and gas industry. Its goal was the development of a fully-autonomous robot for inspection with a focus on seamless switching to teleoperated and semi-autonomous behaviors.

The competition incorporated common tasks like reading gauges, measuring temperatures and checking lever positions. Simultaneously the robot had to detect anomalies and react to unexpected situations.

[1] `http://www.argos-challenge.com`

Oil and gas sites are generally located in challenging locations with extreme conditions like cold and wet environments. Therefore, the robustness of the robot platform was a major concern.

TU Darmstadt and Taurob collaborated as Team Argonauts to participate in the challenge. After two preliminary events and the finals in March 2017, Team Argonauts won the ARGOS Challenge. Now, the next goal is to bring the promising results of the competition to real platforms.

1.3 Argonaut Tracker

The Argonaut Tracker (Figure 2) was developed by Team Argonaut to compete at the ARGOS Challenge. It combines a robust robot platform with multiple sensors to enable complex autonomous behaviors.

The robot is designed to work in hazardous environments and is therefore completely waterproof. The chassis is sealed and pressurized to prevent accidents even in an explosive atmosphere. This is accredited by the ATEX certificate.

The two tracks with rubber profiles enable the robot to navigate rough terrain. A unique flipper design changes the shape of the track to traverse obstacles.

An arm with five DOF is mounted on the robot base that ends in a sensor head.

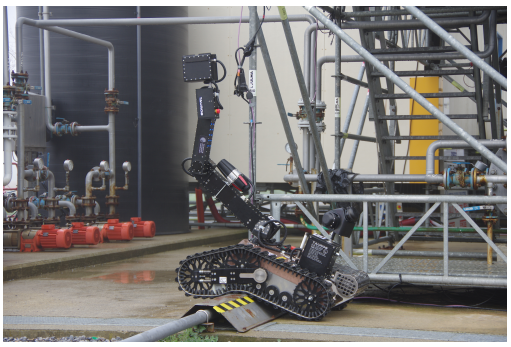


Figure 2: Argonaut Tracker at the ARGOS Challenge

Sensors

The robot is equipped with various sensors to perceive its surroundings and current state. The robot orientation is determined using an IMU (Inertial Measurement Unit). It is mounted on the robot base plate and measures accelerations. For situational awareness, cameras are located at front and rear of the chassis. The environment is scanned by two Hokuyo laser scanners mounted at the rear. They are attached to a rotating platform to create a 3-dimensional point cloud of the environment. Their data is used for simultaneous localization and mapping (SLAM).

The sensor head is equipped with a camera array. The RGB-D camera creates depth images in close range. The wide angle and zoom camera are used for vision-related tasks. Hot spots like fire can be detected with the thermal camera. Furthermore, the head includes a microphone and an ultrasonic detector.

Hector Tracker

While the Argonaut Tracker was used for the competition, evaluation of this work was performed on the Hector Tracker. The robot is identically constructed with a few modifications. Since it was built for academic research, the ATEX certificate is not required. Moreover, the dual-Hokuyo setup was replaced with a single Velodyne VLP-16.

1.4 Whole-Body Planning

In an autonomous setting, a robust behavior of the robot is very important. The robot should be able to adapt to the current situation and possible disturbances. Damaging the hardware or the environment is expensive especially when human intervention is needed.

A common challenge for autonomous robots is the traversal of obstacles. Even in a structured environment like an oil site, obstacles can occur for example due to damage or changes to the platform. The robot should still be able to continue its work and therefore has to be able to cross the obstacle in a safe way. To effectively increase its stability, the robot can use all joints (whole-body). Transferred to the Argonaut Tracker, it can use its arm with the sensor head for balancing and the flippers to maintain a stable ground position.

The problem can be divided into two sub-problems: Pre-planning and reactive behavior. The pre-planning approach relies on a map of the environment, which can be used to create a plan, whereas the reactive behavior adapts the plan to the current situation.

Generally, both aspects are needed. Planning in advance is not sufficient as unexpected disturbances like unstable ground and slippage require constant corrections. Neither is a reactive behavior alone sufficient, as the robot might reach irrecoverable states, leading to inevitable tipover.

While both aspects are important, this work will focus on pre-planning.

1.5 Overview

In the following, a short overview of the structure is given. Section 2 explains the foundations, which are general principles applied in this thesis. This is followed by the Related Work in Section 3 that takes a look at existing research in the field. In Section 4, the proposed approach is explained in detail. This basic algorithm is later improved in Section 4.3. Section 5 goes into implementation details and presents the used framework and libraries. Next, the approach is evaluated in Section 6. In different scenarios, the effectiveness of the algorithm is demonstrated in simulation and on the real robot. Finally, the conclusion in Section 7 summarizes the approach and results presented in this work and shows ways for future improvements.

2 Foundations

The following section explains basic concepts applied in this work. The first part introduces the transformation notation that is used to describe the proposed method. The scope of this work is the prevention of vehicle tipover, therefore the general stability criterion is presented next. Afterward, different world representations are discussed that are used for contact estimation and collision checking during motion planning.

2.1 Transformations

To keep track of different parts of the robotic systems or objects in the world, they are assigned a coordinate system or frame. These frames are defined relative to another frame with a transformation. The world coordinate system is labeled W.

The transformation from frame A to B is given by a $3x3$ rotation matrix ${}^{A}R_{B}$ and a $3x1$ translation vector ${}^{A}r_{B}$. These can be combined in a $4x4$ homogeneous transformation matrix ${}^{A}T_{B}$ (Equation 1), where $\mathbf{0} = (0,0,0)^{T}$ is the zero vector.

$$\left(\begin{array}{c|c} {}^{A}R_{B} & {}^{A}r_{B} \\ \hline \mathbf{0}^{T} & 1 \end{array}\right) \tag{1}$$

The advantage of the homogeneous representation is, that transformations can be chained by multiplying them:

$${}^{A}T_{C} = {}^{A}T_{B}\,{}^{B}T_{C} \tag{2}$$

Points $p = (x,y,z)^{T}$ given relative to a frame A, are denoted with ${}^{A}p$. To transform the point, the homogeneous form has to be used $p_{hom} = (x,y,z,1)^{T}$. A transformation from A to B is done by multiplication by the respective matrix:

$${}^{B}p_{hom} = {}^{B}T_{A} \cdot {}^{A}p_{hom} \tag{3}$$

In this case, the inverse transform ${}^{B}T_{A}$ is needed. For the special case of transformation matrices, the inverse is given by:

$${}^{B}T_{A} = ({}^{A}T_{B})^{-1} = \left(\begin{array}{c|c} ({}^{A}R_{B})^{T} & -({}^{A}R_{B})^{T} \cdot {}^{A}r_{B} \\ \hline \mathbf{0}^{T} & 1 \end{array}\right) \tag{4}$$

2.2 Stability

To prevent mobile robots from falling over, their stability has to be considered while driving. Stability can be divided into static and dynamic stability. Static stability implies the robot is stable without any motion. This only depends on the contact points of the robot with the ground (contact geometry), the position of the center of mass (COM) and external forces like gravity and is therefore independent of total mass.

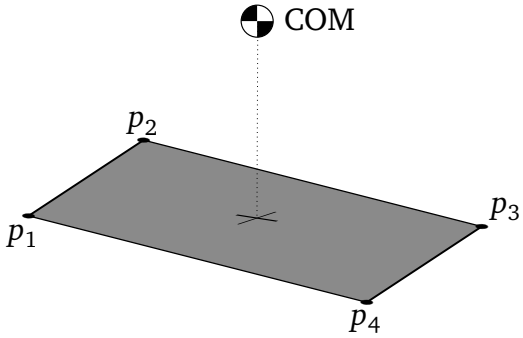


Figure 3: Support polygon of the four ground contact points p_1, p_2, p_3 and p_4. The center of mass (COM) projects into the support area (gray), so the contact is stable.

Only contact points that are part of the convex hull of all contact points when projected onto a horizontal plane have to be considered. They define the support polygon (shown in Figure 3). The area inside the support polygon is called support area.

As long as the support area is greater than zero and the COM is inside this area when projected to a horizontal plane, the robot is statically stable. Therefore, the robot needs at least three contacts with the ground.

Dynamic stability also considers forces and moments produced by movements. The stability must be ensured by controlled motions and the modeling and usage of inertia [1, p. 49-51].

2.3 World Representation

To navigate and interact with an unknown environment, a mobile robot has to create a map of its surroundings. In this work, the map is used for contact estimation and collision checking.

There are multiple representations available, each with different advantages and disadvantages. The methods have in common that the world is represented as an n-dimensional uniform grid.

2.3.1 Occupancy Grid

An occupancy grid is a probabilistic map representation originally introduced by Moravec *et al.* [2]. Each grid cell contains a random variable representing the existence of an obstacle. The mapping algorithm computes the approximate posterior, that a cell, indexed by (x, y, z), is occupied $p(o_{x,y,z}|m_{1:t})$ given the measurements $m_{1:t}$. Figure 4 shows an example of a 2-dimensional occupancy grid.

Collision checks using an occupancy grid are done by evaluating the grid at boundary points of the robot. If the probability is above a certain threshold, the robot is in collision. However, choosing the threshold is ambiguous and there is no direct information how to recover from a collision.

2.3.2 Signed Distance Field

In an Signed Distance Field (SDF) each cell contains the distance to the closest surface. The distance is evaluated at the center of the cell. Cells in a 3-dimensional grid are also called voxels.

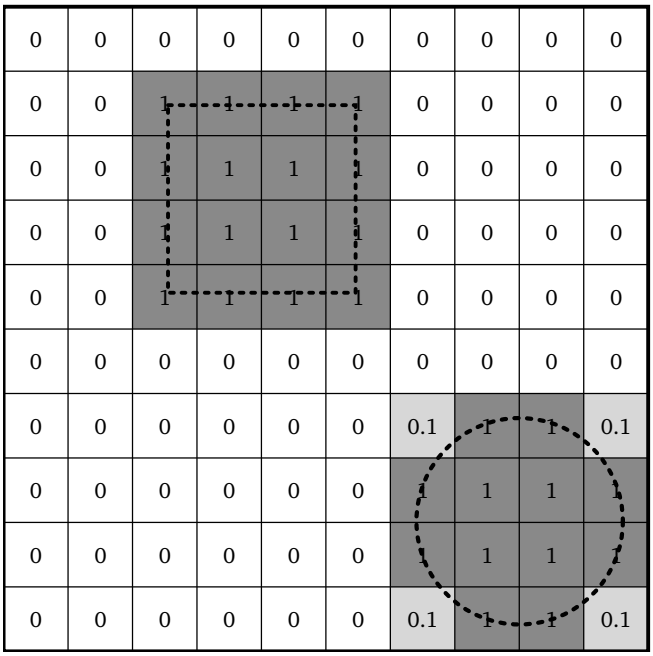


Figure 4: 2D occupancy grid. Each cell contains the approximate posterior that it is occupied. Obstacles are marked with dotted lines.

Cells inside and outside of objects are differentiated by their respective sign with positive values outside of objects and negative inside. Therefore surfaces are implicitly encoded at the zero-transition.

SDFs are differentiable and the gradient points away from obstacles. This makes them a good choice for collision checking in optimization-based planning approaches. The manipulator can recover from contacts by simply following the gradient.

Truncated SDF

Truncated Signed Distance Fields (TSDFs) [3] contain the projective distance along the sensor ray to the surface. They are only defined in a region around surfaces up to the truncation distance. Due to this, they are fast to construct. Figure 5 shows a 2-dimensional TSDF.

Euclidean SDF

Unlike the TSDF, Euclidean Signed Distance Fields (ESDFs) return the real Euclidean distance to the closest surface. They are typically created from Occupancy Grids [4] or TSDFs [5]. Figure 6 shows a 2D ESDF.

While being computationally expensive to create, ESDFs are better suited for planning than TSDFs due to their accurate representation of distances. To enable real-time application on robotic systems, maps are often built incrementally.

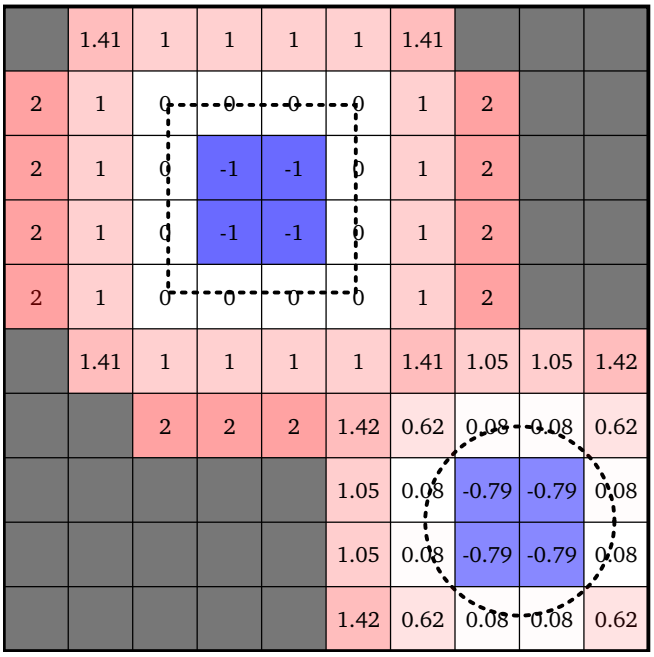


Figure 5: 2D TSDF with a truncation distance of $2\,\mathrm{m}$. Each cell covers an area of $1x1$ m and contains the distance to the closest surface. The illustration is idealized, so distances are Euclidean instead of projective. Obstacles are marked with dotted lines.

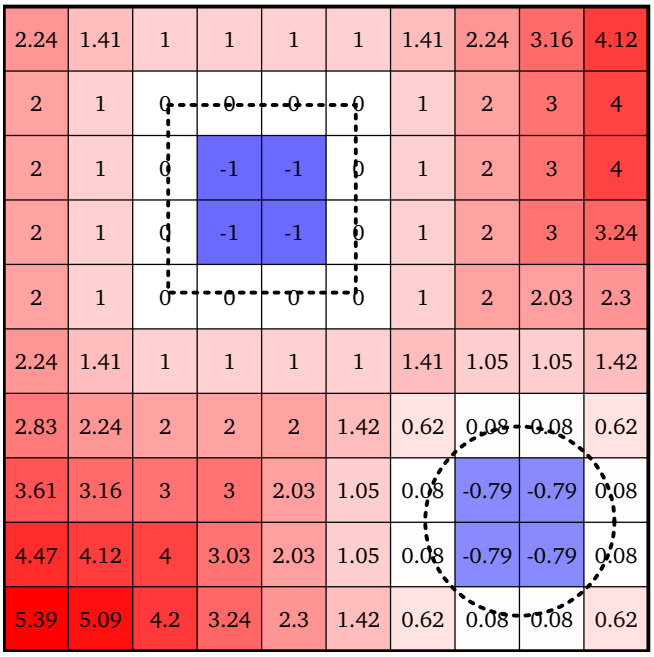


Figure 6: 2D ESDF. Each cell has a size of $1x1$ m and contains the Euclidean distance to the closest surface. Obstacles are marked with dotted lines.

3 Related Work

Planning stable motions combines approaches from multiple research topics. Therefore, common metrics to asses stability margins of a mobile system are introduced first. Subsequently, contact estimation approaches to predict the contact geometry of the robot are presented. This is followed by a look on existing research on vehicle tipover prevention and trajectory optimization.

3.1 Stability Margin

The criterion for static stability introduced in Section 2.2 provides a binary decision whether a configuration of the robot is stable or unstable. In contrast, stability margins allow to evaluate the stability as a continuous value. Different measures have been proposed in the past.

McGhee and Frank [6] proposed a stability margin for the analysis of quadruped walking gaits. They defined the margin as the shortest distance from the COM, projected onto the plane of the support polygon, to any point on the boundary of the support polygon. However the magnitude of the margin does not change, when the COM-height changes (top-heaviness).

An energy-based formulation was proposed by Messuri *et al.* [7]. The stability margin relates to the impact energy which can be sustained by the vehicle without overturning. It is calculated for each support polygon edge individually by computing the work required to rotate the body COM about that edge until the vehicle becomes unstable. The stability of the whole system is defined as the minimum energy stability level of all edges. Compared to the work by McGhee *et al.*, this formulation is sensitive to top-heaviness.

3.1.1 Force-Angle Stability Margin

The force-angle stability margin was proposed by Papadopoulos *et al.* [8] and has been used in numerous works [9, 10, 11, 12]. It combines the distance of the COM to the support polygon edges with the angles of the gravity vector to the edges.

Similar to the energy-based formulation by Messuri *et al.* [7], the measure is sensitive to height changes of the COM. Furthermore, the evaluation is computationally cheap. An illustration of the important components of the stability margin is shown in Figure 7. In the following, the computation of the stability margin is shortly summarized.

The contact geometry is defined by the n corner points of the support polygon $p_i, i = 1, ..., n$. They are numbered in clockwise order when viewed from above. p_c refers to the location of the COM. The line between adjacent pairs of points defines a tipover axis a_i (Equation 5). Tipping of the robot is only possible over one of these edges.

$$a_i = p_{(i+1) \bmod n} - p_i, \quad i = 1, ..., n \tag{5}$$

Next, axis normals through p_c are computed by projecting p_c onto each axis a_i,

$$l_i = (I - \hat{a}_i \hat{a}_i^T)(p_{i+1} - p_c) \tag{6}$$

where $\hat{a} = a/\|a\|$ and I refers to the $3x3$ identity matrix.

The net force vector f_r describes the sum of all forces acting on the vehicle. We assume that no external forces act on the vehicle except gravity, therefore $f_r = m_{tot} g$, where m_{tot} is the total

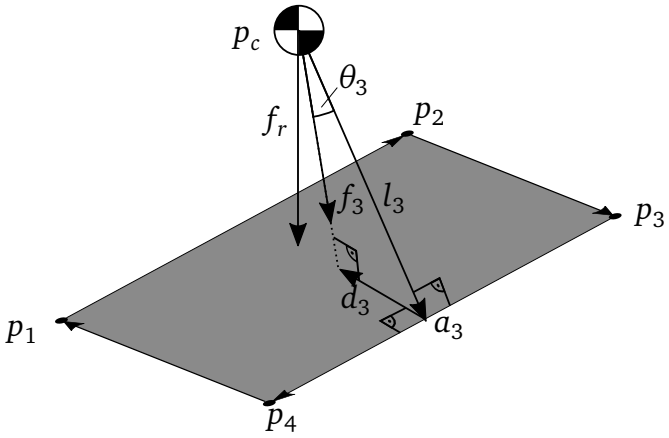


Figure 7: Illustration of the force-angle stability margin components of one axis. The tipover axis a_3 connects the contact points p_3 and p_4. l_3 is the axis normal that goes through the COM location p_c. f_r labels the net force vector that acts on the COM. Its projection onto a_3 is shown with f_3. The signed-angle between l_3 and f_3 is labeled θ_3. Lastly, d_3 indicates the distance between f_3 and a_3.

vehicle mass and g the gravitational acceleration. We only have to consider the force component f_i that acts about the respective axis a_i, so we again compute the projection:

$$f_i = \left(I - \hat{a}_i \hat{a}_i^T\right) f_r \tag{7}$$

Therefore, f_i and l_i lie on a plane orthogonal to a_i.

Similarly, the net moment n_r is a sum of all moments acting on the COM. We assume that no external moments act on the vehicle, so $n_r = 0$.

The force-angle stability margin consists of three components. The first component is the minimum distance of each axis a_i to the respective force component f_i. The distance vector d_i is computed with:

$$d_i = -l_i + \left(l_i \cdot \hat{f}_i\right) \hat{f}_i \tag{8}$$

where $(\cdot)$ refers to the dot product of two vectors and $\hat{f}_i = f_i / \|f_i\|$.

The next component of the stability measure is the angle θ_i between the axis normal l_i and the projected force vector f_i,

$$\theta_i = \sigma_i \cos^{-1}\left(\hat{f}_i \cdot \hat{l}_i\right) \tag{9}$$

with $\hat{l}_i = l_i / \|l_i\|$. The sign of the angle is given by σ_i and depends on the force vector f_i. If f_i is directed inside the support polygon, the sign is positive and otherwise negative:

$$\sigma_i = \begin{cases} +1, & \left(\hat{f}_i \times \hat{l}_i\right) \cdot \hat{a} > 0 \\ -1, & otherwise \end{cases} \tag{10}$$

The last component of the stability margin is the norm of the projected force vector f_i. Therefore, the stability of axis i is given by:

$$\beta_i = \theta_i \cdot \|d_i\| \cdot \|f_i\| \tag{11}$$

The stability of the whole system β corresponds to the least stable axis:

$$\beta = \min(\beta_1, ..., \beta_n) \tag{12}$$

A positive value indicates a stable position, critical stability occurs at 0. For negative values, the robot is unstable and tips over.

3.2 Contact Estimation

To compute the force-angle stability margin (Section 3.1.1), the contact points of the robot with the ground are needed. These are unknown beforehand, so they have to be estimated using the world model captured by the robot. The ground contact geometry constraints the pose of the robot. Therefore, contact estimation and pose estimation are strongly coupled problems. There are different approaches available to estimate the contact geometry and robot pose.

Norouzi *et al.* [13] used a rigid body dynamics simulation with the open-source simulator Open Dynamics Engine (ODE). As shown in their work, the approach is very accurate, because real physical interactions are simulated. The authors did not give any information about the time each estimate takes. In general, physics-based simulation approaches are comparably slow since each time step has to be simulated until the robot is in a steady state.

A geometrically-motivated approach was taken by Daun and Schubert [14]. The contact points are derived by constructing the contacts of the robot underside with a point cloud. The approach is very fast because only few and simple computations are needed. The robot is approximated by a plane, so complex geometry like sub-tracks is not considered by the estimation. Moreover, raw point clouds are very noisy, so they have to be filtered in some way. This makes the approach rather inaccurate.

To address these shortcomings, an optimization-based approach will be presented in Section 4.1.

3.3 Stability Control

Only limited research on the prevention of vehicle tipover on uneven terrain has been conducted. Approaches can be categorized into two categories: pre-planning and reactive behavior.

Reactive

Grand *et al.* [9] optimized stability and traction of the wheel-legged robot *Hylos* by adjusting its posture. To assess the robot stability, the force-angle stability margin is computed. The corrections are calculated while driving, no model of the world is required.

A different approach for the same robot was proposed by Besseron *et al.* [10]. They exploit existing redundancies of robot kinematics by decoupling control of posture and trajectory. The stability is optimized with a potential field formulation.

In contrast, Ohno *et al.* [15] used a tracked vehicle with flippers for tipover prevention in roll direction. Similar to the previous approaches, online reconfigurations of the flipper are performed to increase stability. The normalized energy stability margin [16] is used to asses the stability. The approach does not utilize a world model. Instead, contact points are approximated by deciding between different cases dependent on sub-track contact.

Pre-Planning

There are also a few approaches that consider stability during motion planning.

Norouzi *et al.* [11] proposed a combined motion and path planning method for the tracked vehicle *iRobot Packbot*. The approach generates a path while also finding the optimal configuration of the flipper and 1-DOF arm. The optimization is embedded into the *A** search algorithm and considers visibility, traction, energy consumption and stability. The simulation engine ODE is used together with a 3D model of the environment to predict contact points.

The work of Beck *et al.* [12] was implemented on the same platform. Given a path, the approach optimizes flipper and arm position by considering stability, equal distribution of contact forces, low energy consumption and operation within nominal joint positions. The stability is measured using the force-angle stability margin. The prediction of contact points is idealized and depends on the flipper position and terrain slope.

While a few methods exist to prevent tipover of reconfigurable mobile robots, they are not suitable for the Hector Tracker. With five DOF its arm geometry is much more complex than previous evaluation platforms. Furthermore, important aspects like collision avoidance have not been addressed yet.

3.4 Trajectory Optimization

There has also been research on optimization-based trajectory planning. While not considering stability, the joint configuration is also optimized for related criteria like collisions.

Ratliff *et al.* [17] propose the standalone planner CHOMP. The planner continuously refines the trajectory by updating it with covariant gradient techniques. A SDF is used to represent obstacles in the cost function.

This approach was later improved by Kalakrishnan *et al.* [18] using a stochastic technique. The algorithm generates noisy trajectories to explore the space, making it less prone to local minima. Moreover, the optimization does not rely on gradient information.

Finally, Pavlichenko *et al.* [19] built upon this work by optimizing for multiple criteria and adding adaptive collision checking.

4 Method

In this section, the proposed whole-body motion planning method is presented. First, a novel approach to estimate ground contacts formulated as an optimization problem is introduced. This serves as the basis for the following stability optimization approach. Subsequently, improvements of the original approach are proposed, which increase the performance substantially.

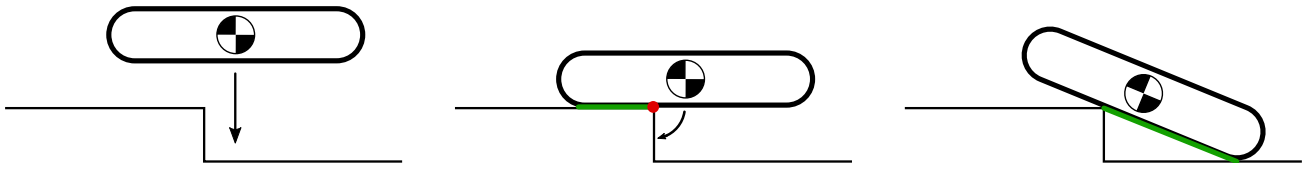
4.1 Iterative Optimization-based Contact Estimation

To compute the stability margin, the contact points of the robot with the ground have to be predicted. Using the world model, the contact estimation determines the contact geometry and the dependent robot pose. The robot pose is given by the transformation from world frame to base link ${}^{W}T_{B}$.

The complete robot pose has six DOF, the position given by x, y and z and the orientation given by the Euler-angles roll ϕ, pitch θ and yaw ψ. The contact estimation takes a partial pose $(x, y, \psi)^{T}$ that is composed of the robot position on the xy-plane and the heading (yaw). Additional required information is the joint configuration $\mathbf{q}$, that determines the geometry of the vehicle underside and the location of the COM relative to the base ${}^{B}p_{c}$. To make computations easier, a frame is aligned with the COM with the same orientation as the base link, given by ${}^{B}T_{C}$.

The missing pose components $(z, \phi, \theta)^{T}$ are constrained by the contact geometry and to be estimated. Initially they are set to 0. If the contact estimation is called repeatedly, the previous results are used for initialization. The algorithm is iterative, so the current estimate ${}^{W}T_{B}^{(k)}$ is indexed with the iteration number k starting at 0.

The proposed approach uses iterative optimizations to find an accurate estimation of the robot pose in a short time. In the following, a brief explanation of the algorithm will be given. A visualization of the process can be seen in Figure 8. In a physics-based approach, each iteration simulates a time step. In contrast, in the proposed optimization-based formulation each iteration simulates the complete time until the contact geometry changes. The first iteration is a *falling* step. It simulates free fall until the robot is in contact with the ground for the first time. Afterward, each iteration is a *tipping* step that simulates tipping of the robot until it is in a stable state. Each iteration is formulated as an optimization problem, that models the effects of gravity and collisions. A pseudo-code of the complete algorithm can be seen in Algorithm 1.



(a) Initial state **(b)** After falling **(c)** After tipping

Figure 8: Three steps of the contact estimation. The initial state is shown in 8a. After the falling phase (8b), the robot is in contact with the ground. The rotation axis is marked with a red dot. In the final state, the robot is stable after tipping (8c). The green line visualizes the support polygon. The free parameters of the two optimizations steps are shown with arrows.

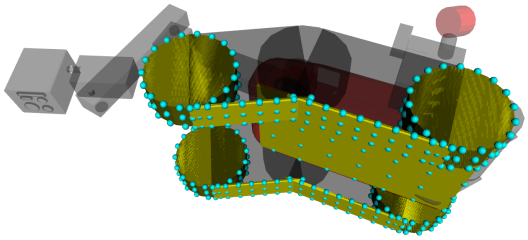


Figure 9: The robot underside is modeled with simple geometric shapes, shown in yellow. The cyan dots mark sampling points.

Objective Formulation

The general objective of each iteration is to minimize the z-coordinate of the center-of-mass (COM) in world frame ${}^{W}p_c^z$ (Equation 13). This acts identically to the gravity in a simulation approach.

$$\varphi(\mathbf{p}) = {}^{W}p_c^z \tag{13}$$

The world model is represented by a Signed Distance Field (SDF) (see Section 2.3.2). Ground collisions are considered by modeling the underside of the robot with simple geometric shapes. These shapes are evenly sampled with an increased sampling density for tracks (Figure 9). For each sample point s_i a constraint $b_i(\mathbf{p})$ is added to the optimization problem to prevent the robot from moving through the ground:

$$b_i(\mathbf{p}) = \Phi({}^{W}s_i) > 0 \tag{14}$$

The function $\Phi(\cdot)$ evaluates the SDF. It returns the distance to the closest surface with positive values outside and negative values inside of objects. Therefore, the SDF-value has to be greater than zero for each point.

The free optimization parameter $\mathbf{p}$ determines the robot pose ${}^{W}T_B^{(k)}$ and therefore the location of COM ${}^{W}p_c^{(k)}$ and sampling points ${}^{W}s_i^{(k)}$. The estimation is divided into a *falling* and a *tipping* step. In the following, the respective free parameter formulation is detailed.

Falling phase

The falling phase estimates the pose of the robot until first contact (free fall). In this case, the free optimization parameter $\mathbf{p}$ is directly the z-component of the COM:

$$\mathbf{p} = {}^{W}p_c^z \tag{15}$$

To compute the position of the sampling points, we first compute the initial location of the COM ${}^{W}T_C^{(0)} = {}^{W}T_B^{(0)B}T_C$ using the initial transformation of the robot base link. ${}^{W}T_C^{(1)}$ is now computed by replacing the z-component of the translation with p and ${}^{W}T_B^{(1)} = {}^{W}T_C^{(1)C}T_B$.

Next, ${}^{W}s_i^{(1)}$ can be computed:

$$ {}^{W}s_i^{(1)} = {}^{W}T_B^{(1)B} s_i \tag{16} $$

After optimization, the iteration counter is increased by one $k := 1$.

Tipping phase

After the falling phase, the robot is in contact with the ground but usually not stable. In the tipping phase, the robot is repeatedly rotated about the least stable axis until it is in a stable position. To compute the next tipover axis, we require the current contact points. All sampling points s_i with an SDF-value smaller than a threshold ϵ are considered contact points:

$$ CP = \left\{ {}^{W}s_i^{(k)} \middle| |\Phi({}^{W}s_i^{(k)})| < \epsilon \right\}, \quad \text{for } i = 1, ..., N_s \tag{17} $$

N_s refers to the number of sampling points and CP to the set of contact points. The contact points are then used to compute the tipover axis. There are three possible contact configurations.

Point Contact: In case of one contact $CP = \{p_1\}$, the robot will rotate around a vector v_r orthogonal to the normalized gravity vector $g = (0, 0, -1)^T$ and the line connecting contact point and COM $v_{\bar{pc}} = p_1 - {}^{W}p_c^{(k)}$:

$$ v_r = g \times v_{\bar{pc}} \tag{18} $$

Line Contact: For more than one contact, we distinguish whether the contact points form a single line or span a polygon. In case of two contact points $CP = \{p_1, p_2\}$, only a line contact is possible. The rotation axis is given by $v_r = p_2 - p_1$. If the set of contact points contains more points $CP = \{p_1, ..., p_n\}$, we need to evaluate if they approximate a line. First, the distance d_i of every point $p_i \in CP$ to the line defined by the first two contact points p_1 and p_2 is computed:

$$ d_i = \frac{\|(p_i - p_1) \times (p_i - p_2)\|}{\|p_2 - p_1\|}, \quad \text{for } i = 1, ..., |CP| \tag{19} $$

If the distance is below a certain threshold for every point, they form approximately a line. In this case, the rotation axis is given by the line connecting the first two contact points $v_r = p_2 - p_1$.

Polygonal Contact: In the final case, multiple contact points form a support polygon. To determine the polygon, only the convex hull of the contact points is relevant. It is determined by projecting all points onto the xy-plane and computing the 2D convex hull. Afterward, the force-angle stability margin (see Section 3.1.1) is computed using the respective 3D points of the convex hull. If the stability margin β is greater than 0 the position is stable and the algorithm terminates here. Otherwise, the rotation axis v_r corresponds to the potential tipover axis a_i with the least stability.

Next, the robot is rotated around v_r while again minimizing ${}^{W}p_c^z$. The free variable is the rotation angle $\mathbf{p} = \alpha$. For simplification, a frame F is defined on v_r, given by ${}^{W}T_F$. The x-axis of the new frame is aligned with the rotation axis, so only rotation around x (roll) has to be applied. The location of the COM is transformed to this new frame ${}^{F}p_c = {}^{F}T_W {}^{W}T_B^{(k)B} p_c$ by using the latest pose estimate.

Afterward, ${}^{W}p_c^{(k+1)}$ is computed by applying the rotation and transforming to world:

$$ {}^{W}p_c^{(k+1)} = {}^{W}T_F R_x(\alpha)\, {}^{F}p_c \tag{20} $$

The location of each sampling point is calculated in a similar manner:

$$ {}^{W}s_i^{(k+1)} = {}^{W}T_F R_x(\alpha)\, {}^{F}T_B\, {}^{B}s_i \tag{21} $$

The next solution ${}^{W}T_B^{(k+1)}$ is now given by ${}^{W}T_B^{(k+1)} = {}^{W}T_F R_x(\alpha)\, {}^{F}T_B$. After optimization, the iteration counter is increased $k := k+1$.

The next iteration starts again with a tipping step by estimating the contact points. This process is repeated until the robot is in a stable position.

The proposed approach has several advantages. Because it does not rely on simulating time steps, it is fast. Moreover, unlike physics-based algorithms, the initial robot pose ${}^{W}T_B^{(0)}$ is not required to be above the ground. The estimation still works when the start estimate is in contact or even below the ground.

Algorithm 1 Pseudo-code of the optimization-based contact estimation.

```
function ContactEstimation(W T_B^(0))
    W T_B^(1) = fallingStep(W T_B^(0))          ⇒ move along gravity vector
    k := 1
    repeat
        CP = computeContactPoints(W T_B^(k))
        if isPoint(CP) then
            tipping_axis = g × (p_1 − W p_c^(k))
            stable = False
        else if isLine(CP) then
            tipping_axis = p_2 − p_1
            stable = False
        else if isPolygon(CP) then
            support_polygon = computeConvexHull(CP)
            stable = checkStability(support_polygon)
            if not stable then
                tipping_axis = findLeastStableAxis(support_polygon)
            end if
        end if
        if not stable then
            W T_B^(k+1) = tippingStep(W T_B^(k), tipping_axis)   ⇒ rotate around tipping_axis
        end if
        k := k + 1
    until stable
    return W T_B^(k)
end function
```

4.2 Whole-Body Planner

The whole-body planner generates a motion plan to safely cross obstacles. The planner receives a path $\mathbf{s}$, defined by N_s equidistant waypoints. The number of points depends on the sampling resolution of the path. Each waypoint $s_i = (x, y, \psi)^T, i = 1, ..., N_s$ is defined by a 2D-position (x, y) in the xy-plane and a heading ψ.

At each point, the joint configuration is optimized to increase stability. The optimization only considers static stability, so quasi-static movement is assumed.

Stability Optimization

Goal of the optimization is the maximization of the robot stability by finding an optimal robot posture $\mathbf{p}_i^*$ at each point on the path s_i. The parameter vector $\mathbf{p} = \left(q_1, ..., q_{N_p}\right)$ specifies the joint angles of the robot, where N_p is the number of available DOF. It is evaluated using an appropriate objective function $\varphi(\mathbf{p})$, which has to be minimized:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \varphi(\mathbf{p}) \tag{22}$$

A straightforward choice is the force-angle stability margin presented in Section 3.1.1:

$$\varphi(\mathbf{p}) = -\beta(\mathbf{p}) = -\min\left(\beta_1(\mathbf{p}), ..., \beta_N(\mathbf{p})\right) \tag{23}$$

The joint configuration $\mathbf{p}$ directly influences the stability margin by moving the COM and changing the contact geometry with the ground when moving the flippers. Therefore, the estimated pose of the robot ${}^W T_B$ depends on $\mathbf{p}$ as well and is computed by the contact estimation presented in Section 4.1.

However, directly applying the force-angle stability margin yields multiple issues. First, because of the minimum-function, the stability margin is not differentiable everywhere. It also means that the stability depends only on the weakest tipover axis. If a DOF of $\mathbf{p}$ cannot influence the stability of this axis, it will be unconstrained regarding the optimization.

These issues are addressed by using a weighting function $w(\cdot)$ applied to the stability β_i of every tipover axis i (Equation 24). Now the stability of every axis contributes to the objective function and $\mathbf{p}$ is properly constrained.

$$\varphi(\mathbf{p}) = \sum_{i}^{N} w(\beta_i(\mathbf{p})) \tag{24}$$

In [9] a quadratic-inverse weighting function was used:

$$w(x) = \frac{1}{x^2} \tag{25}$$

The inverse favors high stabilities while the square punishes stabilities close to zero. However, this objective function also rewards negative values as would be the case for unstable postures. This is not a problem as long as the optimization is initialized in a stable configuration. If the optimization is gradient-based, it will stay inside the local minimum of the support polygon. But outside, the optimizer can minimize the stabilities to also get a small sum.

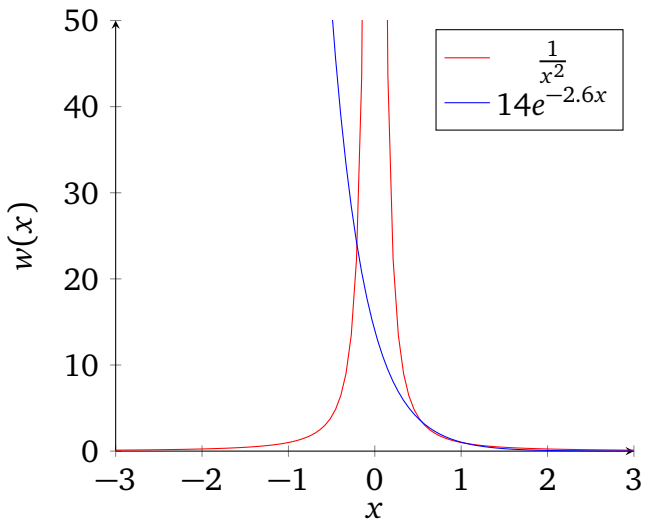


Figure 10: Comparison between the weighting functions in Equation 25 and 26. While the quadratic cost decreases for large negative values, the exponential cost is only increasing.

To also punish unstable postures, an exponential function is used instead (Equation 26) with the parameters a, b and c.

$$w(x) = ae^{-bx+c} \tag{26}$$

Figure 10 shows a comparison between the weighting functions in Equation 25 and 26. In the positive domain, the curves are similar and punish low stabilities. In the negative domain however, the function value of the quadratic weighting function goes down again, while the exponential still increases.

Additionally, the sum-formulation in Equation 24 is improved by normalizing by the total number of tipover axes N to make the magnitude independent of the number of ground contact points:

$$\varphi(\mathbf{p}) = \frac{\sum_i^N w(\beta_i(\mathbf{p}))}{N} \tag{27}$$

Environment Collision Avoidance

While we optimize for stability, collisions of the manipulator arm have to be taken into account to prevent damage to the robot. We consider a posture $\mathbf{p}$ to be valid, if it is not in collision with the environment. Therefore, the optimization problem is constrained.

The collision constraints with the environment are formulated using an SDF. Modeling environmental collisions is only necessary for the manipulator arm since the flipper is intended to be in contact with the ground. The arm consists of N_{arm} links. Each link of the robot arm l_1, .. $l_{N_{arm}}$ is approximated with multiple spheres (Figure 11). The number depends on the geometry of the link and is labeled with N_{l_i}. A sphere is defined by the center position relative to the attached link ${}^{l_i}c_{i,j}$ and a radius $r_{i,j}$. They are indexed with the corresponding link i and the sphere $j \in \{1, ..., N_{l_i}\}$.

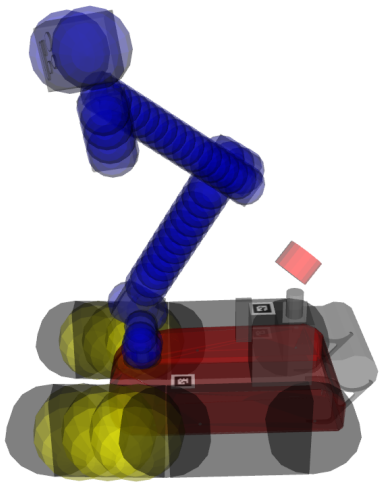


Figure 11: The robot geometry is approximated with multiple spheres for collision modeling. The blue spheres model the arm and are used for environment and self-collision. The flippers, visualized with yellow spheres, are only considered for self-collision.

One constraint per sphere is added to the optimization problem. Equation 28 shows the transformation of the sphere center from link frame l_i to world frame W.

$$ {}^{W}c_{i,j} = {}^{W}T_{B}\,{}^{B}T_{l_i}(\mathbf{p})\,{}^{l_i}c_{i,j} \tag{28} $$

${}^{W}T_{B}$ denotes the estimated world pose of the robot. The forward kinematics from base B to link i is given by ${}^{B}T_{l_i}(\mathbf{p})$ and depends on the joint configuration $\mathbf{p}$.

Because we use spheres for modeling, collision checking is straightforward. The constraint is formulated as:

$$ b_{i,j}(\mathbf{p}) = \Phi({}^{W}c_{i,j}) - r_{i,j} > 0 \tag{29} $$

The function $\Phi(\cdot)$ evaluates the SDF at the sphere center ${}^{W}c_{i,j}$ to get the distance to the closest obstacle. Next, the radius r of the respective sphere is subtracted from this distance to check for a collision.

Self-Collision Avoidance

Additionally to collisions with the environment, the posture $\mathbf{p}$ has to be free from self-collisions as well. To compute collisions efficiently, we distinguish between dynamic and static links. The position of dynamic links depends on the optimization parameters $\mathbf{p}$ whereas the static part is rigidly attached to the robot base.

Dynamic Collisions: The dynamic part consists of the robot arm as well as the flippers. Analogous to the environment collision avoidance, the dynamic links of the robot are sampled with spheres (Figure 11). One constraint per unique pair of spheres is added to the problem. First, the sphere centers are transformed to one common frame, the base link B. To prevent a collision, the distance between the centers minus their radii must be greater than zero:.

$$ \begin{aligned} &b_{i,j;m,n}(\mathbf{p}) = \|{}^{B}T_{l_i}(\mathbf{p})\,{}^{l_i}c_{i,j} - {}^{B}T_{l_m}(\mathbf{p})\,{}^{l_m}c_{m,n}\|_2 - r_{i,j} - r_{m,n} > 0 \\ &\text{for } i,m \in \{1,\ldots,N_{arm}\}, i \neq m; j \in \{1,\ldots,N_{l_i}\}; n \in \{1,\ldots,N_{l_m}\} \end{aligned} \tag{30} $$

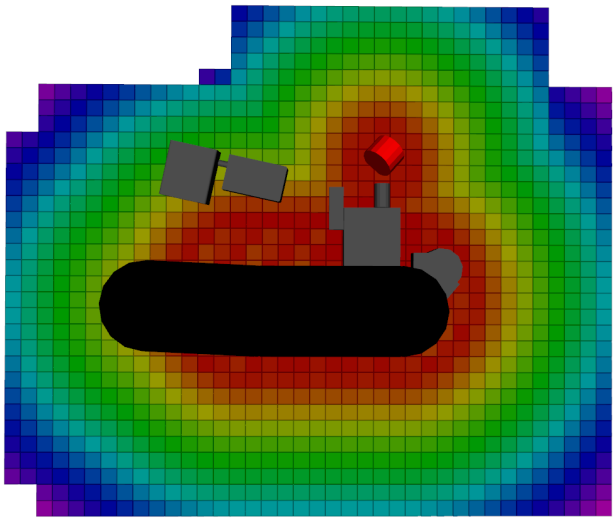


Figure 12: Slice of the robot SDF used for self-collision checking. Only static links with regard to the base link are considered. Voxels close to the robot are colored red.

$\|\cdot\|_2$ specifies the Euclidean norm. The first link is indexed with i and its spheres with j, the second link is indexed with m and its spheres with n.

To reduce the number of constraints, the Allowed Collision Matrix (ACM) is used. It stores information about allowed collisions, like adjacent joints, or links that can never collide.

Static Collisions: To reduce computational cost, static links are represented as an SDF. It is generated by computing the distances to the links in a grid around each link. An example can be seen in Figure 12.

Analogous to the environment collision check, the collision spheres of the dynamic links are then checked against this SDF (Equation 29).

Movement Penalty

With the formulation presented previously, the robot can do huge motions between waypoints even though the stability is only improved slightly. This results in a long execution duration of the final trajectory. Therefore a penalty term $r(p)$ for joint movement is added with the factor K_r:

$$\varphi(\mathbf{p}) = \frac{\sum_i^N w(\beta_i(\mathbf{p}))}{N} + K_r r(\mathbf{p}) \tag{31}$$

By increasing K_r and therefore the influence of the penalty term, the execution time is reduced. Since this also affects the stability, the selection of the parameter is a trade-off between stability and trajectory execution time.

A common pick for the penalty function is the Sum of Squared Differences (SSD) (Equation 32), which puts a quadratic penalty on the difference between the current joint configuration p and the previous q.

$$r_{SSD}(p) = \|p - q\|_2^2 \tag{32}$$

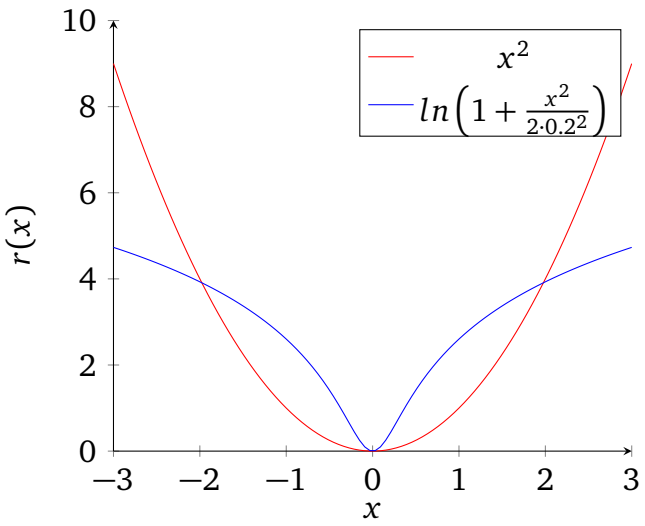


Figure 13: Comparison between squared differences and Lorentzian penalty function.

But sometimes it is reasonable to do huge motions if it is coupled with a meaningful stability increase. Therefore a "capped" penalty is a better choice. The Lorentzian function (Equation 33) fulfills this requirement. The shape of the curve can be further tuned with the parameter σ.

$$r_{lor}(p) = \ln\left(1 + \frac{(p-q)^2}{2\sigma^2}\right) \tag{33}$$

Figure 13 shows a comparison between SSD and Lorentzian.

4.3 Improvements

The previous section presented the basic approach for stability optimization. In the following, multiple aspects are modified to improve performance. First, the force-angle stability margin introduced in Section 3.1.1 is reformulated to be differentiable. Afterward, the stability optimization is divided into two separate problems so gradient-based algorithms can be utilized. Subsequently, a new objective function for the first sub-problem is introduced to replace the stability objective. Finally, a method for intentional tipping is presented.

4.3.1 Differentiable Stability Margin

As defined by Papadopoulos *et al.* [8], one of the components of the force-angle stability margin is the angle θ_i between the normalized projected net force vector $\hat{f}_i$ and the axis normal $\hat{l}_i$:

$$\theta_i = \sigma_i \cos^{-1}\left(\hat{f}_i \cdot \hat{l}_i\right) \tag{34}$$

The sign of the angle is determined by σ_i and depends on the force vector f_i. If f_i is directed inside the support polygon, the sign is positive and negative otherwise:

$$\sigma_i = \begin{cases} +1, & \left(\hat{f}_i \times \hat{l}_i\right) \cdot \hat{a} > 0 \\ -1, & otherwise \end{cases} \tag{35}$$

This formulation has the shortcoming of not being entirely differentiable because σ_i jumps from $+1$ to -1. The discontinuity makes the stability margin unsuitable for gradient-based optimization algorithms.

To make the stability margin differentiable and retain the sign-information, a signed-angle formulation is used:

$$\theta_i = \text{atan2}((\hat{f}_i \times \hat{l}_i) \cdot \hat{a}_i,\ \hat{l}_i \cdot \hat{f}_i) \tag{36}$$

Therefore, an explicit calculation of the sign σ_i is no longer necessary.

4.3.2 Splitting Flipper and Arm Optimization

The formulation of the objective function as presented in Section 4.2 is not entirely differentiable which can lead to poor performance and bad results of the optimization. While the stability margin itself is differentiable with the modifications shown in Section 4.3.1, the computation relies on the contact estimation (Section 4.1) to determine robot pose ${}^{W}T_{B}$ and support polygon (see Figure 14a). Due to its iterative implementation, it is not differentiable. This means to optimize the objective, only gradient-free solvers are suitable. While robust, these are generally much slower because of the high number of required function evaluations. Additionally, most solvers cannot deal with nonlinear constraints which are used for collision avoidance.

To address these issues, the optimization problem is split into separate optimizations for arm p_a and flipper joints p_f, with the flipper configuration being optimized first (see Figure 14b). The first advantage is, that the contact geometry with the ground only depends on the flipper position so only this optimization needs to evaluate the non-differentiable contact estimation and therefore use a gradient-free solver. Additionally, no collision constraints are needed, because the tracks are intended to be in contact with the ground.

For arm optimization, an efficient gradient-based solver can be used. We assumes, that the contact geometry does not change when moving the arm. This assumption holds true as long as the robot does not tip during optimization, which is discouraged by the objective function.

Moreover, a significant speed-up can be expected as gradient-based solvers generally converge faster and the comparably slow contact estimation is no longer needed for evaluating the objective function. Instead, the robot pose and support polygon are considered static and are a result of the flipper optimization.

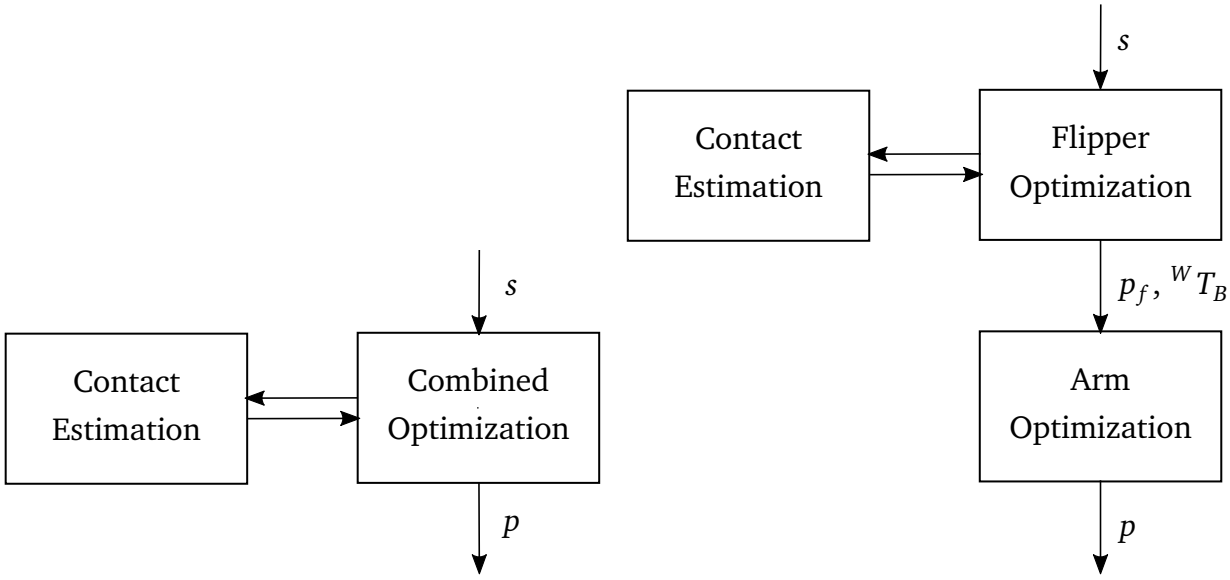
In the following sections, the original approach is referred to as *combined* while the modified version detailed in this section is referred to as *split*.

4.3.3 Flipper Optimization Heuristic

Splitting up the optimization raises the question if an objective function based on stability is still ideal for flipper optimization. While it increases the immediate stability, it does not necessarily provide a good basis for arm optimization. The stability highly depends on the COM, which is mainly influenced by the arm configuration. Because the arm configuration is optimized afterward, the COM is not in an ideal location, yet.

Therefore a heuristic is proposed, that is mainly independent of the COM location. Instead, it maximizes the support area A:

$$\varphi(p_f) = -A \tag{37}$$



(a) Combined Optimization **(b)** Split Optimization

Figure 14: Flowchart comparing the two optimization approaches. The combined method (Fig. 14a) optimizes flipper and arm configuration in one step and relies on the contact estimation. In contrast, the split method (Fig. 14b) optimizes the flipper configuration p_f first and uses the resulting robot pose ${}^W T_B$ to optimize the arm configuration p_a.

A big support area can be used by the arm optimization to increase the distance of the COM to the support polygon and therefore increase stability.

This is illustrated in Figure 15. The state before optimization can be seen in Figure 15a. The COM is at the back, so the frontal edge is stable and the robot won't perform movements that increase its stability further. If instead the heuristic is used, the flippers move downwards to increase the support area which can be used by the arm optimization to shift the COM forwards (Figure 15b).

Moreover, the contact information with the ground supplied by the contact estimation can be used to integrate more optimization criteria. To estimate the ground contact, the robot underside is sampled with points (see Section 4.1). This can be used to encourage contact of the tracks with the ground to improve traction. Additionally, we can penalize contacts with the chassis:

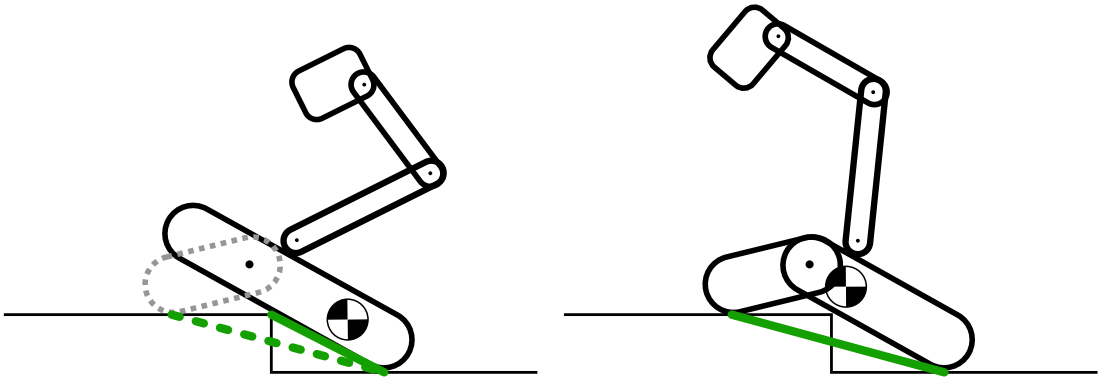
$$\varphi(p_f) = -A - K_t \frac{C_t}{N_t} + K_c \frac{C_c}{N_c} \tag{38}$$

K_t and K_c are weighting factors, C_t and C_c specify the number of track and chassis contacts respectively.

To make these criteria independent of sampling density, they are normalized by the total number of sampling points of tracks N_t and chassis N_c.

4.3.4 Intentional Tipping

A side effect of the changes made in Section 4.3.2 is, that it is no longer possible to intentionally tip the robot using the arm. Tipping is only possible by shifting the COM across one of the edges



(a) Before optimization **(b)** After optimization

Figure 15: The frontal edge is stable, so the robot won't move the flipper using the stability objective (Fig. 15a). By instead using the heuristic, the support area increases and the arm optimization can shift the COM to make the pose more stable (Fig. 15b).

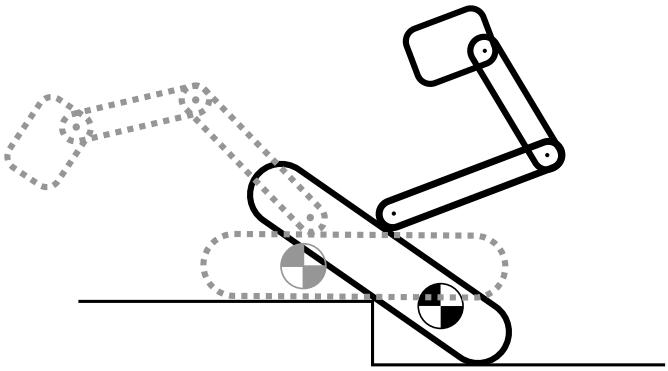


Figure 16: The robot can move its arm forward to get onto the step.

of the support polygon. But since the support polygon is not updated during arm optimization, the robot won't tip over regardless.

However, intentional tipping can be quite useful if a more stable state is only reachable by tipping, for example when getting onto a step (see Figure 16).

To achieve this behavior, a check if tipping is reasonable is added. This is done by checking the current support area A against a robot-dependent threshold A_{thres}. If the area is smaller than the threshold $A < A_{thres}$, the robot tries to tip. A small area indicates, that a large part of the robot is not touching the ground, therefore it might be reasonable in this situation to tip into a pose with more ground contact.

We assume, that tipover is only intended in driving direction, so forward edges that can be used for tipping have to be identified. For simplicity, the driving direction is defined as the x-direction of the robot base link. The contact points ${}^{W}p_{1,\dots,n}$ are first transformed to this link using the current pose estimate ${}^{W}T_{B}$:

$$ {}^{B}p_i = {}^{B}T_{W}\,{}^{W}p_i \tag{39} $$

Next, all possible tipover axes a_i are computed:

$$ {}^{B}a_i = {}^{B}p_{(i+1) \mod n} - {}^{B}p_i, \quad i = 1, \dots, n \tag{40} $$

For each axis, it is checked if they are facing forwards by computing the cross product with the x unit vector $e_x = (1,0,0)^T$:

$$v_i = e_x \times {}^B a_i \tag{41}$$

If the edge is facing forwards, the resulting vector points downwards. To also filter out edges, that are nearly parallel to x, the z-component has to be smaller than $\epsilon < 0$, where ϵ is a small number below zero:

$$F = \left\{ a_i | v_i^z < \epsilon \right\}, \quad \text{for } i = 1, ..., n \tag{42}$$

F labels the set of forward-facing edges. For each edge, the force-angle stability margin (Section 3.1.1) is computed. The least stable axis is used for tipping. It will be denoted with $a^\star$ and its two contact points $p_1^\star$ and $p_2^\star$.

To force tipping, a virtual COM is placed in front of $a^\star$. The current COM p_c is shifted orthogonally to the tipover axis in the xy-plane until it is slightly in front. For the following computations, $a^\star$, $p_2^\star$ and p_c are transformed to world and projected to the xy-plane by setting their z-component to zero. To make the equations easier to read, frames are not explicitly stated.

The COM is orthogonally projected onto the tipover axis to compute the normal vector l (Equation 43), with $\hat{a} = a/\|a\|$ and I being the 3x3 identity matrix.

$$l = (I - \hat{a}^\star \hat{a}^{\star T}) \left(p_2^\star - p_c \right) \tag{43}$$

Next, the COM is shifted along the normal vector l. To move p_c further across the tipover axis, a small distance $d = 0.05$ in the same direction is added by normalizing l with $\hat{l} = l/\|l\|$. Afterward, the original z-component is augmented, where $e_z = (0,0,1)^T$ is the z unit vector:

$${}^W p_{cv} = p_c + l + d\hat{l} + p_c^z e_z \tag{44}$$

Finally, the virtual COM p_{cv} is transformed back to the base frame:

$${}^B p_{cv} = {}^B T_W \, {}^W p_{cv} \tag{45}$$

At this point, it is unknown if the robot can actually move its arm in a way to reach the virtual COM and tip the robot. To update the support polygon, contact estimation is re-run with the virtual COM. Afterward, flipper and arm optimization are performed as usual. Note that the optimization starts in an unstable configuration because we used a virtual COM to compute the support polygon.

If the optimization finds a stable solution, the robot was able to tip into this position. Otherwise, the optimization is repeated with the original support polygon.

With this addition, the robot is able to shift its COM forwards to climb obstacles and find more stable configurations that are not reachable without tipping.

5 Implementation

This section will go into implementation details of the whole-body motion planning. First the framework and used libraries for world model integration and optimization are briefly presented. Subsequently, the motion planning pipeline is described.

5.1 ROS

This project is based on ROS (Robot Operating System). ROS is a flexible framework for developing and running software specifically designed for robots. It provides a collection of tools, libraries and core functionalities to make development easier and faster. Due to the modular design and encapsulation of projects into packages, libraries can be easily reused and shared with others.

Among other things, ROS provides the build system *catkin* with dependency management, inter-process communication through standardized messages and *rviz* for customizable data visualization. Figure 17 shows an example of the robot being visualized.

5.2 Voxblox

Voxblox [5] is a library for voxel-based mapping. It provides implementations for creating TSDF and ESDF. Because of the desirable properties of SDFs detailed in Section 2.3.2, they are used throughout this work to represent the environment for planning. The ESDF-variant is utilized because it provides the most accurate representation of distances to obstacles. Using trilinear interpolation, the grid can be accessed with sub-voxel accuracy.

An example of an ESDF created with voxblox can be seen in Figure 18. The range scans of a lidar were integrated into a TSDF. In a second processing step, the TSDF was converted to an ESDF.

5.3 Optimization

For optimization, the library NLopt [20] was used. It is a free collection of algorithms for nonlinear optimization. The algorithms cover local and global optimization as well as gradient-based and gradient-free methods for bound and constrained problems. The library provides a common interface, so testing different algorithms is very easy. A C++ interface and a ROS package are available.

In this work, contact estimation and whole-body planning are formulated as optimization problems. In the following, the used algorithms are briefly explained.

Contact Estimation

The contact estimation minimizes the z-coordinate of the COM (Equation 13) and uses nonlinear constraints for collision modeling (Equation 14). It is divided into two phases with different problem formulations for *falling* (Equation 15, 16) and *tipping* (Equation 20, 21).

The respective functions are automatically derived with infinitesimals using Ceres [21]. Nevertheless, the *falling* phase uses the gradient-free solver *Cobyla* [22]. The algorithm linearly approximates the objective function and constraints in each iteration and optimizes the parameters in a trust region. In practice, the gradient-free solver worked better due to the higher robustness against local minima.

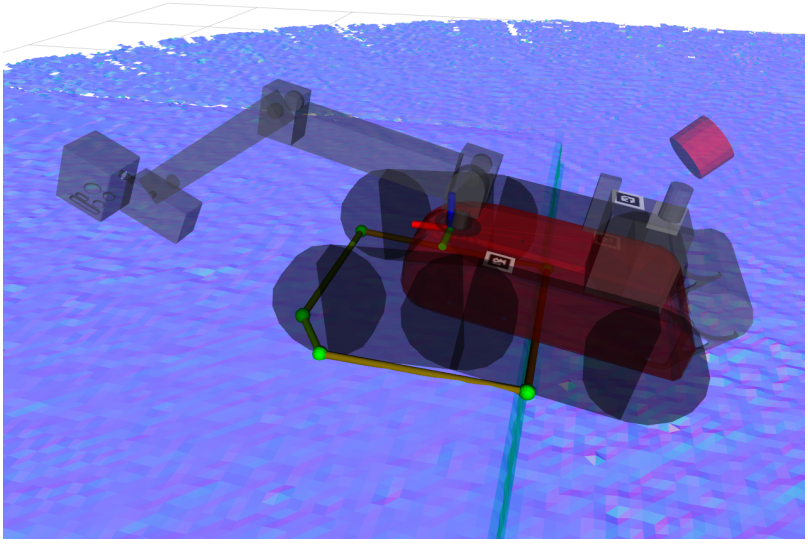


Figure 17: Visualization in rviz of the robot on the edge of a step. The robot model is displayed with transparency. The current COM location is visualized by the floating coordinate system just below the arm base. The green lines below the robot indicate the support polygon. The world is represented by a blue mesh, which was constructed from TSDF data.

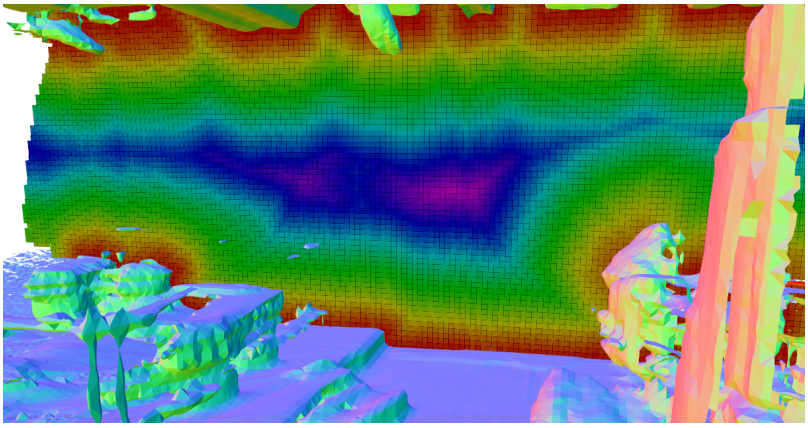


Figure 18: ESDF slice of a room. A small distance to the closest surface is colored red and far distances blue.

The *tipping* phase utilizes the gradient with Sequential Quadratic Programming (SQP) [23]. The algorithm approximates the objective function with a second order Taylor-expansion. The resulting quadratic programming problem is optimized with BFGS (Broyden–Fletcher–Goldfarb–Shanno) updates and linear first order approximation of the constraints.

Whole-Body Planning

Whole-body planning utilizes optimization to improve stability (Equation 22, 24). In the *combined* formulation, the contact estimation is needed to evaluate the objective function. Therefore, no derivative is available and a gradient-free algorithm is needed. Because the problem is also nonlinearly constrained, only *Cobyla* is applicable.

For the *split* problem, two different optimizers are used. The flipper optimization depends on the contact estimation, so no derivative is available. Because the ground contact is also very discontinuous, local gradient-free methods like *Nelder-Mead* fail to find good solutions. Therefore, the global algorithm *Direct-L* [24] is used with a fixed iteration limit of 10. It systematically divides the search space into hyperrectangles that become smaller with every iteration to find the global optimum.

Because of the assumptions made in Section 4.3.2, a gradient is available for arm optimization. To optimize the arm configuration, the algorithm method-of-moving-asymptotes (MMA) [25] is used. MMA forms a local and convex approximation at point x with an additional penalty term to make the approximation conservative. This is now easily solvable with a dual method to get a candidate x. If the approximation was indeed conservative, meaning an upper bound of the actual function value at the candidate point, the process is restarted at the candidate. Otherwise, the approximation is made more conservative by increasing the penalty term.

5.4 Trajectory Planning

After planning has finished, we have an optimal joint configuration p_i for every waypoint s_i on the path. To execute the joint motion, a trajectory connecting the configurations is needed first. For every configuration p_i, a trajectory to the subsequent configuration p_{i+1} is generated for $i = 1, ..., N_s - 1$.

The motion planning library *MoveIt!*[2] provides the necessary functionality. It creates ten waypoints for each state $p_{i,k},\ k = 1, ..., 10$ using an iterative sampling-based solver with $p_{i,1}$ being the original start state. The resulting trajectory is free of self-collision. While the library also provides functionality to avoid contacts with the environment, we cannot use it, because the robot is driving while executing the joint motion.

Before we can execute the trajectory, timestamps have to be generated. Each state $p_{i,k}, i = 1, ..., N_s; k = 1, ..., 10$ is assigned a time $t_{i,k}$ after which the state is reached. The first state $p_{1,1}$ is assigned $t_{1,1} = 0$. The time between waypoints is bound by the velocity and acceleration limits of the involved joints. The time-parameterization is also determined by *MoveIt!* with an iterative solver. An illustration of the whole pipeline can be seen in Figure 19.

5.5 Plan Execution

The plan is simultaneously executed by the vehicle controller and the joint controllers. The controllers are only synchronized by time.

[2] `https://moveit.ros.org/`

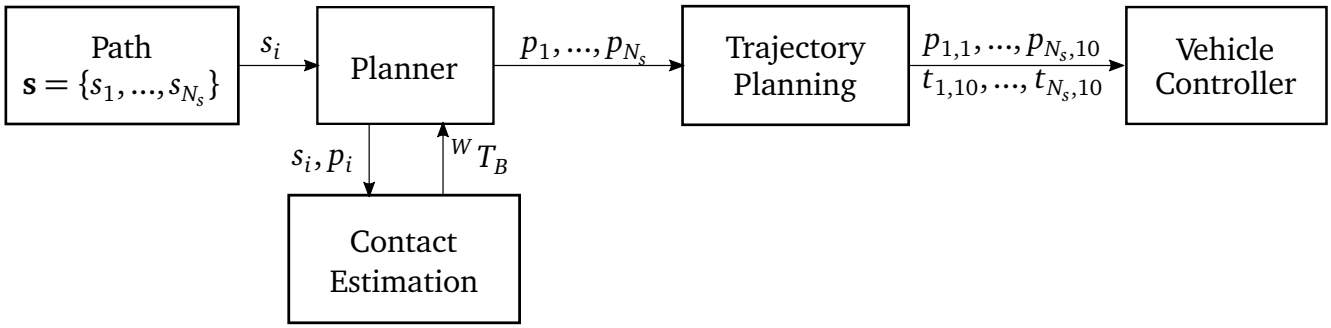


Figure 19: Diagram of the whole-body motion planning pipeline. The planner takes waypoints and optimizes the joint configuration. The trajectory planning connects the waypoints with trajectories and computes the time-parameterization.

The vehicle controller is responsible for following the path $\mathbf{s}$ and implemented as a standard carrot controller. The path is augmented with the respective time stamps of the joint trajectory, so waypoint s_i is reached at time $t_{i,1}$. Vehicle speed is adjusted by a P-controller to meet the time constraints.

The joints are controlled with PD-controllers with position interface. They receive the trajectory, defined by the joint positions $p_{i,k}$ and the timestamps $t_{i,k}$ for $i = 1, ..., N_s; k = 1, ..., 10$. In between the waypoints, the positions are cubically interpolated.

5.6 Other Libraries

In Section 4.1, *qhull*[3] was used to compute the convex hull of the contact points and the convex hull area. The library implements the Quickhull algorithm proposed by Barber *et al.* [26]. A ROS interface is available by using Point Cloud Library (PCL), a library for 2D/3D image and point cloud processing.

To check for collisions, the robot geometry is approximated with spheres in Section 4.2. Each link collision geometry is approximated with a cylinder using the ROS package *geometric_shapes*. To get the sphere decomposition, the cylinder is then evenly sampled with spheres at a fixed distance. This pipeline is part of *MoveIt!*.

Finally, the Rigid Body Dynamics Library (RBDL) was used to compute the COM. It is open-source and contains highly efficient code for forwards and inverse kinematics. For COM computation RBDL also provides the analytical gradient.

[3] `http://www.qhull.org/`

6 Evaluation

In this section, the proposed method for whole-body planning is evaluated. First, the contact estimation is tested by comparing it with data from the real robot. Next, the performance of the whole-body planner is evaluated in different scenarios in simulation and on the real robot.

As detailed in Section 1.3, tests were performed on the Hector Tracker, which is mostly identically constructed to the Argonaut Tracker.

6.1 Contact Estimation

A novel method for contact estimation based on iterative optimizations is proposed in Section 4.1. The prediction of ground contact points has to be precise so stability can be assessed accurately. Therefore, the prediction accuracy is evaluated by comparing with data recorded on the real robot.

The performance is evaluated by driving straight across multiple ramps with the real robot platform. The manipulator arm is folded into a compact driving position and joint configuration is not changed during the test. The reference pose of the robot is determined using a SLAM approach that fuses IMU and laser scan data. The pose is recorded approximately every 0.05 m along the path. Afterward, the generated map data is used to predict the robot pose along the same path using the contact estimation. The contact estimation is initialized with the first measured pose of the robot. Afterward, roll, pitch and z are estimated by providing the current x, y and yaw values.

A visualization of the generated map is shown in Figure 20 and the results are depicted in Figure 21. The graph compares measured with predicted pose. It shows, that the prediction follows the measurement very closely. This is also confirmed by the error statistics in Table 1. The mean absolute difference (MD) of the angles is below 0.03 rad (∼1.72°) and below 0.02 m for the estimated height. It can be concluded, that the contact estimation provides an accurate prediction of the robot pose.

As the contact estimation is part of the stability objective, it is crucial to evaluate the execution time to ensure that optimization finishes in a reasonable time span. Therefore, measurements of the time needed for contact estimation are also performed. The evaluation is executed on an *Intel Core i7-4710HQ CPU @ 2.50GHz*. The 72 estimations that were performed during this test needed an average time of 11.66 ms with a maximum of 34.37 ms.

As mentioned in Section 5.3, flipper optimization has a fixed limit of 10 iterations. Therefore, the contact estimation is executed 10 times per optimization. Using the average evaluation time of 11.66 ms, one flipper optimization takes 116.6 ms. On a path of 1 m, the joint configuration is optimized 50 times using a sampling resolution of 0.02 m. The flipper optimization along this path therefore takes 5830 ms or 5.83 s which is a reasonable time span.

Error	Roll (rad)	Pitch (rad)	z (m)
MD	0.0165	0.0246	0.0142
RMS	0.0228	0.0306	0.0181

Table 1: Estimation errors of roll, pitch and z. The first row shows the mean absolute difference (MD) and the second the root mean square (RMS)

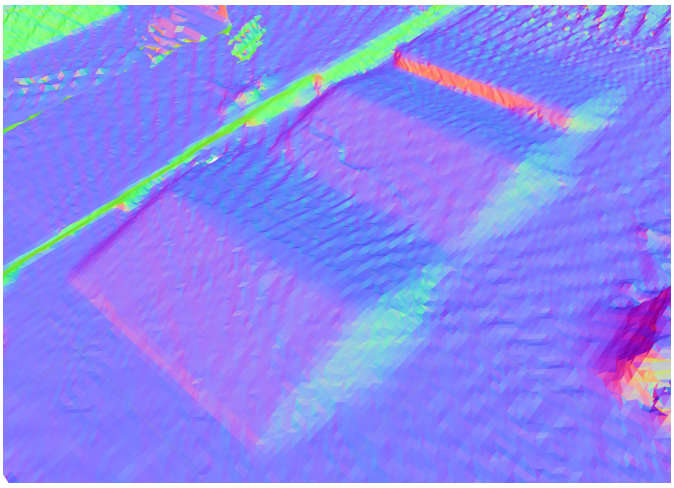
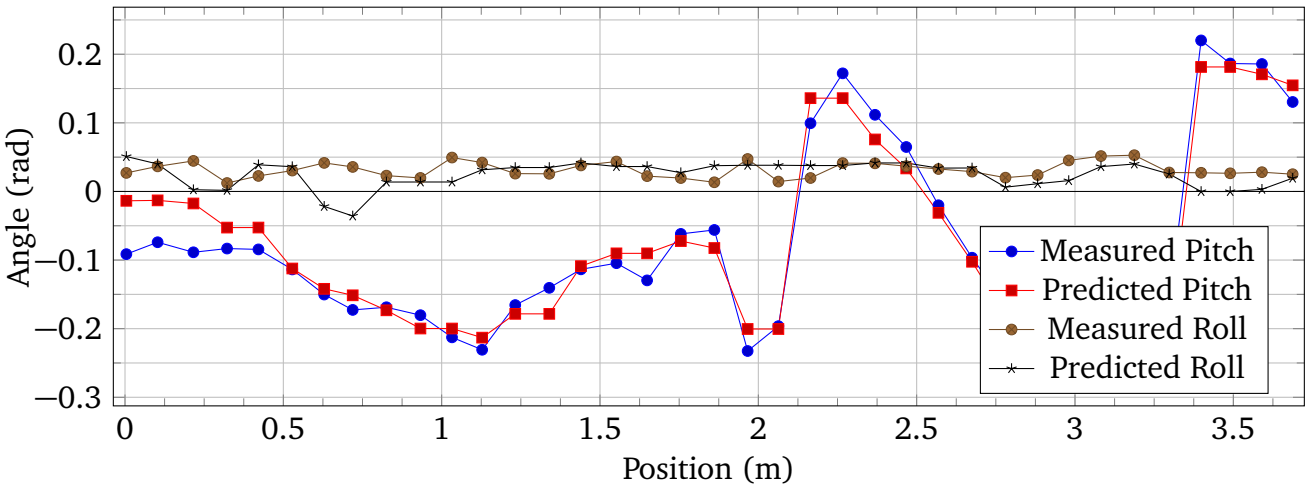
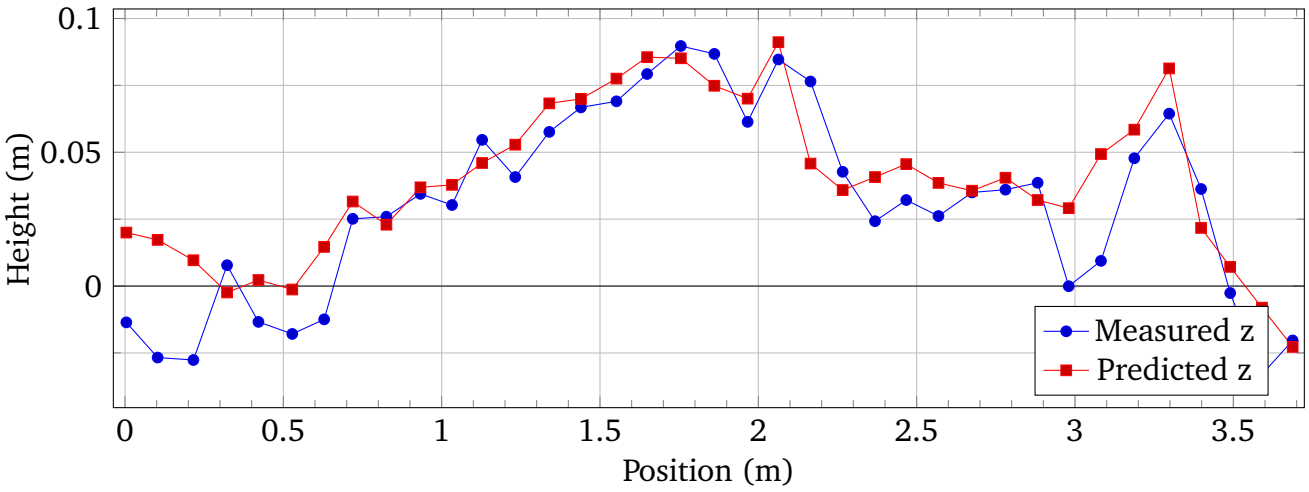


Figure 20: Recorded map data of the evaluation setup for contact estimation. The robot had to drive across multiple ramps. The mesh was generated from the TSDF with voxblox.



(a) Comparison of roll and pitch



(b) Comparison of robot height

Figure 21: Comparison of measured pose of the real robot with pose predicted by the contact estimation along the same path.

6.2 Whole-Body Planning

In the following section, the proposed whole-body planning approach is tested in simulation and on the real robot platform.

The robot has to cross different obstacles without tipover. The path across the obstacles is straight and defined in the xy-plane. It is sampled with a resolution of 0.02 m.

The map of the obstacles was obtained beforehand and is considered given regarding the whole-body planning. It is represented with an ESDF that was generated with the Velodyne VLP-16 laser scanner and a depth camera. To get a complete map, the robot drove around the obstacle and also extended its arm to look on top. The map is only generated before planning, no updates are made during planning or execution.

The parameters used for optimization can be found in Table 2. While the robot arm offers five DOF, only three of them are free parameters of the optimization. The first joint in the base, as well as the last yaw joint of the sensor head, are fixed at zero. Additionally, the position of the 1-DOF flipper is optimized as well.

During execution of the planned trajectory, the actual stability is measured with a constant frequency of 0.5 Hz. In simulation, the ground truth is used as robot pose. On the real robot, a fused estimate between track odometry, IMU and laser scan matching is used.

To evaluate the performance, stability during execution is compared. The stability refers to the force-angle stability margin presented in Section 3.1.1, which is the minimum stability of the tipover axes (Equation 12).

The magnitude of the stability has no direct physical interpretation but can be seen as robustness against disturbances. Therefore, it is used to compare different approaches. A positive stability means, that the robot is stable. Critical stability is reached at zero and a negative value indicates, that the robot is unstable and that tipping is in progress.

Exponential function parameters	
a	14
b	−2.6
c	0.3
Movement Penalty Arm	
K_r	0.0025
σ	0.2
Movement Penalty Flipper	
K_r	0.2
σ	0.2
Flipper Heuristic	
K_t	4
K_c	3
Support Area Limit	
A_{thres}	0.35

Table 2: Table of parameters used for evaluation.

6.2.1 Simulation

First, the simulation is used to evaluate the improvements proposed in Section 4.3. Subsequently, the whole-body motion planning with improvements is compared against an existing reference implementation.

Gazebo

The robot is simulated using the open-source robotics simulator Gazebo[4]. It natively integrates with ROS which allows using the exact same software for simulation and the real robot.

Gazebo provides a rigid body physics engine, sensor simulation and 3D visualization. It is easily expendable with plugins. A wide variety is already available to simulate sensors like cameras and laser scanners or different locomotion types like wheels or tracks.

The tracks of the Hector Tracker are simulated using a modified version of the *SimpleTrackedVehiclePlugin* to support the flippers. The plugin implements a fast approximation proposed by Pecka *et al.* [27] that yields plausible results.

Testing Scenarios

In simulation, four scenarios are used for testing: step, ramp, cinder block and an asymmetrical step. The scenarios were selected to be an accurate representation of expected obstacles. The robot starts at $x = 0$, facing the obstacle. Obstacles are located along the x-axis.

The first testing scenario is a simple step with a height of 0.15 m and length of 1 m. The robot has to get onto the step and down on the other side. The edges are at $x = 0.7\,\text{m}$ and $x = 1.7\,\text{m}$. Figure 22a shows the step in Gazebo.

The next testing scenario consists of two ramps connected by an elevated floor. The ramps have an inclination of about 40°. The scenario can be seen in Figure 22b.

In the third scenario, the robot has to cross a cinder block. It has a height of 0.14 m and width of 0.2 m. The front edge is located at $x = 0.7\,\text{m}$. It can be seen in Figure 22c.

The last testing scenario is an asymmetrical step. It is identical to the step from the first scenario but shifted along the y-axis with 0.1 m distance to the x-axis. Figure 22d shows the setup.

Flipper Heuristic

Section 4.3.3 introduced a new objective function for flipper optimization as an alternative to direct stability optimization. In the following, the two approaches are compared by planning a path in the step scenario with each method. To eliminate the influence of movement penalty on the stability, the respective weight is set to zero for the flipper joint only.

A graph of the stability can be seen in Figure 23. It shows, that the heuristic achieves a comparable stability to the original objective. This is confirmed when looking at the average stability, which is 2.3 for the stability objective and 2.26 for the heuristic.

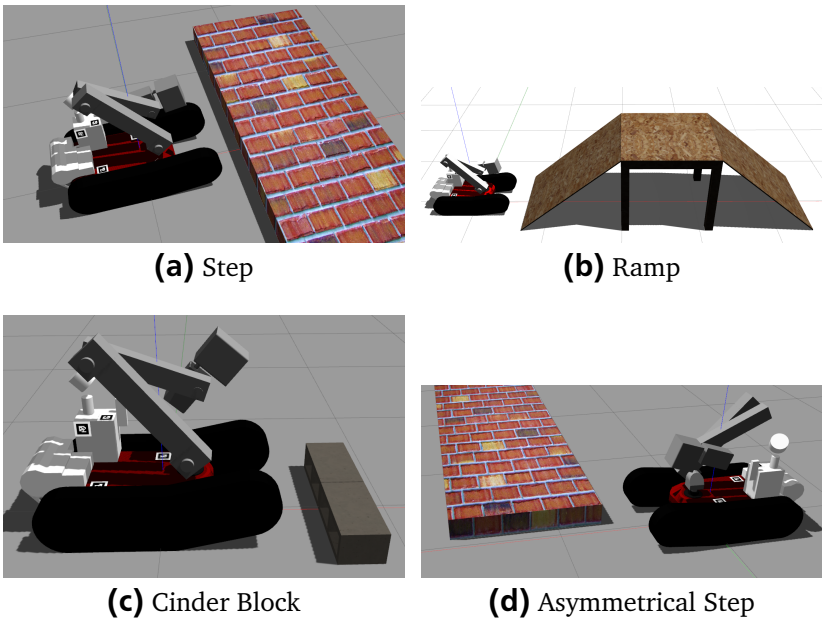
At the 1.1 m mark, the stability objective seems to be better. However, the robot just moves earlier onto the step using the heuristic.

A disadvantage of the stability objective becomes obvious when looking at the generated postures directly. Figure 24 shows the robot on the step. The back part of the tracks lost contact with the ground which leads to a reduced traction. This solution is possible with the stability

[4] `http://gazebosim.org/`

objective because the back axis is still very stable and therefore does not contribute to the cost sum.

Altogether, the heuristic is a better pick for the flipper objective function. It achieves similar stability and can be adapted with secondary objectives like traction.



(a) Step **(b)** Ramp

(c) Cinder Block **(d)** Asymmetrical Step

Figure 22: The four evaluation scenarios in Gazebo.

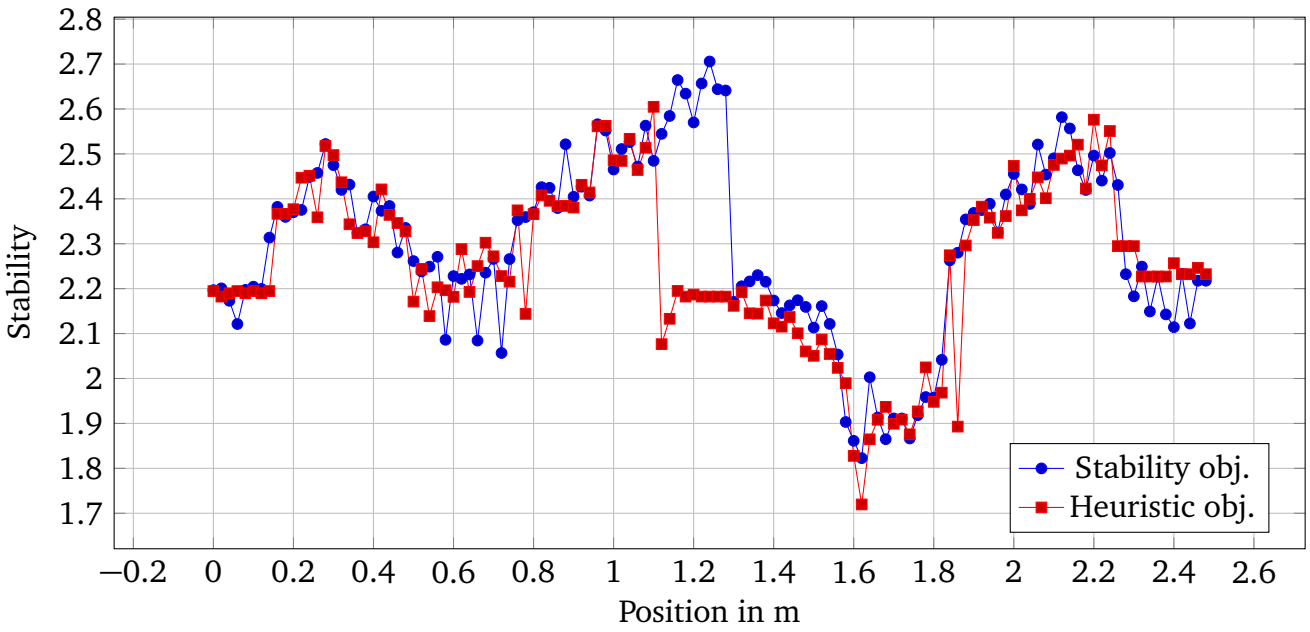


Figure 23: Comparison between heuristic and stability objective functions.

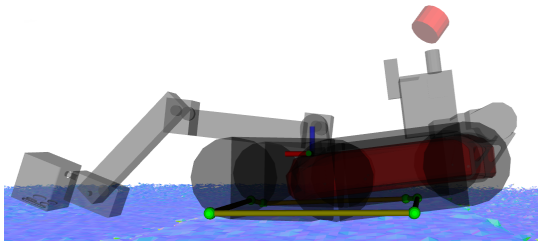


Figure 24: Visualization of the robot during planning using the stability objective. The backside is lifted because the corresponding tipover axis is still stable and therefore does not contribute to the objective cost.

Split Optimization

After introducing the original optimization problem in Section 4.2, a modified version that splits optimization of flipper and arm joints is proposed in Section 4.3.2. This modification is tested by planning a path across the step. To eliminate the effects of movement penalty on stability, it was omitted for all joints in both approaches.

A graph comparing the stability can be seen in Figure 25. The combined optimization achieves a higher stability most of the time. This is also confirmed by the average stability which is 2.29 for combined and 2.28 for split optimization.

However, looking at the minimum stability, the split approach performed much better with 1.7 against 0.79. The combined optimization failed to find a good solution at the edge of the step as can be seen in Figure 26. Instead of moving the arm forward to tip onto the step, the COM remained in the current support polygon. The optimization was therefore stuck in a local optimum. In the split optimization approach, this is addressed by intentional tipping detailed in Section 4.3.4.

Moreover, the combined optimization sometimes failed to meet the constraints. This resulted in collisions with the environment and the robot itself.

Next, the average objective function evaluation time is compared. As can be seen in Figure 27 the split approach is much faster. This is due to the fact, that the combined approach needs the contact estimation for every evaluation whereas the split approach needs it only for few flipper optimization steps.

These results show, that the split optimization formulation should be preferred over the combined formulation.

Comparison with Reference

In this part of the evaluation, the proposed approach is compared against an existing reference implementation. During the ARGOS Challenge (Section 1.2) a simple behavior for obstacle crossing was developed. It switches between predefined joint configurations based on IMU (Inertial Measurement Unit) feedback. The implementation is specific to the Argonaut Tracker. The approach worked well in the past but is not very generic.

In the following, we evaluate if the whole-body planning increases stability compared to the reference. Additionally, the prediction accuracy is assessed by comparing the predicted stabilities of the motion plan with the measured stabilities during execution. Furthermore, tipover has to be prevented in diverse scenarios to evaluate the flexibility.

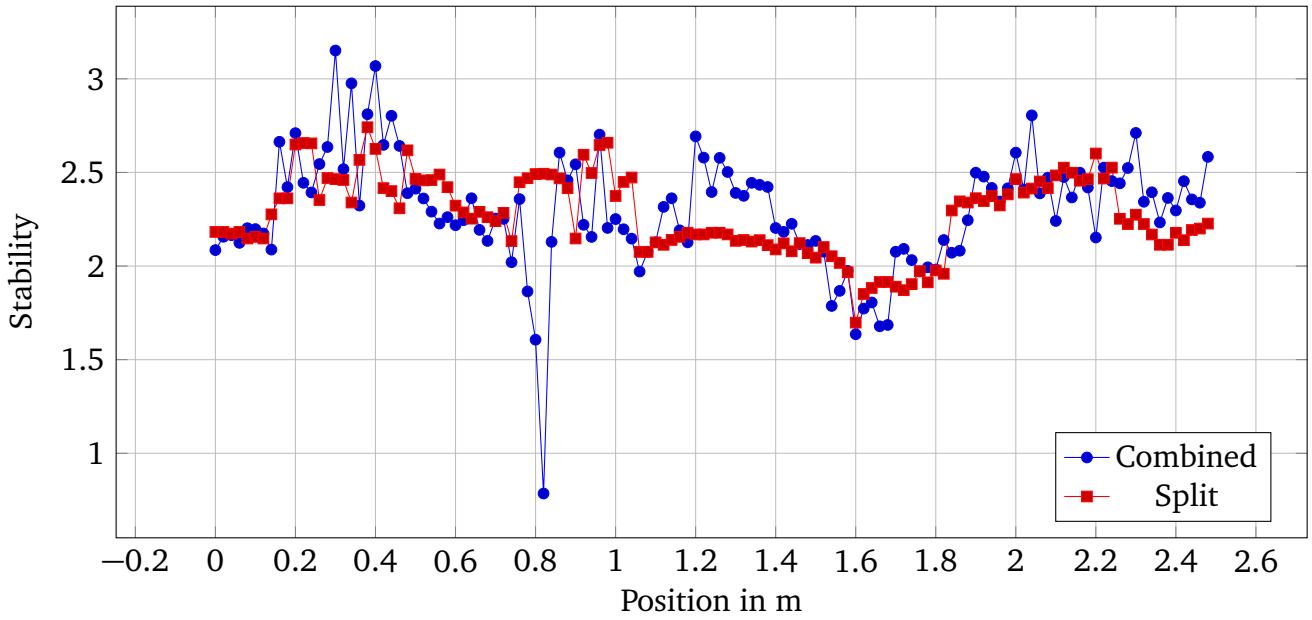


Figure 25: Stability comparison between split and combined optimization

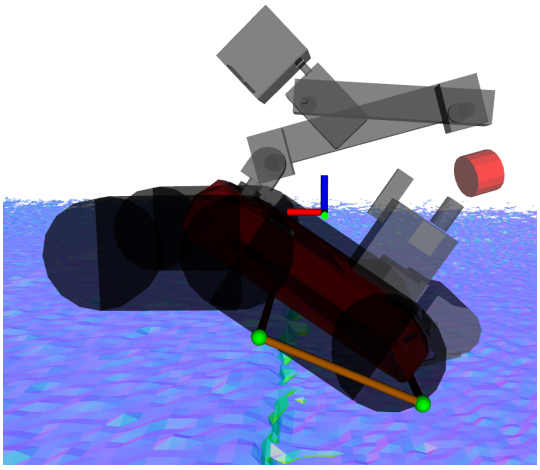


Figure 26: Robot at the edge of the step, flippers are not touching the ground. The combined optimization failed to find the better solution of tipping onto the step.

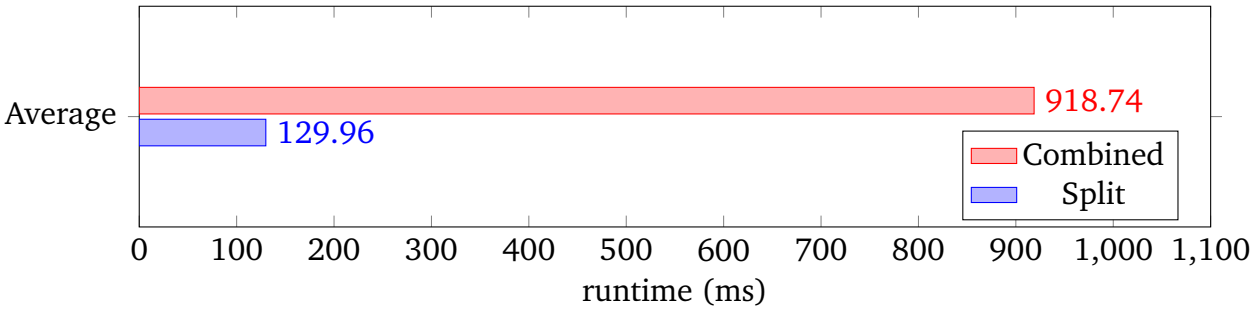


Figure 27: Runtime comparison between combined and split approach.

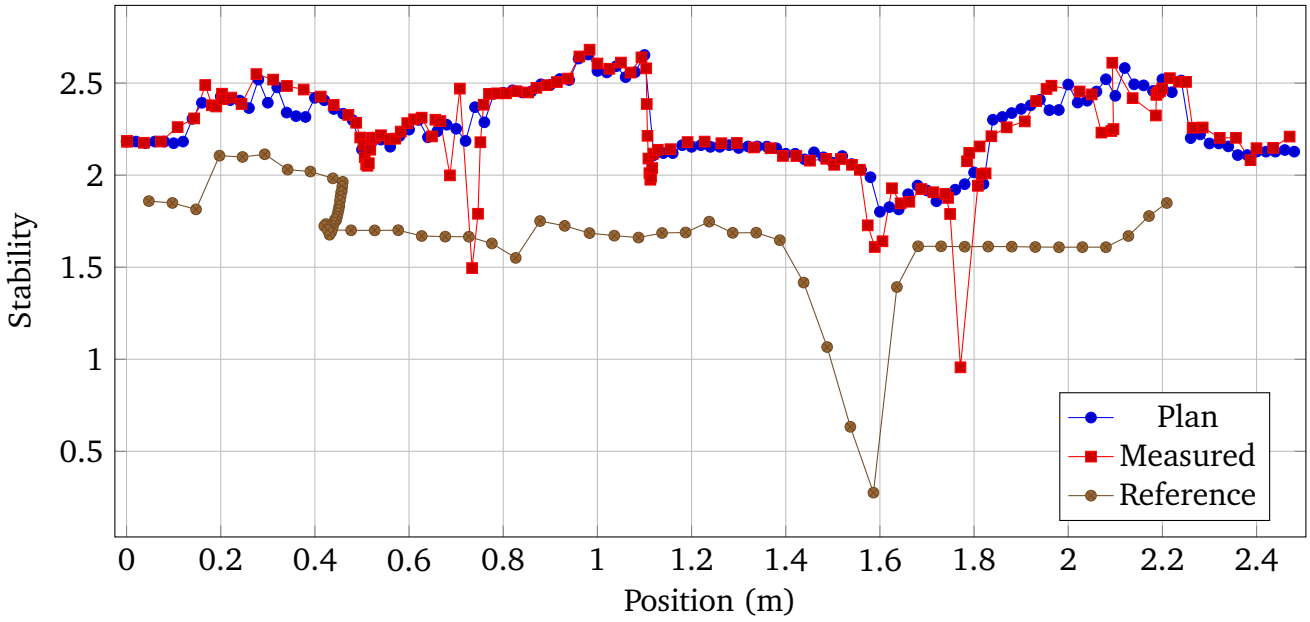


Figure 28: Stability graph of the step scenario. *Plan* labels the originally planned stability. *Measured* and *Reference* are the recorded stabilities during execution of the whole-body plan and reference implementation respectively.

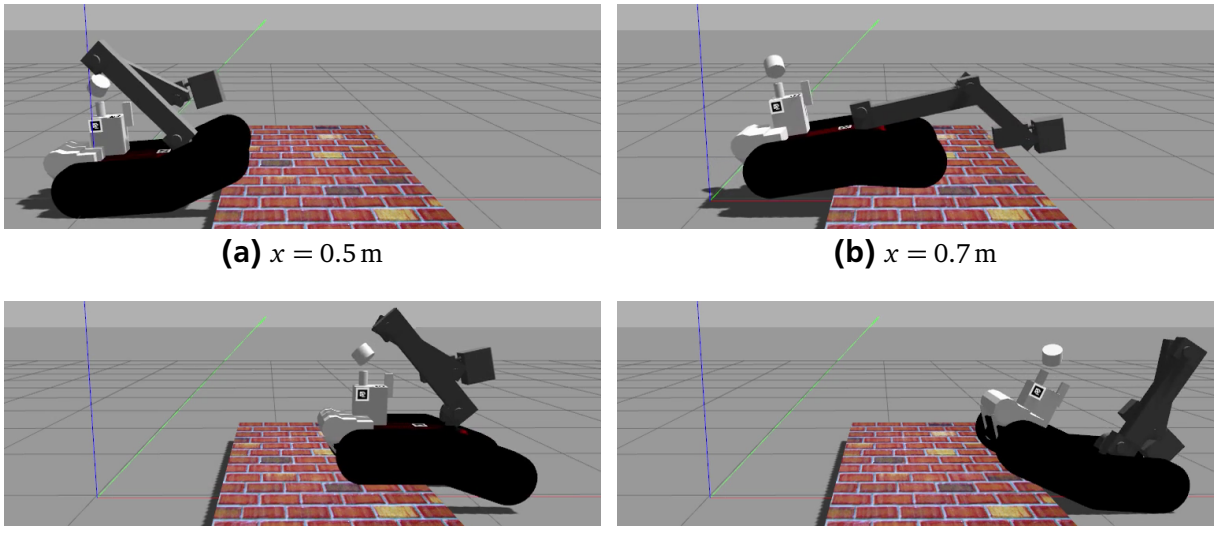
The whole-body planning is performed with split optimization as well as the heuristic for flippers. The tests are performed in all four scenarios by generating and executing a motion plan. During execution, the actually reached stabilities are measured.

Step: For the first test, the step scenario is used. The execution of the whole-body plan can be seen in Figure 29. The measured stability is compared to the original plan and the reference in Figure 28. The measured stabilities follow the plan very closely. This indicates, that the planning accurately predicts the robot pose and contact points. The only big difference can be seen at $x = 1.77\,\text{m}$ where the measured stability drops to 1. Because of inaccuracies in the map data and contact estimation, the robot lifts the flippers too early and tips down the step. The robot just before tipping can be seen in Figure 29c.

The reference falls behind the whole-body approach in terms of stability. This can also be seen when looking at the average stability, which is 1.7 for the reference and 2.23 for whole-body planning. Because the reference does not use map data, the flippers were lifted too early ($x = 0.4\,\text{m}$). When getting onto the step, the arm movement was not adapted to the step height. For the same reason, the robot became almost unstable when leaving the step at $x = 1.6\,\text{m}$.

Another important factor is the time needed to cross the obstacle. These were compared in Figure 36. The step obstacle is crossed when the robot reaches $x = 2.2\,\text{m}$. The reference only needs about half the time to get over the obstacle, because it only changes the robot posture at specific trigger points. This leads to overall fewer arm movements and therefore a faster execution.

Section 4.2 introduced a movement penalty to reduce the execution time of the whole-body plan. Because stability and execution speed is a trade-off, the current parameters are selected with a greater focus on stability. The time difference between the approaches could, therefore, be reduced by increasing the penalty.



(a) $x = 0.5\,\mathrm{m}$ **(b)** $x = 0.7\,\mathrm{m}$

(c) $x = 1.7\,\mathrm{m}$ **(d)** $x = 2\,\mathrm{m}$

Figure 29: Robot crossing steps using the whole-body planner.

Ramp: The next testing scenario is the ramp. The stability graph can be seen in Figure 30. The steep slopes proved to be a challenge for both approaches. As the negative stabilities of the reference implementation indicate, the robot failed to cross the ramp. It successfully managed to climb the first ramp, but at about $x = 3.25\,\mathrm{m}$ the robot fell forwards while going down the slope.

The whole-body approach successfully passed both ramps. Its solution can be seen in Figure 31. At two points, the execution differed from the plan. First, when transitioning from the first slope to the high ground, the robot tipped forwards slightly, as can be seen in Figure 31b. This resulted in a reduced stability at $x = 2\,\mathrm{m}$. Second, the stabilities from $x = 4.32\,\mathrm{m}$ to $x = 4.75\,\mathrm{m}$ are significantly lower than planned. When touching the ground with the flippers, the vehicle controller lost track and had to re-orientate.

As the reference failed to complete the obstacle, run times cannot be compared. The whole-body approach, however, took a long time with 148.5 s. Most time was spent during transitioning from high ground to the downwards slope at $x = 3.2\,\mathrm{m}$. The robot had to do huge arm movements to keep its balance. This shows, that driving down the slope frontally without risking tipover is a challenging task.

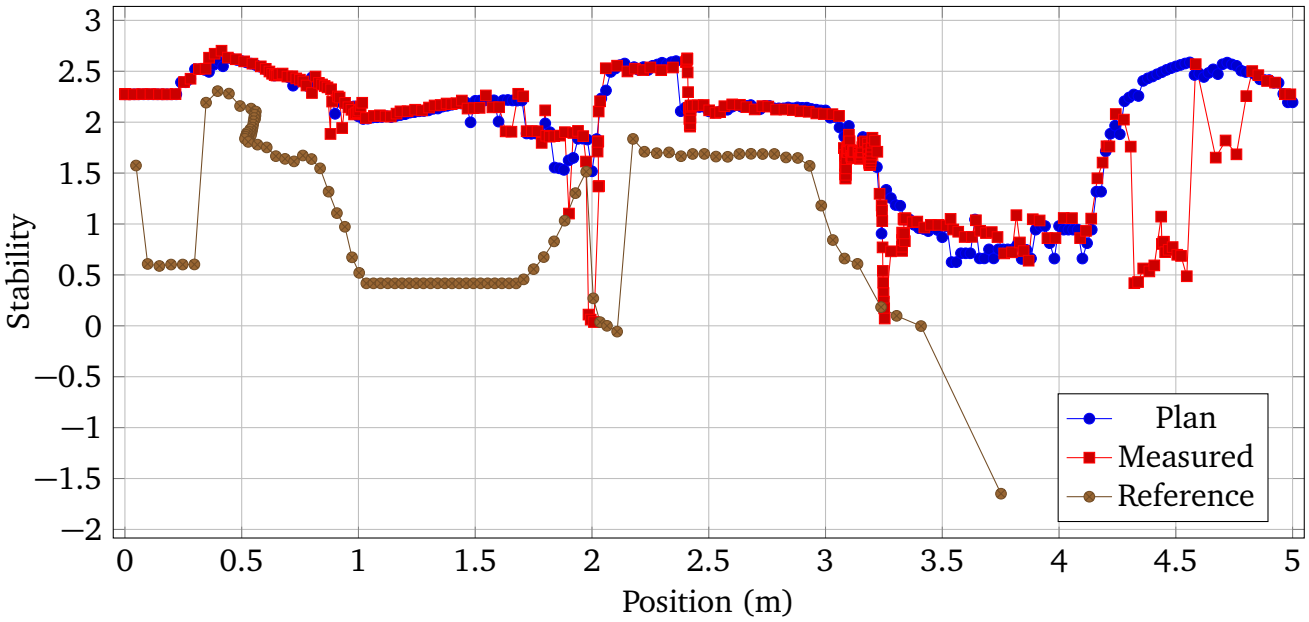
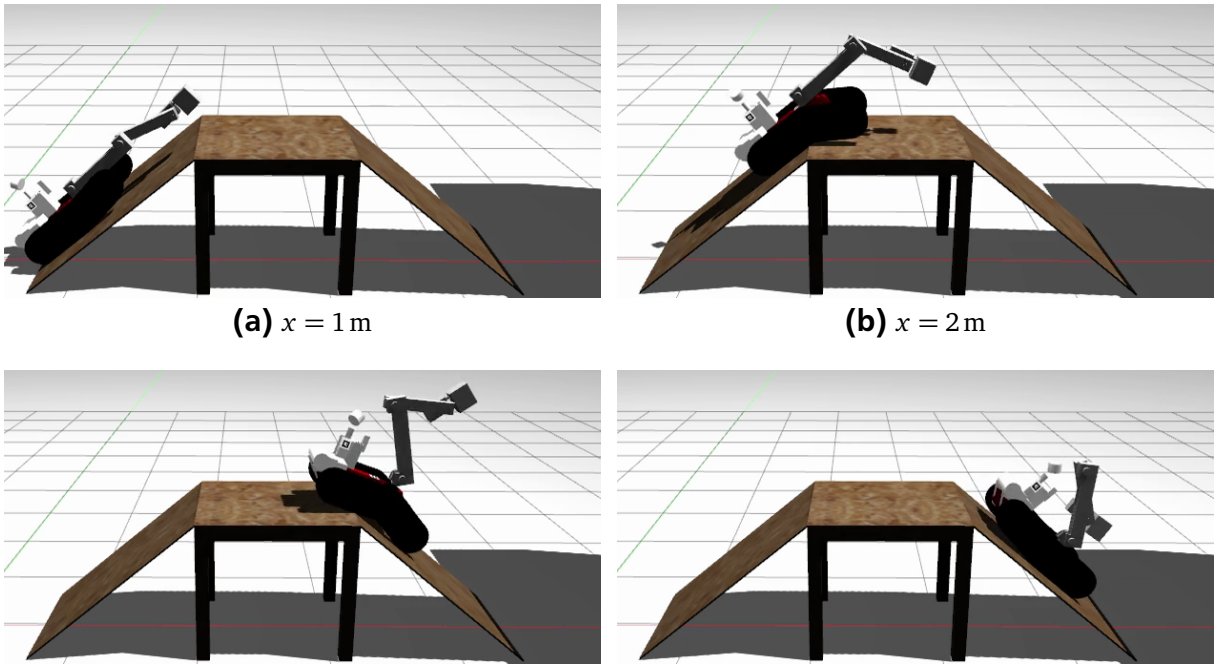


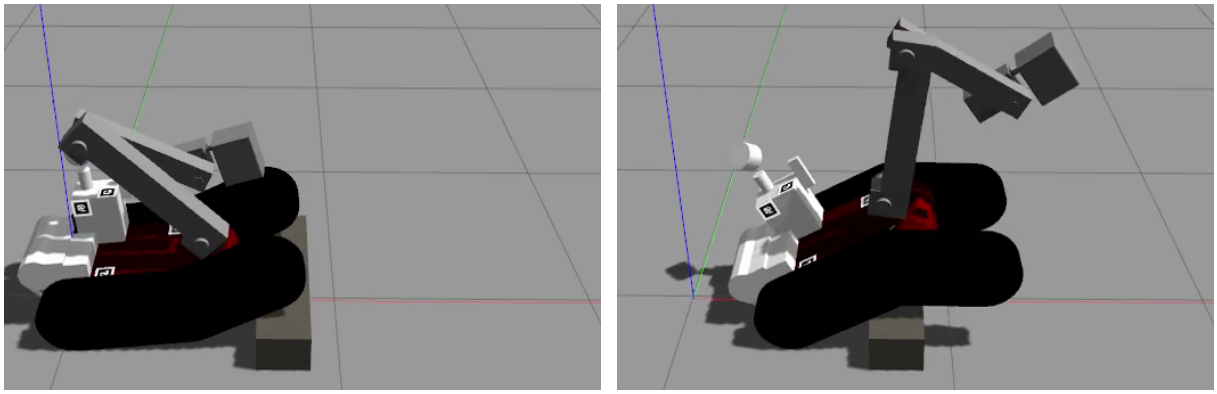
Figure 30: Stability comparison on the ramp scenario.



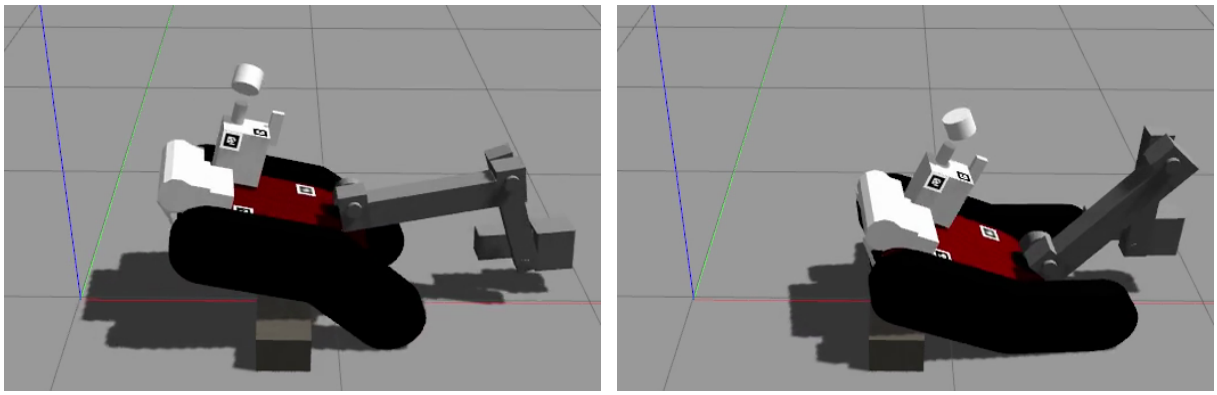
(a) $x = 1\,\mathrm{m}$ **(b)** $x = 2\,\mathrm{m}$

(c) $x = 3.25\,\mathrm{m}$ **(d)** $x = 3.8\,\mathrm{m}$

Figure 31: Robot crossing the ramps using the whole-body planner.



(a) $x = 0.6\,\mathrm{m}$ **(b)** $x = 0.7\,\mathrm{m}$



(c) $x = 0.8\,\mathrm{m}$ **(d)** $x = 1.3\,\mathrm{m}$

Figure 32: Robot crossing a cinder block using the whole-body planner.

Cinder Block: In the next scenario, the robot has to cross a cinder block. The execution of the whole-body plan can be seen in Figure 32 and the stability comparison in Figure 33. Both whole-body planning and reference implementation successfully manage to cross the obstacle. While the average stability of the whole-body approach is higher with 1.92 to 1.68, the robot had problems to safely get over the block. At about $x = 0.8\,\mathrm{m}$ the robot tipped over the obstacle by applying intentional tipping (see Section 4.3.4). Because the arm was moved forwards while also tipping forwards, the robot hit the ground with high velocity and the sensor head almost collided with the ground (see Figure 32c). The fast motion violated the quasi-static assumption, so the robot was close to tipover. While the reference implementation lets the robot tip at the same point, it does not have this problem because the arm is in a much higher configuration.

After $x = 1.15\,\mathrm{m}$ the vehicle controller had problems to track the remaining path when transitioning to the ground which explains the lowered stability. As the same issue occurred in the ramp scenario, it might be related to the vehicle controller implementation.

The times to cross the cinder block for both approaches are given in Figure 36. The obstacle is crossed at $x = 1.4\,\mathrm{m}$. As expected, the reference was much faster due to the reasons pointed out before.

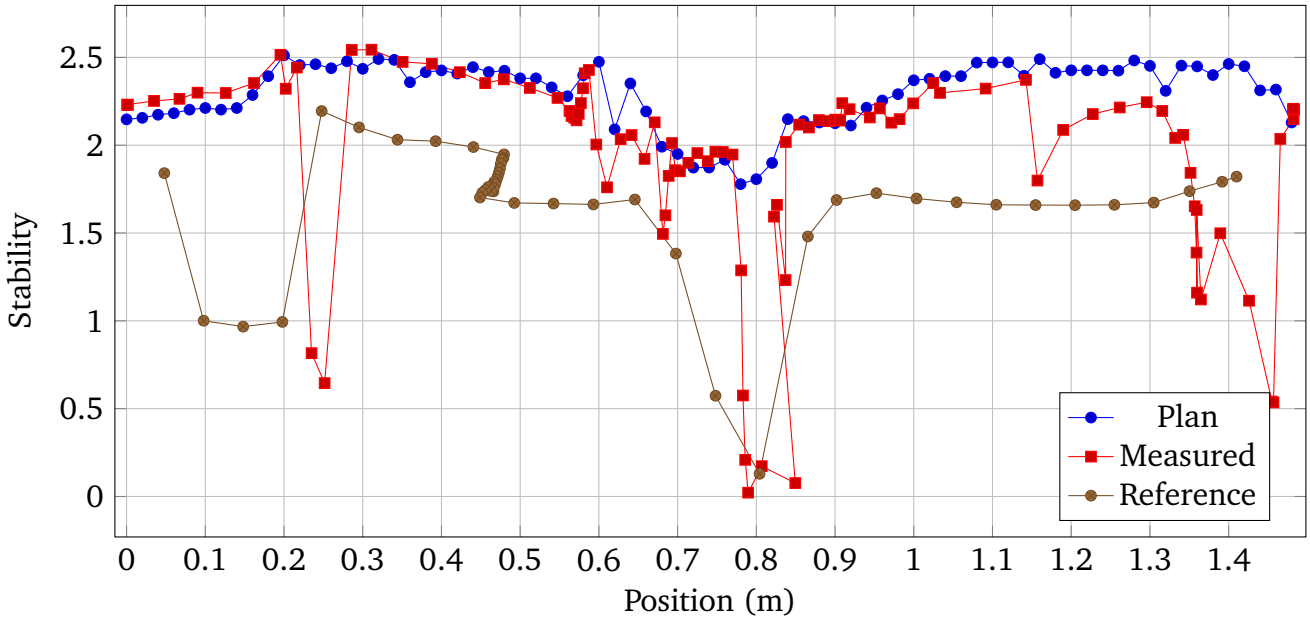


Figure 33: Stability comparison for crossing the cinder block.

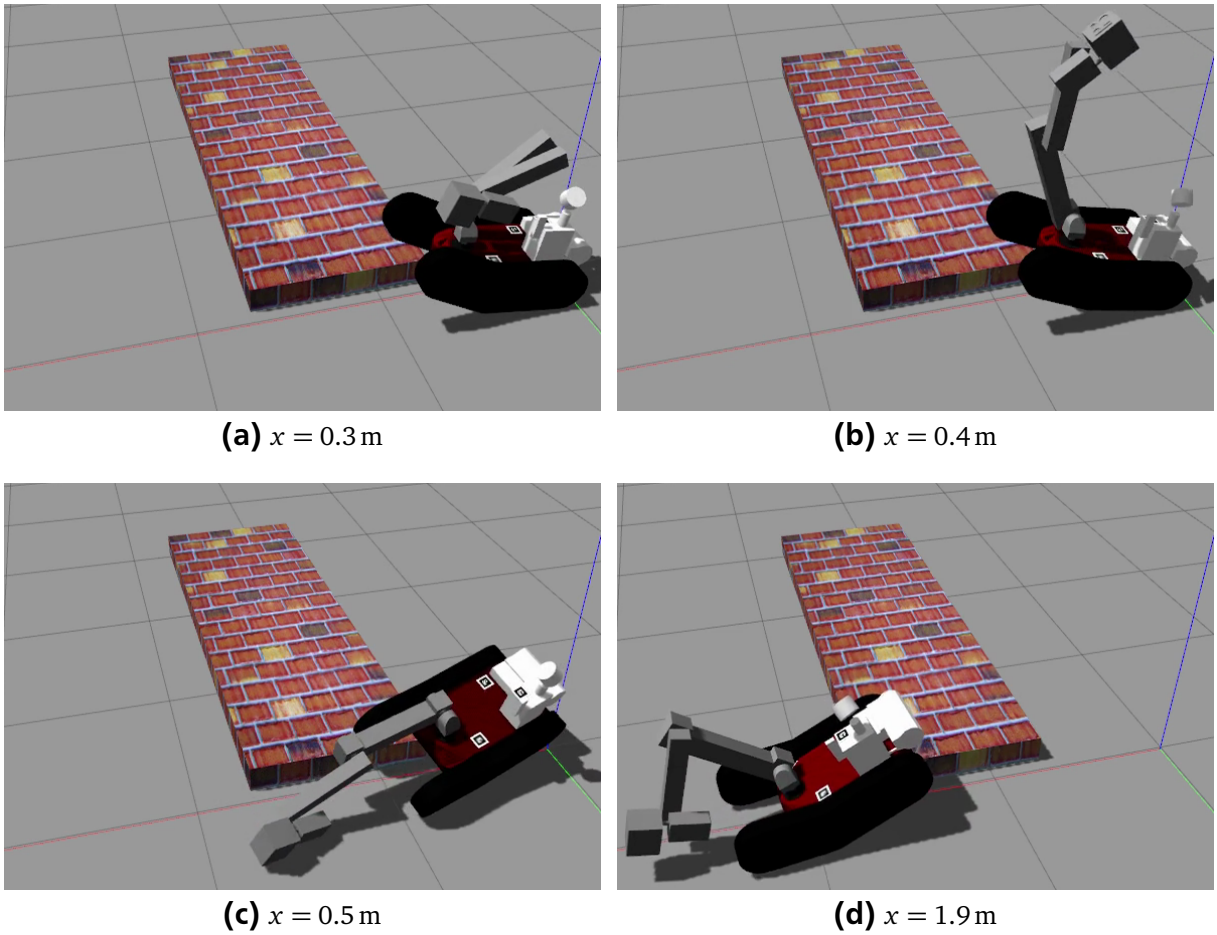
Asymmetric Step: In the final testing scenario, performance on an asymmetrical step is evaluated. The stability graph can be seen in Figure 35. Again both approaches could successfully traverse the obstacle with an average stability of 0.57 for the reference and 1.57 for whole-body planning.

The reference implementation almost tipped over after getting onto the step and tipping to the side at $x = 0.5\,\text{m}$. The whole-body approach had problems at the same position, similar to the cinder block scenario. The configuration before and after tipping can be seen in Figure 34b and 34c. Again the robot tipped with high velocity and the sensor head came dangerously close to the ground. But the vehicle controlled managed to drive back on track and the remaining plan was executed as desired.

The time to cross the obstacle is also compared in Figure 36. The obstacle is completed at $x = 2.2\,\text{m}$. As before, the reference is much faster and only needs a third of the time.

Summary: In summary, the tests have shown that the whole-body planning successfully increases robot stability and prevents vehicle tipover. The approach consistently outperformed the existing reference implementation in terms of measured stability. The accuracy of the prediction was shown by comparing predicted with measured stability. Moreover, the flexibility of the method was demonstrated by successfully crossing different obstacles.

The limits of the approach became apparent when crossing the cinder block and the asymmetrical step. Fast movements violate the quasi-static assumption and can lead to instabilities as only static stability is considered during optimization.



(a) $x = 0.3\,\mathrm{m}$ **(b)** $x = 0.4\,\mathrm{m}$

(c) $x = 0.5\,\mathrm{m}$ **(d)** $x = 1.9\,\mathrm{m}$

Figure 34: Robot crossing the asymmetrical step using the whole-body planner.

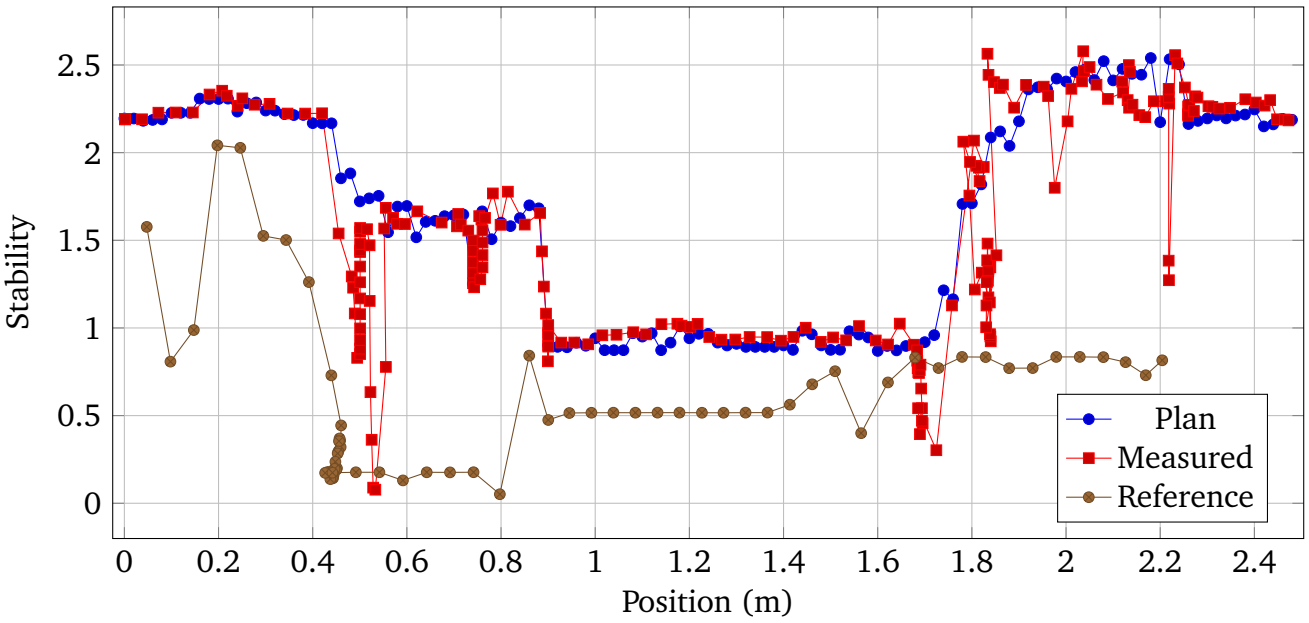


Figure 35: Stability comparison for crossing the asymmetrical step.

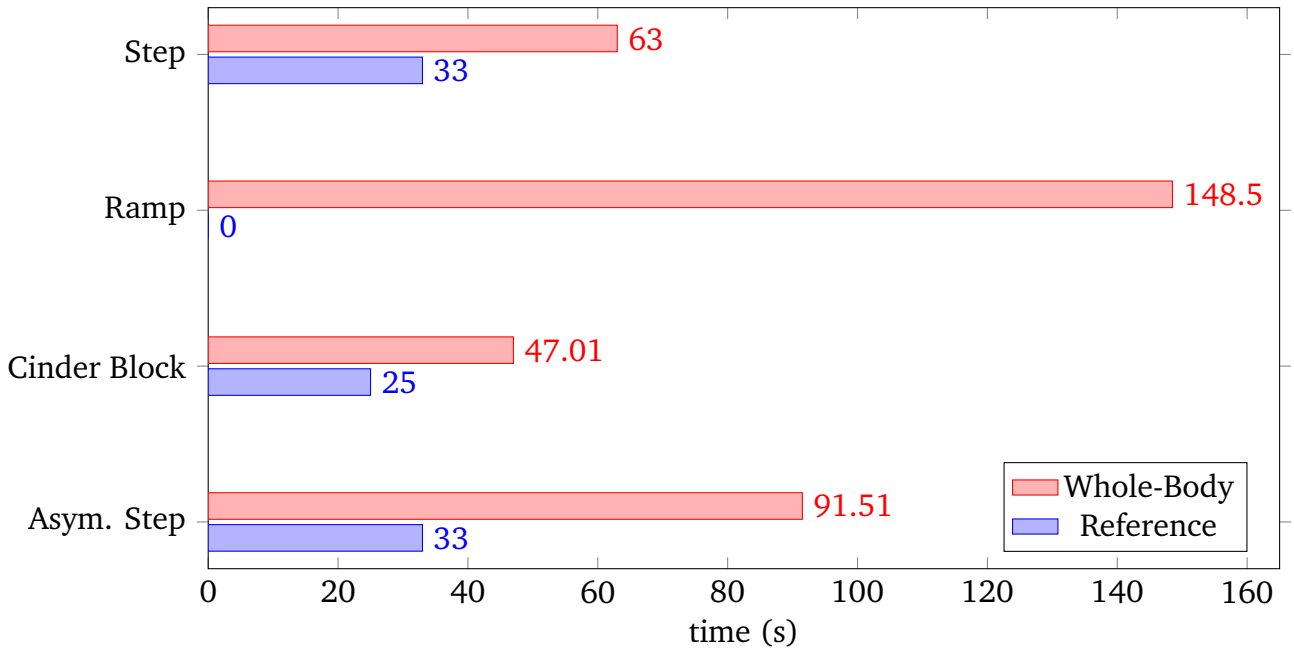


Figure 36: Comparison of the time needed to cross the different obstacles. Overall, the reference implementation is 2-3 times faster than the whole-body planning. As the reference implementation could not finish the ramp obstacle, no time is given.

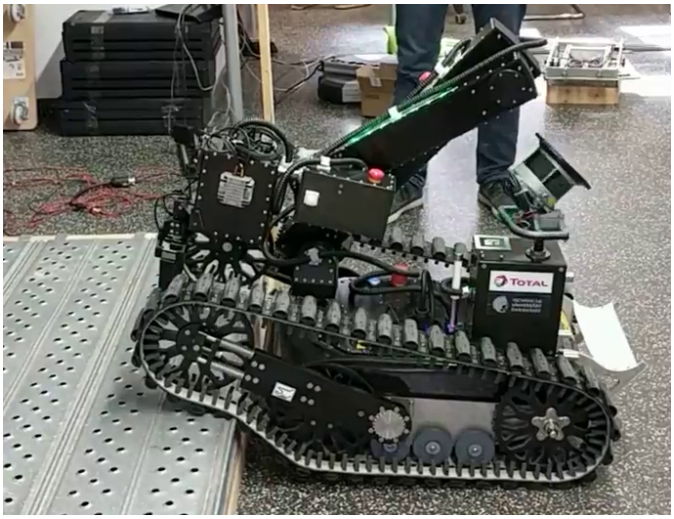


Figure 37: Hector Tracker driving up a step.

6.2.2 Real Robot

The whole-body planning was also evaluated on the real robot platform. Compared to the simulation, the planning has to deal with inaccuracies of the robot model. Therefore, this test also evaluates the robustness of the approach against model inaccuracies.

The test was conducted using a single step with a height of 0.18 m. It can be seen in Figure 37 with the robot on the step.

The robot successfully managed to drive up the step. The measured stability plotted against the original plan can be seen in Figure 38.

For the most part, the measured stability matches the planned stability. The first divergence happened at $x = 2\,\text{m}$, where the measured stability drops to 0.5. At this point, the estimated pose was slightly wrong, so the front left track lost contact with the ground. In reality, the robot did not lose contact, so the measurement is erroneous.

The second big difference appears at $x = 0.9\,\text{m}$. Here, the prediction was wrong because of model inaccuracies and the robot lost contact with the front flippers for a moment. Stability is restored after the robot tips slightly forward onto the flippers.

This test shows that the simulation results transfer to the real robot. While model inaccuracies had an impact on the accuracy of the stability prediction, the robot was still able to climb the step without tipover.

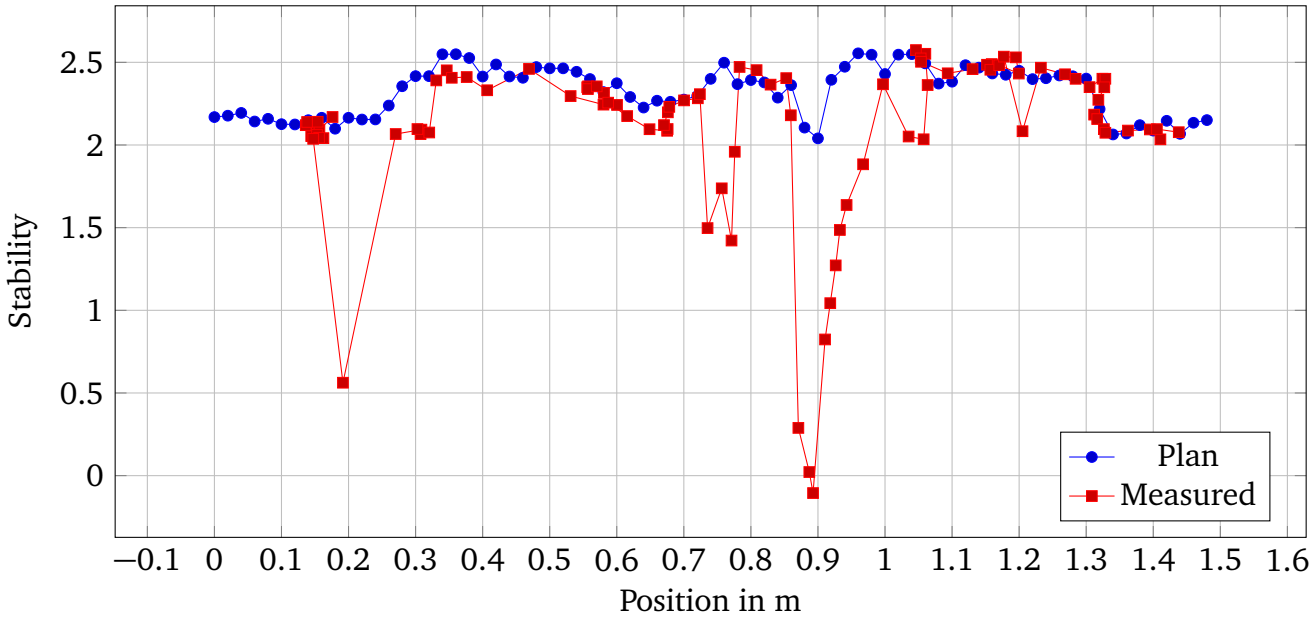


Figure 38: Stability recorded on the real robot when crossing a step.

7 Conclusion and Future Work

7.1 Conclusion

In this work, a new whole-body planning approach was proposed for traversal of obstacles with autonomous ground robots. Using this approach, the robot is able to optimally use its arm and flipper to prevent tipover and maximize stability along a given path. The implementation is robot-independent. The method generalizes to alternative robot platform as long as basic sensor requirements are satisfied.

The effectiveness of the approach was shown in tests in simulation and on the real robot. The robot was able to traverse diverse obstacles successfully without tipover. The average measured stability was consistently higher than an existing approach used for reference.

It was shown that the optimization constraints based on an SDF world model effectively prevented collisions with the environment and the robot itself. Moreover, by selecting diverse test scenarios, the flexibility of the approach was highlighted. During execution the measured stability was very close to the originally planned stability, indicating that the prediction is accurate.

The results in simulation were consistent to evaluations performed with the real robot platform.

The whole-body planning was realized using a new contact estimation based on iterative optimizations. By again utilizing an SDF world model and a simplified representation of the robot underside, a fast and accurate estimate of the robot pose and contact points can be achieved. It is also noteworthy, that the estimation does not rely on an initialization above the ground like physics-based simulations.

7.2 Future Work

While the whole-body planning approach achieves promising results in its current state, there is still room for improvement and future innovation.

One aspect, that was also mentioned in the introduction is a reactive behavior. In the current state, a plan is generated using the available map data and then followed without adaption. In a future improvement, the robot could further increase its stability by incorporating current sensor data during execution. This can be complemented with hardware modifications. Inoue *et al.* [28] integrated touch sensors into the tracks. This gives accurate contact information while driving without the need for complete map data.

Another point is the problem of acquiring the map data. For evaluation, the data was previously recorded, so it was considered given. In a real-world scenario in an unstructured environment, the map has to be continuously updated with new sensor data. To tailor a pre-planning approach to these needs, the planning also has to happen continuously with updates when new data is available.

In the current approach, the desired path is sampled with waypoints in Cartesian space. In practice, there can be a huge difference in joint space between two adjacent waypoints. Since stability is only optimized at these points, the robot could theoretically become unstable when moving from one state to the next. The same problem applies for collision checking. A collision-free state is only guaranteed at waypoints and collisions are theoretically possible in-between. These issues could be addressed by planning and optimizing in joint space. However, the problem is much harder than static trajectory planning because the robot base is also moving during execution.

Another point that can be considered is, that previous solutions influence the next solution. The current approach only considers one state. This can lead to situations, where no feasible solution exists because of previous results. A backtracking or post-processing approach could be used to adapt solutions in the past that lead to better future solutions.

A problem that became apparent during the evaluation is the quasi-static assumption. It does not always hold true because sometimes tipping motions are required to cross an obstacle. Ideally, the algorithm could also ensure dynamic stability.

List of Acronyms

SLAM simultaneous localization and mapping

SDF Signed Distance Field

ESDF Euclidean Signed Distance Field

TSDF Truncated Signed Distance Field

COM center of mass

ZMP zero moment point

ODE Open Dynamics Engine

DOF degrees of freedom

ACM Allowed Collision Matrix

SSD Sum of Squared Differences

MMA method-of-moving-asymptotes

SQP Sequential Quadratic Programming

BFGS Broyden–Fletcher–Goldfarb–Shanno

ROS Robot Operating System

PCL Point Cloud Library

IMU Inertial Measurement Unit

RBDL Rigid Body Dynamics Library

MD mean absolute difference

RMS root mean square

List of Figures

References

[1] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.

[2] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121. IEEE, 1985.

[3] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

[4] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Improved updating of euclidean distance maps and voronoi diagrams. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 281–286. IEEE, 2010.

[5] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

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

[7] Dominic Messuri and Charles Klein. Automatic body regulation for maintaining stability of a legged vehicle during rough-terrain locomotion. *IEEE Journal on Robotics and Automation*, 1(3):132–141, 1985.

[8] Evangelos Papadopoulos and Daniel A Rey. The force-angle measure of tipover stability margin for mobile manipulators. *Vehicle System Dynamics*, 33(1):29–48, 2000.

[9] Christophe Grand, Faïz Benamar, Frédéric Plumet, and Philippe Bidaud. Stability and traction optimization of a reconfigurable wheel-legged robot. *The International Journal of Robotics Research*, 23(10-11):1041–1058, 2004.

[10] Guillaume Besseron, Ch Grand, F Ben Amar, and Ph Bidaud. Decoupled control of the high mobility robot hylos based on a dynamic stability margin. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2435–2440. IEEE, 2008.

[11] Mohammad Norouzi, Jaime Valls Miro, and Gamini Dissanayake. Planning stable and efficient paths for reconfigurable robots on uneven terrain. *Journal of Intelligent & Robotic Systems*, 87(2):291–312, 2017.

[12] Chistoph Beck, Jaime Valls Miró, and Gamini Dissanayake. Trajectory optimisation for increased stability of mobile robots operating in uneven terrains. In *Control and Automation, 2009. ICCA 2009. IEEE International Conference on*, pages 1913–1919. IEEE, 2009.

[13] Mohammad Norouzi, Jaime Valls Miro, and Gamini Dissanayake. Planning high-visibility stable paths for reconfigurable robots on uneven terrain. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2844–2849. IEEE, 2012.

[14] Kevin Daun and Johannes Schubert. 3d-motion planning for autonomous rescue robots. 2014.

[15] Kazunori Ohno, Eijiro Takeuchi, Valerie Chun, Satoshi Tadokoro, Tomotake Yuzawa, Tomoaki Yoshida, and Eiji Koyanagi. Rollover avoidance using a stability margin for a tracked vehicle with sub-tracks. In *Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, pages 1–6. IEEE, 2009.

[16] Shigeo Hirose, Hideyuki Tsukagoshi, and Kan Yoneda. Normalized energy stability margin and its contour of walking vehicles on rough terrain. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 181–186. IEEE, 2001.

[17] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[18] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

[19] Dmytro Pavlichenko and Sven Behnke. Efficient stochastic multicriteria arm trajectory optimization. In *Int. Conference on Intelligent Robots and Systems (IROS)*, 2017.

[20] Steven G. Johnson. The nlopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`.

[21] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[22] M. J. D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pages 51–67. Springer Netherlands, Dordrecht, 1994.

[23] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[24] J.M. Gablonsky and C.T. Kelley. A locally-biased form of the direct algorithm. *Journal of Global Optimization*, 21(1):27–37, Sep 2001.

[25] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM J. on Optimization*, 12(2):555–573, February 2002.

[26] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[27] Martin Pecka, Karel Zimmermann, and Tomáš Svoboda. Fast simulation of vehicles with non-deformable tracks. *arXiv preprint arXiv:1703.04316*, 2017.

[28] Daisuke Inoue, Kazunori Ohno, Shinsuke Nakamura, Satoshi Tadokoro, and Eiji Koyanagi. Whole-body touch sensors for tracked mobile robots using force-sensitive chain guides. In *Safety, Security and Rescue Robotics, 2008. SSRR 2008. IEEE International Workshop on*, pages 71–76. IEEE, 2008.