Autonomous Navigation with Obstacle Avoidance of a Nonholonomic Four Wheel Steering Robotic Platform

Technische Universiteit Delft

Autonomous Navigation with Obstacle Avoidance of a Nonholonomic Four Wheel Steering Robotic Platform

M.F.A. Damen

Autonomous Navigation with Obstacle Avoidance of a Nonholonomic Four Wheel Steering Robotic Platform

MASTER THESIS
For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

M.F.A. Damen

November 11, 2015

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF
BIOMECHANICAL DESIGN

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled
AUTONOMOUS NAVIGATION WITH OBSTACLE AVOIDANCE OF A NONHOLONOMIC FOUR WHEEL STEERING ROBOTIC PLATFORM
by
M.F.A. DAMEN
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE.

Dated: ___________________

Supervisor(s):

____
Prof. dr. F.C.T. van der Helm

____
Dr.-Ing A. Schiele

____
Dr. G.A. Delgado Lopes

____
Msc. S. Kimmer

Preface

This document is part of the graduation project for the master programme BioMechanical Design (BMD) at the Delft University of Technology, The Netherlands. As a requirement, the work done during the project should be presented in an IEEE style article. This is to learn how to present a vast amount of work done in a compact way, which is an important skill for researchers. The article, titled *Autonomous Navigation with Obstacle Avoidance of a Nonholonomic Four Wheel Steering Robotic Platform*, is the first part of the document. The remainder of the document, consisting of several appendices, contains a detailed description of the work done during the project. This is to provide a usable reference for future research and to show all the work done during the graduation project that was not covered in the article. The appendices are self-contained, and reading the paper is not required to understand the work described.

The appendices that follow after the article are structured as follows. A short introduction to the thesis project is given in Appendix I. Appendix II contains a detailed derivation of the configuration kinematic model of the INTERACT robotic platform. Appendix III shows how the model was verified and validated. It also contains results of experiments where the simulated platform was compared to the actual platform. Details about visualization of the simulated system including an accurate CAD model of the platform are given in Appendix IV. In Appendix V the localization algorithm is presented, along with accuracy results and possible improvements that could be applied in the future to increase performance. One of the tools used to increase localization performance is platform calibration. This method is covered extensively in Appendix VI. Appendix VII introduces the *Rapid-Exploration Random Trees* (RRT) algorithm that is used for path planning for nonholonomic systems. In a comprehensive table, the performance of several versions of this algorithm is summarized. The feedback control of the platform is presented in Appendix VIII which is the last part necessary for autonomous navigation. Results of full autonomous navigation experiments are given in Appendix IX. The final part of the document, Appendix X, gives a discussion based on the results of the project and proposes further research and improvements.

I would like to thank André Schiele for providing the opportunity of graduating at the Telerobotics and Haptics laboratory at ESA Estec in Noordwijk, and for his feedback on several reports and this thesis itself. Also, many thanks to Stefan Kimmer for his daily supervision, feedback and help during the project. Professor Frans van der Helm, who was the university supervisor and head of the committee present during the defense of the thesis, is also thanked for his time and contribution to the final result. Finally, a lot of gratitude is directed to the members of the lab that were willing to help out during experiments and were available for discussion to find the best approach. Especially Joao Rebelo and Eloise Matheson gave a lot of useful input and feedback during many occasions.

M.F.A. Damen
Delft, November 2015

Contents

Autonomous Navigation with Obstacle Avoidance of a Nonholonomic Four Wheel Steering Robotic Platform

M.F.A Damen, *TU Delft*

***Abstract*—INTERACT is a space technology demonstration experiment in which an advanced wheeled mobile robot (WMR) will be tele-operated from space. The experiment consists of a navigation phase and a manipulation phase. To cope with adverse communication conditions and limited means for control, an autonomous navigation algorithm needs to be developed for the navigation phase of the experiment. The algorithm requires accurate localization, path planning among obstacles and feedback control of the nonholonomic system. The main goal is to autonomously position the platform in front of a taskboard within 15 cm in position and 6° in heading.**

Such a navigational algorithm has not been developed before for a four wheel steering platform capable of crab motion and rotation in place. Furthermore, it is a challenge to achieve the necessary localization accuracy with the available sensors.

The navigation algorithm is implemented on the actual INTERACT platform and its performance is experimentally validated. The algorithm is limited to 2D planning and control. The path is planned using the Rapid-exploration Random Tree algorithm and tracking control is performed using approximate linearization around the reference trajectory and PID control laws. Feedback is provided by localization based on dead-reckoning and odometry. The experimental results show an average error over 10 trials of 14 cm in x, 13 cm in y and 4.7 degrees in heading, which is within limits.

***Index Terms*—Navigation, odometry, dead-reckoning, path planning, RRT, feedback control, approximate linearization**

I. Introduction

THE INTERACT project is a space technology demonstration experiment by the European Space Agency (ESA). A four wheel steering, four wheel drive robotic platform located on Earth is operated from the International Space Station (ISS) by an astronaut. The goal of the experiment is to drive the platform through partially unknown terrain towards a taskboard where several manipulation tasks such as pressing a button and peg-in-hole can be performed. The astronaut can operate the robot via a one Degree of Freedom (DoF) joystick capable of providing haptic feedback and a custom GUI running on a Dell tablet PC. The purpose of the project is to prove the concept of real time control including haptic feedback of the manipulator arms under micro gravity conditions while handling time delays of up to 1 second and periods with loss of signal.

The experiment is a first step towards astronaut control of a robot in an extraterrestrial environment (e.g. the surface of Mars) from an orbiter. The use of robots on the surface is necessary because sending astronauts and especially returning them is both costly and dangerous.

Previous experiments with this platform have taken place [1] during which it was manually controlled by the astronaut, but results have not been published yet. However, during the experiments it was noticeable that navigation of the robot under adverse communication conditions and with limited means for astronaut control, is very tedious and time consuming work. Furthermore, to reduce the load on the operator during these experiments, the drive modes of the platform were limited to rotations in place and straight driving. The four wheel steering and crab mode (parallel steering of all four wheels) were not available to the astronaut, and hence the platform could not be used to its full capability.

To solve this problem, an autonomous navigation algorithm that takes over the control from the astronaut has been developed and is presented in this paper. The algorithm is able to position the platform in front of the taskboard while avoiding obstacles in the environment. The operator has to provide a rough location estimate of the taskboard and the position of obstacles. During navigation, the operator only has a supervisory role and can intervene at any time by terminating execution and resetting the start and goal positions.

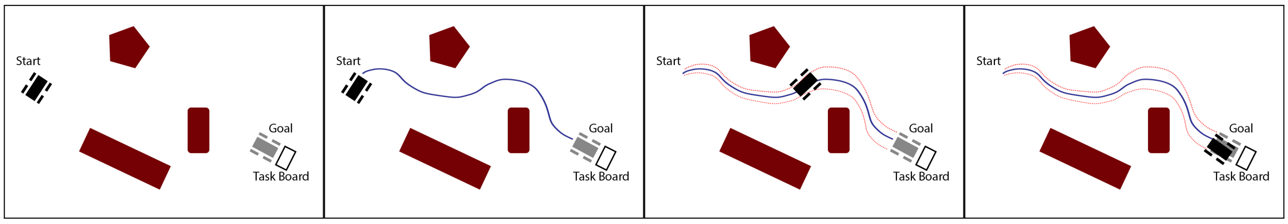
To realize autonomous navigation, the INTERACT platform needs to be able to: (1) localize itself within the environment, which requires an estimate of the system pose, (2) plan a path while avoiding obstacles, and (3) accurately track this path using feedback control. These three topics will constitute the main part of the navigation algorithm and will be treated thoroughly in this paper. A necessary tool and basis for designing the algorithm is a mathematical model of the four wheel steering, four wheel drive platform, which captures all possible drive modes. Hence, a kinematic model of the platform will be discussed as well.

A. Problem Statement

Figure 1 shows the navigational problem to be solved by the algorithm. The platform is initially positioned at an arbitrary location in the environment and needs to navigate to the taskboard. It is assumed that the platform cannot identify all obstacles autonomously, and it is left for the astronaut to exercise his human judgement on which areas of the environment are save to traverse. For this purpose, the astronaut receives a bird's-eye view similar to Figure 1a to indicate the obstacles and goal position. After this is done, a path planning algorithm is required to find a feasible path to the goal.

The planning algorithm should take the nonholonomic constraints of the system into account and result in a feasible path. Furthermore, obstacles should be avoided such as shown in Figure 1b. There exist many planners capable of doing this. These planners can roughly be divided into two groups. First, *combinatorial based* planners, which are complete (i.e. if a solution exist it will be found, if not, this will be reported) but generally require long computation

M. Damen is a Master student at the Department of Mechanical Engineering, Delft University of Technology, Delft, The Netherlands, performing his graduation work at ESA, ESTEC in Noordwijk, The Netherlands. E-mail: matthijs.damen@esa.int



(a) Define obstacles, goal position (b) Path Planning (c) Path Tracking (d) Taskboard Reached

Fig. 1. Sequence showing the several steps of the navigational problem. 1a shows the starting conditions with the obstacles (red) and goal position (grey). 1b shows the planning of a path (blue) towards the goal. 1c shows execution of the paths with accumulating tracking/localization errors (dashed red). 1d shows the platform arriving at the taskboard.

times. Examples are the potential field method [8] and the skeleton based method [9]. Second, the *sampling based* planners are not complete but are generally faster. They are based on random sampling of the configuration space which results in a probabilistic road map (PRM). These planners often use graph search algorithms to find a solution path within the generated road map. Examples are Obstacle Based PRM [2], lazy PRM [3], Medial Axis PRM [4] and Rapid-exploration Random Tree (RRT) algorithms [5], [6]. RRT algorithms are very suited for nonholonomic problems, but an implementation that takes all drive modes of the INTERACT platform into account has not been found in literature.

For the execution of the path, both localization as well as feedback control are necessary. Localization is challenging with this platform as no sensors are available that give an absolute position measurement. Hence, the estimate of the system pose must fully rely on the proprioceptive sensors such as wheel encoders and the inertial measurement unit (IMU). Because these measurements are all relative, they suffer from unbound error accumulation, as is indicated in Figure 1c with a dashed red line. The accuracy can be increased by performing platform calibration using e.g. the UMBmark technique [17], [18], [19], [20]. However, no reports that show application of this method to a four wheel steering vehicle have been found. Other techniques to increase accuracy are sensor fusion [15], [16], and wheel slip detection [21], [22].

Feedback control of nonholonomic systems is complicated because the platforms are generally under-actuated. This was first described in an article by Brockett [23] where he shows that one needs discontinuous and/or time-varying feedback to stabilize nonholonomic systems. Several controller designs can be found in literature that apply this type of feedback [24], [28], [29], [30]. However, the control laws are applied to simulated car-like systems and unicycles, but not to four wheel steering platforms. Furthermore, no experimental validation of the controller is reported.

To the authors knowledge, no single report exists that addresses the end-to-end navigation task of a four wheel drive, four wheel steering vehicle such as INTERACT.

B. Goal

The goal of this work is to develop, implement and experimentally validate an autonomous navigation algorithm for the four wheel steering, four wheel drive INTERACT platform.

The complete algorithm should be accurate enough such that the resulting final position is suitable for the manipulation tasks, i.e. the taskboard is reachable for the robotic arms and no relocation is required to do the tasks. While performing the experiments described in [1], acceptable limits on the final position were established: it should be reached with an accuracy of 15 cm in position (both directions), and 6° in heading. The algorithm should be capable of achieving this accuracy for paths with a length in the order of 20 meters. Furthermore, a bound on the localization error of 0.5% of the travelled distance and a tracking error that does not exceed the minimum turning radius (1 meter) is required to keep the platform close enough to the path such that collisions are prevented.

To achieve a feasible path, the system needs to be modelled including the limitations imposed by the nonholonomic constraints. This mathematical model is used in the path planning, and to develop a controller capable of accurate tracking of a reference trajectory. Both planning and control are performed in 2D, which is deemed sufficiently descriptive while it simplifies the problem greatly.

To summarise the requirements for the navigation algorithm:

- The path planning should be able to incorporate all platform drive modes.
- The path planning should generate a feasible path while avoiding obstacles in the environment, keeping into account the increasing error in the localization estimate.
- Tracking accuracy should be within the minimum turning radius, which is 1 meter for this platform.
- The upper bound on the localization error is 0.5% of the travelled distance.
- The final position should be reached with an accuracy of 15 cm in position, and 6° in heading.
- The path length can be up to 20 meter.

II. Approach

The structure of the proposed navigation algorithm is given in Figure 2. After the operator inputs the location of the obstacles and taskboard position in the environment, the first stage of navigation is entered: the *coarse approach*. The path planning algorithm should plan a feasible path from the initial platform location to the point indicated by the operator, while the obstacles in the environment are

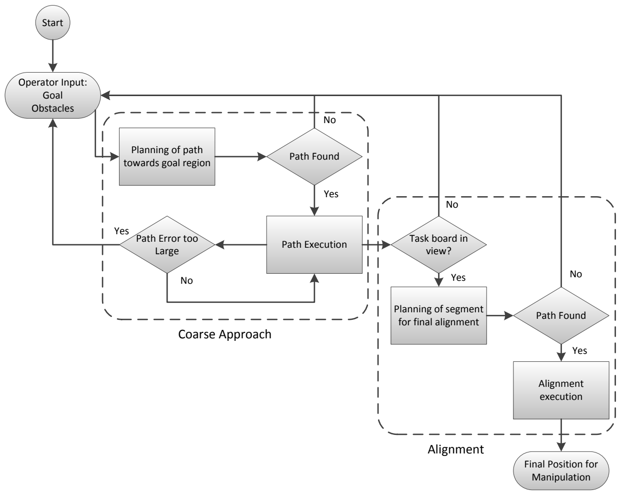


Fig. 2. Flow-chart of the proposed autonomous navigation algorithm. Two stages of the algorithm can be identified. The first stage is the *Coarse Approach*. The second stage is the *Alignment* stage where the taskboard is maneuvered in front of the taskboard.

avoided. Preferably, the platform should be used to its full capability including crab mode, rotate in place, and four wheel steering. It is not a requirement that the path be optimized for a certain cost function based on e.g. length. This results in the Rapid Exploration Random tree (RRT) algorithm as a choice for the planning. If a path cannot be found, additional operator input is required. Else, the first path execution state is entered.

During path execution, the platform should track the path by using the position estimate from localization as feedback for the controller. As stated before, the platform can only use the on-board proprioceptive sensors (wheel and steering encoders, IMU) to estimate the location. By using platform calibration and simple wheel slip detection, an accuracy of 0.5% of the total travelled distance should be achieved.

The feedback controller developed for four wheel steering is based on approximate linearization around the reference trajectory as described in [28], which is extended to a system with four wheel steering. This control method requires the system to be transformed to the chained form. This canonical structure is often used for non-linear systems to generalize the controller derivation and simplify the establishment of feedback laws (see e.g. [38], [39], [25]). By putting the equations of the system in the chained form, an underlying linear structure becomes apparent, although the system itself is non-linear. The controller uses the feedforward command generated by the planning algorithm and computes a correction using the localization estimate. It is able to stabilize the system on the reference trajectory in four wheel steering mode. For turn in place and crab mode, a separate PID controller is developed. If, at any point in the trajectory execution, the tracking error becomes too large (i.e. larger than the minimum turning radius), the algorithm is reset. This means that the operator needs to provide the obstacle and goal positions again and planning is repeated with the new data. The operator is able to intervene and terminate the execution at any point in time if he deems this necessary for safe and successful operation.

When the platform reaches the end of the path of the first planning stage, it should be close enough to the taskboard such that it can be recognized using vision software (namely Halcon-12) and the 3D position and attitude can be estimated. The implementation of the object recognition itself is beyond the scope of this paper, and hence assumed to be available. The distance at which the taskboard should be recognized is in the order of 5 meter, which is left as a requirement for the vision system. If the taskboard is in view, the second stage called *alignment* is entered and a final position in front of the taskboard can be deduced. Furthermore, the odometry error accumulated during the execution of the first path segment is reset to zero since an accurate position measurement w.r.t. the taskboard is now available. Using the same path planning algorithm, a path is planned that brings the robot in front of the taskboard. Through the use of visual servoing, the taskboard can be kept within view of the platform while driving. Hence, the accurate position measurement w.r.t. the taskboard is also available during execution, and the localization is not solely dependent on the proprioceptive sensors, which greatly increases accuracy.

III. Kinematic Model of the System

A. *Implementation*

A mathematical model of the wheeled mobile robot (WMR) is developed. The model is used to simulate the system in Matlab/Simulink and thereby make the development of the algorithm faster as different versions and parameter tuning can be tested in a simulation. Furthermore, the path planning algorithm requires an accurate model of the system to generate a feasible path. Finally, some of the control methods specific to nonholonomic systems require a (kinematic) model of the system in chained form to derive appropriate control laws.

A kinematic model is chosen as the system representation. This has several reasons. The model should preferably take the same inputs as the actual platform (which are u_1, velocity in [m/s], and u_2, steering angle in [°]). A dynamic model generally has forces and torques as input, which is not necessary in this case. Lower level control such as for the torque of the drive motors is taken care of by the on board platform controllers designed by the manufacturer. Because the velocities used during the experiment are relatively small (well below 1 m/s), many dynamic aspects such as the suspension, steering actuators and acceleration of the platform can be neglected without introducing significant inaccuracy. Hence, it is deemed sufficient to develop a kinematic model of the platform.

The kinematic model of the four wheel steering, four wheel drive INTERACT platform is developed based on the systematic approach given in [24] and uses the simplification towards a double steering bicycle model presented in [34]. With this approach, the front and rear wheel pairs are merged into a single virtual wheel located in the middle of the wheel pair axis as is shown by the dashed lines in Figure 3. Figure 3 also shows important platform parameters such as the wheel base w and the distance L between the front and rear virtual wheels. A reference point P is chosen which

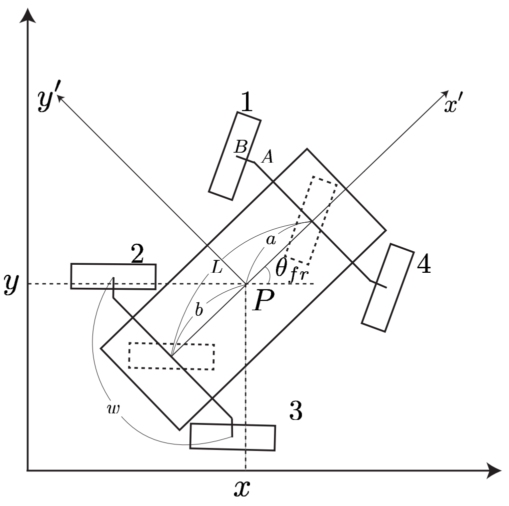


Fig. 3. Sketch of the parameters that define the INTERACT system and the simplification towards a bicycle model. The image shows the world frame with the coordinates (x,y) of the reference point P on the chassis. P indicates the origin of the robot centred coordinate frame. The angle θ_{fr} is the angle between the robot frame and the world frame. Both the two front wheels as well as the two rear wheels are collapsed into a virtual wheel centred on the respective axis (dashed wheels). The distance from P to the front and rear wheel are a and b respectively. The distance between the wheel axes is $L = a + b$. The wheel base is given by w.

serves as the origin for the robot centred reference frame that indicates the platform position x, y and heading θ_{fr}. The distance from P to the front and rear wheel is given by a and b respectively. The state vector $\boldsymbol{q}$ is given by the pose of the platform

$$\boldsymbol{q} = \begin{bmatrix} x & y & \theta_{fr} \end{bmatrix}^T. \tag{1}$$

The two inputs to the system are the velocity input u_1 and the steering angle u_2.

Several assumptions are made to simplify the modelling, which are:

- The robot frame is a rigid body
- Wheels are non-deformable vertical discs able to roll around their horizontal axes which are always parallel to the ground.
- There is a single point of contact between the ground and the wheel.
- The wheels obey the no side-slip and rolling without slipping constraints at all times.

The no side-slip constraint is a nonholonomic constraint [28], [35], which means it is non integrable and implies limitations on the platform generalized velocities $\dot{\boldsymbol{q}}$. It limits the velocity of the wheel to be only in the plane of the wheel, with no component perpendicular to the wheel disc. Mathematically, this no side-slip can be expressed as

$$\dot{x}\sin(\theta_w) - \dot{y}\cos(\theta_w) \quad = \quad 0, \tag{2}$$

which is the basis for the kinematic model. Here, x and y are the position coordinates of the wheel, r_w is the wheel radius, and θ_w is the steering angle of the wheel measured counter clockwise from the x'-axis, which is the second control input u_2.

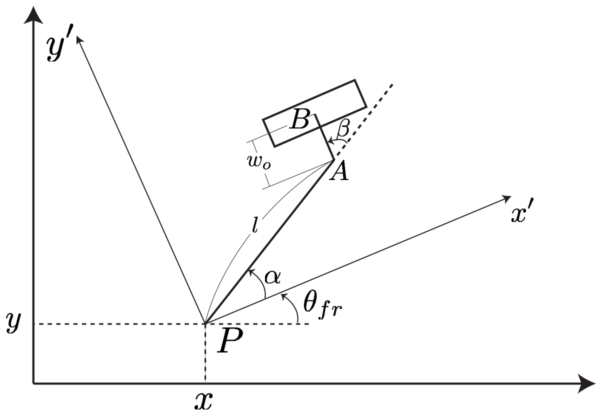


Fig. 4. Sketch of the wheel geometry and parameters. For each wheel, l is the distance from P to A, which is the rotation point of the steering axis. β is the angle between the line PA and the wheel's horizontal axis of rotation. β is the orientation angle of the wheel which is variable if the wheel is orientable. α is the angle between x' and the line PA. The distance between A and the ground contact point B is designated w_o.

The wheel geometry of the INTERACT platform includes an offset w_o between the vertical steering axis A and the wheel-ground point of contact B as shown in Figure 4. Incorporating this geometry into the no side-slip constraint results in

$$\begin{aligned} \dot{x}\cos(\alpha+\beta+\theta_{fr}) \quad &+ \quad \dot{y}\sin(\alpha+\beta+\theta_{fr}) \\ &+ \quad l\dot{\theta}_{fr}\sin(\beta) = 0, \end{aligned} \tag{3}$$

which can be rewritten into

$$\begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & l\sin(\beta) \end{bmatrix} \boldsymbol{R}(\theta_{fr})\dot{\boldsymbol{q}} = 0. \tag{4}$$

In these equations, α indicates the angle between the x'-axis and the line PA as shown in Figure 4. β is the angle between the wheel its horizontal axis of rotation and the extension of the line PA. $\boldsymbol{R}(\theta_{fr})$ is the rotation matrix from the robot frame to the world frame:

$$\boldsymbol{R}(\theta_{fr}) = \begin{bmatrix} \cos(\theta_{fr}) & \sin(\theta_{fr}) & 0 \\ -\sin(\theta_{fr}) & \cos(\theta_{fr}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

The relationship between β and the steering angle θ_w is given by: $\theta_w = \alpha + \beta - \pi/2$. The subscripts 1 to 4 indicate the wheel parameters and steering angles for the four real wheels numbered as shown in Figure 3. The two virtual wheels are indicated with a subscript f for front and r for rear.

The no side-slip constraints for the two virtual wheels can be put into matrix form

$$\boldsymbol{C}(\beta)\boldsymbol{R}(\theta_{fr})\dot{\boldsymbol{q}} \quad = \quad 0, \tag{6}$$

with

$$\boldsymbol{C} \quad = \quad \begin{bmatrix} c(\alpha_f+\beta_f) & s(\alpha_f+\beta_f) & l_f s(\beta_f) \\ c(\alpha_r+\beta_r) & s(\alpha_r+\beta_r) & l_r s(\beta_r) \end{bmatrix} \tag{7}$$

Here, sin and cos are abbreviated to s and c respectively for compactness.

The allowed system velocities can now be found by computing the basis of the null space of the matrix $\boldsymbol{C}$

$$\boldsymbol{\Sigma} = Null\,[\boldsymbol{C}] \tag{8}$$

and the generalized velocities $\dot{\boldsymbol{q}}$ can be computed as

$$\begin{aligned}\dot{\boldsymbol{q}} &= \boldsymbol{R}(\theta_{fr})\boldsymbol{\Sigma}(\beta)u_1 \\ &= \begin{bmatrix} -[\frac{1}{2}s(\theta_{fr})(c(\beta_r)s(\beta_f)+c(\beta_f)s(\beta_r))+s(\beta_f)s(\beta_r)c(\theta_{fr})] \\ [\frac{1}{2}c(\theta_{fr})(c(\beta_r)s(\beta_f)+c(\beta_f)s(\beta_r))-s(\beta_f)s(\beta_r)c(\theta_{fr})] \\ -\frac{\sin(\beta_f-\beta_r)}{L} \end{bmatrix} u_1\end{aligned} \tag{9}$$

Where u_1 represents the platform velocity input in [m/s]. In the case of four wheel steering, the front steering angle is always equal but opposite to the rear steering angle: $\theta_f = -\theta_r = \theta_w$. Substituting this in (9) results in the simplified system for four wheel steering

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \cos(\theta_{fr})\cos(\theta_w)^2 \\ \sin(\theta_{fr})\cos(\theta_w)^2 \\ \frac{\sin(2\theta_w)}{L} \end{bmatrix} u_1 \tag{10}$$

The kinematic model (9) is able to represent the four wheel steering and crab mode of the platform. However, it contains a singularity for $\theta_f = \pi/2$ and $\theta_r = -\pi/2$ or, equivalently, $\beta_f = \beta_r = 0$. The INTERACT platform is capable of rotation in place, which requires the steering angles to be precisely this. Hence, a separate kinematic model needs to be derived for this special maneuver.

The model for rotation in place only has one control input: the velocity u_1, which directly determines the angular velocity. The x and y coordinates of the point P are stationary or describe a circular motion if P is offset from the middle. The generalized velocities can be computed as follows

$$\dot{\theta}_{fr} = w_o + \sqrt{\frac{L^2}{4} + \frac{w_A^2}{4}}\boldsymbol{\eta} \tag{11}$$

$$\dot{x} = -\left(\frac{L}{2} - a\right)\sin(\theta_{fr})\dot{\theta}_{fr} \tag{12}$$

$$\dot{y} = \left(\frac{L}{2} - a\right)\cos(\theta_{fr})\dot{\theta}_{fr} \tag{13}$$

B. Method

Both the model for four wheel steering and crab mode, as well as the model for rotate in place are validated by comparing simulated output with the real platform motion. This is done by running the simulation and the platform side by side while giving the same input. The output of the models is compared to the actual platform pose $\boldsymbol{q}$ which is measured by a Vicon motion capture system. 6 Vicon cameras are set-up outdoors in a square of roughly 5 by 5 meter. Several motions of the platform are performed such as driving straight, 4 wheel steering with angles ranging from $-25°$ to $25°$, and rotate in place. The correspondence between platform and simulation is quantified by comparing the final position of both and express this as a percentage of the travelled distance d_{total}

$$e_d = \frac{\sqrt{(x_m - x_v)^2 + (y_m - y_v)^2}}{d_{\text{total}}} \cdot 100\% \tag{14}$$

where the subscript m and v indicate the final position according to the model and vicon system respectively.

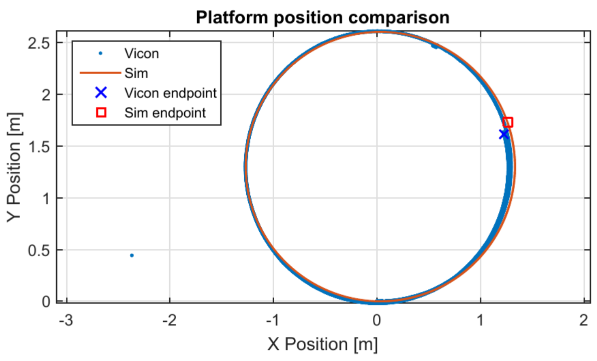


Fig. 5. Example run of a model validation experiment for a steering angle of -20 degrees. The blue line shows the actual platform position measured with the Vicon system. The orange line shows the platform location from the mathematical model.

TABLE I
RESULTING ERRORS BETWEEN THE FINAL POSITION OF THE PLATFORM ACCORDING TO THE MODEL AND ACCORDING TO THE GROUND TRUTH.

Experiment	e_d
Straight driving	1.38%
Turn-in-place	6.44%
10° turn	4.35%
15° turn	2.34%
20° turn	1.29%
$-10°$ turn	0.60%
$-20°$ turn	1.14%
$-25°$ turn	0.86%

C. Results

The results of the model validation experiments are summarized in Table I. The model shows good correspondence with reality: the average final position error is about 2% of the travelled distance. The position comparison of an example run of a validation experiment for a 20° turn is shown in Figure 5.

IV. Localization

A. Implementation

The following sensors are available on the platform to estimate its location:

- Steering angle encoders
- Wheel rotation encoders
- Three axis gyroscopes (IMU)
- Three axis accelerometers (IMU)

The basis of the localization algorithm is the incremental distance travelled which is computed from the wheel rotation encoders. This method of localization is called *dead-reckoning*. The position update equations for a four wheel steering platform are taken from [36]. First, the cumulative encoder count N for the wheel rotation can be used to compute the incremental count ΔN

$$\Delta N = N_i - N_{i-1}. \tag{15}$$

From the incremental count, the incremental distance Δd can be computed using a conversion factor C based on

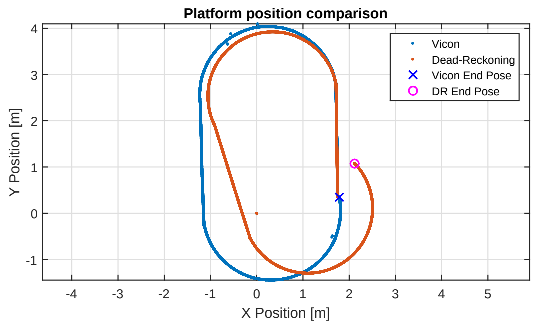


Fig. 6. Example run of the calibration experiment. The blue line shows the actual platform position measured with the Vicon system. The red line shows the platform location computed from localization (before calibration). The final error which is used to compute the correction factors is the difference between the blue cross and the pink circle.

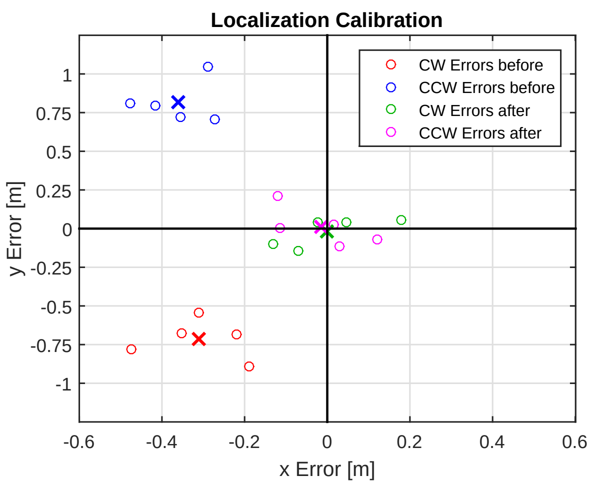


Fig. 7. Final position errors of the localization calibration experiment in CW and CCW direction. The red circles show the CW results before calibration, with the red cross being the mean of the 5 runs. The blue circles represent the CCW results before calibration. The green circles and magenta circles show the same error after calibration of the platform parameters.

the wheel diameter and the number of encoder pulses per revolution N_{rev}

$$\Delta d = C\Delta N, \quad (16)$$
$$C = \frac{2\pi r_w}{N_{rev}}. \quad (17)$$

The update on the robot state can be computed by

$$\Delta\theta_{fr} = \frac{\Delta d_L - \Delta d_R}{w}, \quad (18)$$
$$\Delta x = \Delta d \cos(\theta_{fr_{i-1}} + \Delta\theta_{fr}/2), \quad (19)$$
$$\Delta y = \Delta d \sin(\theta_{fr_{i-1}} + \Delta\theta_{fr}/2). \quad (20)$$

The position update equations (18)-(20) rely on the knowledge of the platform parameters r_w and w. If the platform parameters are not accurately known, the position estimate will have a systematic error. This error can be compensated for by performing platform calibration. A widely used method is the UMBmark calibration technique [37].

For the computation of $\Delta d_{L/R}$ in (18), only one encoder on each side is required. As there are two wheels on each side, there is a redundancy in sensors. This can be used to incorporate wheel slip detection. If the encoder counts of the two wheels on one side differ by more than a couple counts, it can be assumed that the wheel with the higher count is slipping. Hence, the encoder measuring the higher count is ignored, and the lower count is taken to compute (18).

The localization estimate is further improved by including IMU sensor data. An attempt has been made to fuse the IMU accelerometer data with the dead-reckoning estimate using a Kalman filter. However, it was found that the accelerometers suffer from bias and drift, which makes the estimate less accurate than with dead-reckoning alone. Improved accuracy can be achieved, however, by replacing the heading update (18) with measurement and integration of the gyroscope signal from the IMU

$$\Delta\theta = \left(\dot{\theta}_{\text{gyro}} - \dot{\theta}_{\text{bias}}\right) T_s \quad (21)$$

where T_s is the sample time of the localization update and $\dot{\theta}_{\text{bias}}$ is the bias correction for the gyroscope. This correction factor is computed by measuring the sensor output for an extended time while the platform is stationary and averaging the drift over time.

B. Method

The UMBmark method is used to calibrate the platform parameters used in (18)-(20). The method relies on driving the blue pattern shown in Figure 6 five times in both clockwise (CW) and counter clockwise (CCW) direction. This is done in a workspace of roughly 5 by 5 meter. The localization estimate is compared to the ground truth measured with a Vicon motion capture system. The error in the final position of the localization estimate is attributed to unequal wheel diameters and uncertainty in the wheel base, from which a correction factor for the wheel radius and wheel base results.

After platform calibration is done, the performance of the localization is tested by driving longer distances (~100 meter) incorporating all drive modes. The actual platform position is compared to the localization estimate, and the performance is expressed using the metric e_d (14) but using the localization estimate instead of the model result. The driving is done in a workspace of 10 by 5 meters in an indoor location with high traction ground. The platform is operated manually at a speed of about 0.3 m/s.

C. Results

Figure 7 shows the error metric e_d of the localization estimate during the UMBmark experiment before and after calibration has taken place. The blue and red markers indicate the CCW and CW results respectively before calibration. The magenta and green markers show the error after calibration. A significant improvement of the final error can be seen.

Figure 8 shows the development of the localization error e_d over time for the long distance experiments. The acceptable localization error bound of 0.5% of the travelled distance is indicated by the dashed red line. As can be seen

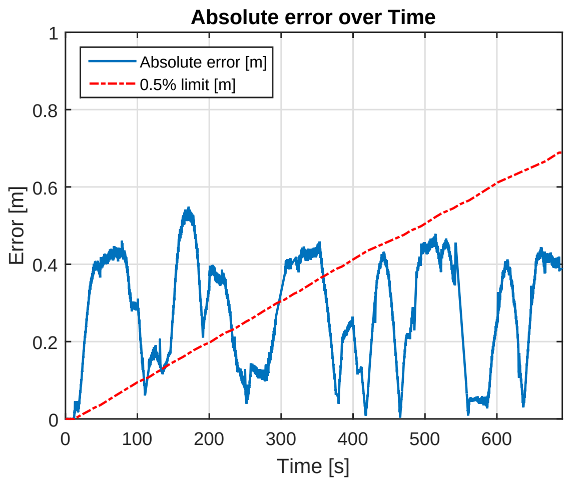


Fig. 8. Development of the localization error over time. The blue line shows the absolute error in meter. The dashed red line shows the acceptable limit of 0.5% of the travelled distance at that time instant.

Algorithm 1 General RRT Algorithm

```
1: $\mathcal{T}$.init($\boldsymbol{q}_{init}$)
2: while !goalReached do
3:     $\boldsymbol{q}_{rand} \leftarrow$ generateRandomState(bounds);
4:     $\boldsymbol{q}_{near} \leftarrow$ nearestNeighbour($\boldsymbol{q}_{rand}$,$\mathcal{T}$)
5:     $[S, \boldsymbol{q}_{new}] \leftarrow$ extendTree($\boldsymbol{q}_{rand}$,$\boldsymbol{q}_{near}$,$\mathcal{T}$)
6: end while
```

Algorithm 2 RRT Nearest Neighbour

```
1: function NEARESTNEIGHBOUR($\boldsymbol{q}_1$,$\mathcal{T}$)
2:     $d = \infty$
3:     for all $q \in \mathcal{T}$ do
4:         if $\rho(q, \boldsymbol{q}_1) < d$ then
5:             $\boldsymbol{q}_{near} = q$
6:             $d = \rho(q, \boldsymbol{q}_1)$
7:         end if
8:     end for
9:     return $\boldsymbol{q}_{near}$
10: end function
```

Algorithm 3 RRT Extend

```
1: function EXTENDTREE($\boldsymbol{q}_{target}$,$\boldsymbol{q}_{near}$,$\mathcal{T}$)
2:     $\boldsymbol{q}_{new} \leftarrow$ generateState($\boldsymbol{q}_{near}$,$\boldsymbol{q}_{target}$,$\mathcal{T}$)
3:     if checkCollision($\boldsymbol{q}_{new}$) then
4:         $\mathcal{T}$.addVertex($\boldsymbol{q}_{new}$)
5:         $\mathcal{T}$.addEdge($u_{new}$)
6:         if $\rho(\boldsymbol{q}_{new}, \boldsymbol{q}_{target}) < \epsilon$ then
7:             $S = Reached$
8:         else
9:             $S = Advanced$
10:        end if
11:    end if
12:    $S = Trapped$
13:    return $[S, \boldsymbol{q}_{new}]$
14: end function
```

from Figure 8, after exceeding the bounds in the first half of the path, the localization error seems to stabilize around 0.3 meter on average and stays well within this 0.5% bound.

V. Path Planning

A. Implementation

The basis of RRT is given in Algorithm 1. A tree $\mathcal{T}$ is constructed in the configuration space of the robot, in which every vertex represents a certain pose or state of the system. The root of the tree is located at the initial position of the platform $\boldsymbol{q}_{init}$. New vertices are generated by selecting a random state within the configuration space $\boldsymbol{q}_{rand}$. Then, by using Algorithm 2, the nearest neighbour $\boldsymbol{q}_{near}$ is found, which is a vertex already in the tree with the lowest distance ρ to $\boldsymbol{q}_{rand}$ according to the distance metric

$$\begin{aligned}\rho(\boldsymbol{q}_1, \boldsymbol{q}_2) \quad &= \quad k_{pos}\sqrt{(x_{\boldsymbol{q}_1} - x_{\boldsymbol{q}_2})^2 + (y_{\boldsymbol{q}_1} - y_{\boldsymbol{q}_2})^2} \\ &\quad + k_\theta \left[1 - \cos(\theta_{fr,\boldsymbol{q}_1} - \theta_{fr,\boldsymbol{q}_2})^2\right] \qquad (22)\end{aligned}$$

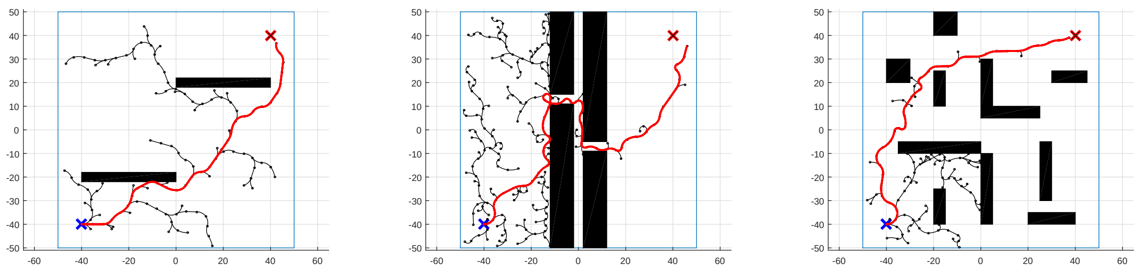
Here, x and y are the position coordinates of the state $\boldsymbol{q}$, and θ_{fr} is the heading. k_{pos} and k_θ are two gains that can be tuned to put a certain weighting between the position and heading.

When $\boldsymbol{q}_{near}$ is found, the RRT Extend function, detailed in Algorithm 3, is called. This function generates a new state $\boldsymbol{q}_{new}$ and checks if this new state causes any collision with the obstacles present in the environment. Because the position of the obstacles is known, both the new state itself and the path from $\boldsymbol{q}_{near}$ to $\boldsymbol{q}_{new}$ can easily be checked for collisions. If no collision is detected, $\boldsymbol{q}_{new}$ is added to the tree, and a check is done whether the goal vertex is reached. Because of the nonholonomic nature of the system, it is virtually impossible to exactly reach the goal position. Hence, if $\boldsymbol{q}_{new}$ is within a certain threshold ϵ of the goal vertex, goal *reached* is returned. If the distance is larger than ϵ, *advanced* is returned. If no new state can be found that does not cause a collision, *trapped* is returned.

Several $\boldsymbol{q}_{new}$ are generated every time RRT Extend is called. These $\boldsymbol{q}_{new}$ are found by integrating the equations of motion for a certain time interval Δt, while the input to the platform u_1, u_2 is constant during this time interval. In this way, a discrete set of ”motion primitives” is available to the path planning, equal to the number of input sets it is allowed to choose from. The new vertex $\boldsymbol{q}_{new}$ is chosen as the motion primitive which minimizes the distance ρ to the goal vertex. The number of motion primitives that is available is a tuning parameter. More motion primitives will make the planning more versatile, but will also increase planning time. By integrating the equations of motion of the system, the nonholonomic constraints are automatically incorporated in the planning, and the resulting path is always feasible. The RRT algorithm is iterated until goal *reached* is returned.

To increase the convergence speed of the algorithm, a technique called *goal bias* can be applied. With a certain chance, the random state $\boldsymbol{q}_{rand}$ is replaced by the goal vertex $\boldsymbol{q}_{goal}$. The amount of bias is a tuning parameter. Alternatively, instead of taking the exact goal vertex, a vertex from a region around the goal can be taken. The size of this region is dynamic and determined by the vertex already in the tree that is closest to the goal. In this way, the goal region is reduced in size when the planning advances. This technique is called *goal zoom*.

An alternative version of the RRT algorithm is the



(a) Simple (b) Narrow Corridor (c) Cluttered

Fig. 9. Three environments used to test the performance of the RRT planning algorithm. A solution found by the planner is shown in red. The start and goal position are indicated with the blue and red cross respectively.

Algorithm 4 Bidirectional RRT Algorithm

```
1:  $\mathcal{T}_a$.init($\boldsymbol{q}_{init}$)
2:  $\mathcal{T}_b$.init($\boldsymbol{q}_{goal}$)
3:  **while** !treesConnected **do**
4:      $\boldsymbol{q}_{rand} \leftarrow$ generateRandomState(bounds);
5:      $\boldsymbol{q}_{near_a} \leftarrow$ nearestNeighbour($\boldsymbol{q}_{rand}$,$\mathcal{T}$)
6:      $[S_a, \boldsymbol{q}_{new_a}] \leftarrow$ extendTree($q$,$\boldsymbol{q}_{near_a}$,$\mathcal{T}$)
7:      **if** $S_a$ != *Trapped* **then**
8:          $\boldsymbol{q}_{near_b} \leftarrow$ nearestNeighbour($\boldsymbol{q}_{new_a}$,$\mathcal{T}$)
9:          $[S_b, \sim] \leftarrow$ extendTree($\boldsymbol{q}_{new_a}$,$\boldsymbol{q}_{near_b}$,$\mathcal{T}$)
10:         **if** $S_b$ = *Reached* **then**
11:             **return** path($\mathcal{T}_a, \mathcal{T}_b$)
12:         **end if**
13:         swap($\mathcal{T}_a, \mathcal{T}_b$)
14:     **end if**
15: **end while**
```

bidirectional one as shown in Algorithm 4. This version makes use of two trees: one rooted at the initial pose of the platform $\mathcal{T}_a$ and one rooted at the goal pose $\mathcal{T}_b$. As in the basic algorithm, one tree is extended, but after this is done, an attempt is made to connect the second tree with the first one. If the two trees are connected, a path to the goal is found. If no connection is made, the trees are swapped and the process is repeated. The connection will again not be exact, but within the similarity threshold ϵ. This results in a discontinuity in the path at the point where the two trees meet.

The planning results in a reference trajectory that consists of a state reference and an input reference. Furthermore, the trajectory has a time schedule, which means every reference state has a time stamp related to it.

B. Method

The *coarse approach* and *alignment* stage both need specific settings for the planning algorithm. These settings are found through simulations of the RRT algorithm. Three simulated test environments as shown in Figure 9 are used to investigate the effect of several algorithm settings. The environments have the following key feature: the first contains a lot of open space, the second contains a narrow corridor, and in the third the space is cluttered with small obstacles. For each specific setting of the RRT algorithm, the planning is run 50 times and the plan time and path length of the solution path is averaged over these 50

TABLE II
PERFORMANCE ANALYSIS OF DIFFERENT VERSIONS OF THE RRT ALGORITHM. BOLD ITEMS ARE THE BEST RESULT FOR THAT SPECIFIC COLUMN.

Simulation	Simple		Narrow corridor		Cluttered	
	Time [s]	Length [m]	Time [s]	Length [m]	Time [s]	Length [m]
Single tree	32.64	145.26	467.23	173.23	74.65	159.95
Bias (0.05)	23.13	141.10	271.05	179.38	56.23	**158.12**
Bias (0.2)	17.98	**136.52**	443.31	172.45	61.78	158.42
Zoom (0.05)	20.54	140.99	424.02	189.58	58.35	161.67
Zoom (0.2)	19.49	144.73	337.57	**170.81**	47.66	159.67
Random input	45.77	156.24	**81.20**	181.99	102.80	170.93
Bidirectional	**11.99**	158.53	318.94	176.86	**47.17**	162.50
Random input	30.30	191.38	91.16	208.78	117.36	188.00

iterations to reduce the effect of the random nature of the planner. The input set used during the simulations is

$$\begin{bmatrix} u_1 \\ u_2 \\ \text{DM} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 32 \end{bmatrix}, \begin{bmatrix} 1 \\ \pm 0.05 \\ 32 \end{bmatrix}, \begin{bmatrix} 1 \\ \pm 0.1 \\ 32 \end{bmatrix}$$

Here, a $\pm$ in front of u_2 means that both a positive and negative steering angle are possible. The third entry DM indicates the drive mode of the platform, which can be four wheel steering (32), crab mode (48) or rotate in place (64). The following planner settings are tested:

- Single tree, basic version
- Single tree with a goal bias of 0.05
- Single tree with a goal bias of 0.2
- Single tree with a goal zoom bias of 0.05
- Single tree with a goal zoom bias of 0.2
- Single tree, basic version with an input set consisting of 5 random inputs.
- Bidirectional, basic version
- Bidirectional, basic version with an input set consisting of 5 random inputs.

C. Results

Results of the simulations are summarized in Table II. The time required to find a solution path and its length are reported. Clearly, the bidirectional algorithm is faster than the single tree version, except for the narrow corridor environment. However, it generally results in a longer path. For the single tree versions, a high goal bias works well in an environment containing much open space. In cluttered environments, the lower bias of 0.05 works better. For goal zoom, higher biases can be applied in every environment, but it only outperforms the goal bias version in the third environment.

VI. Feedback Control

A. Implementation

Each drive mode (four wheel steering, rotation in place, crab mode) needs a separate feedback controller, as there is no single controller that can handle all three modes. Switching between the controllers is based on the feedforward input from the path planner. A general control diagram showing the implementation of the controller within the system is shown in Figure 10.

1) ***Four wheel steering****:* The controller based on approximate linearization around the reference trajectory is implemented. This method requires the system to be in the chained form. The system (10) is first rewritten into

$$\begin{aligned}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}_{fr}\\ \dot{\theta}_w\end{bmatrix} &= \begin{bmatrix}\cos(\theta_{fr})\sin^2(\theta_w)\\ \sin(\theta_{fr})\sin^2(\theta_w)\\ \frac{\sin(2\theta_w)}{L}\\ 0\end{bmatrix}\eta_1 + \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\eta_2\\ \dot{\boldsymbol{q}}' &= \boldsymbol{g_1}\eta_1 + \boldsymbol{g_2}\eta_2 \end{aligned} \tag{23}$$

The general velocity vector $\dot{\boldsymbol{q}}$ is extended with the rate of change of the steering angle $\dot{\theta}_w$. This extended velocity vector is designated with $\dot{\boldsymbol{q}}'$. The first input to the extended system is η_1, which is equal to u_1: the platform velocity. η_2 is the rate of change of the steering angle, and hence relates to u_2 through:

$$\eta_2 = \dot{u}_2 \tag{24}$$

The chained form can be achieved through an input and state transformation, which will transform the system into the following so called 4 state, 2 input chained form

$$\begin{aligned}\dot{z}_1 &= v_1\\ \dot{z}_2 &= v_2\\ \dot{z}_3 &= q_2' v_1\\ \dot{z}_4 &= q_3' v_1\end{aligned} \tag{25}$$

Where z_i are the transformed states and v_i are the transformed inputs.

A systematic way to find the transformation needed to put the system in the form (25) is given in [26]. This method is applied to (23) and results in the following chained form system

$$\begin{aligned}z_1 &= x\\ z_2 &= \frac{\sin(2\theta_w)\sec(\theta_{fr})^2}{L}\\ z_3 &= \tan(\theta_{fr})\\ z_4 &= y\end{aligned} \tag{26}$$

$$v_1 = \eta_1\cos(\theta_{fr})\sin^2(\theta_w) \tag{27}$$

$$\begin{aligned}v_2 = &\frac{2\eta_1\sin(\theta_{fr}) - 2\eta_1\cos(2\theta_w)^2\sin(\theta)}{L^2\cos(\theta_{fr})^3}\\ &+\frac{2L\eta_2\cos(2\theta_w)\cos(\theta_{fr})}{L^2\cos(\theta_{fr})^3}\end{aligned} \tag{28}$$

Now that the system is in chained form, the control law can be derived. First, the state and input errors are defined as

$$\tilde{z}_i = z_i - z_{ir} \tag{29}$$

$$\tilde{v}_j = v_j - v_{jr} \tag{30}$$

Where the subscript r denotes the reference value of the path that needs to be followed which is a result from the path planning algorithm.

The system's error equations now become

$$\begin{aligned}\dot{\tilde{z}}_1 &= \tilde{v}_1\\ \dot{\tilde{z}}_2 &= \tilde{v}_2\\ \dot{\tilde{z}}_3 &= z_2 v_1 - z_{2r}v_{1r}\\ \dot{\tilde{z}}_4 &= z_3 v_1 - z_{3r}v_{1r}\end{aligned} \tag{31}$$

Linearizing this system about the reference trajectory results in a time-varying linear system

$$\begin{aligned}\dot{\tilde{\boldsymbol{z}}} &= \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&v_{1r}&0&0\\0&0&v_{1r}&0\end{bmatrix}\tilde{\boldsymbol{z}} + \begin{bmatrix}1&0\\0&1\\z_{2r}&0\\z_{3r}&0\end{bmatrix}\tilde{\boldsymbol{v}}\\ \dot{\tilde{\boldsymbol{z}}} &= \boldsymbol{A}\tilde{\boldsymbol{z}} + \boldsymbol{B}\tilde{\boldsymbol{v}}\end{aligned} \tag{32}$$

Now, regular control laws for time-varying linear systems can be used. The following feedback law is established

$$\tilde{v}_1 = -k_1\tilde{z}_1 \tag{33}$$

$$\tilde{v}_2 = -k_2\tilde{z}_2 - \frac{k_3}{v_{1r}}\tilde{z}_3 - \frac{k_4}{v_{1r}^2}\tilde{z}_4 \tag{34}$$

The complete control input is given by

$$v = v_r + \tilde{v} \tag{35}$$

To find the actual input u_i to the platform, the input transformation (27) and (28) can be inverted to find η_1 and thus u_1, and η_2. From (24) it follows that η_2 needs to be integrated to find u_2. The choice of (27) shows that the input transformation is only defined for the platform heading $\theta_{fr} \neq \pi/2 \pm k\pi$ with $k \in \mathbb{N}$. This means the controller will not work for platform headings close to $\pi/2 \pm k\pi$: the velocity will go to zero. To solve this, the controller is disabled and only the feedforward command is used for $\theta_{fr} = \pi/2 \pm 0.1$. Furthermore, note that the choice of (34) implies that $v_{1r} \neq 0$ and thus $u_{1r} \neq 0$.

2) ***Crab mode****:* The controller based on approximate linearization is not able to handle crab mode. A control technique that can be applied whether the system is non-holonomic or not, is PID control. Two separate error signals need to be defined: one for the steering input and one for the velocity input. Then, two PID controllers can be applied based on this error. Note that heading control is not possible in this drive mode, as the front and rear steering angle are always equal.

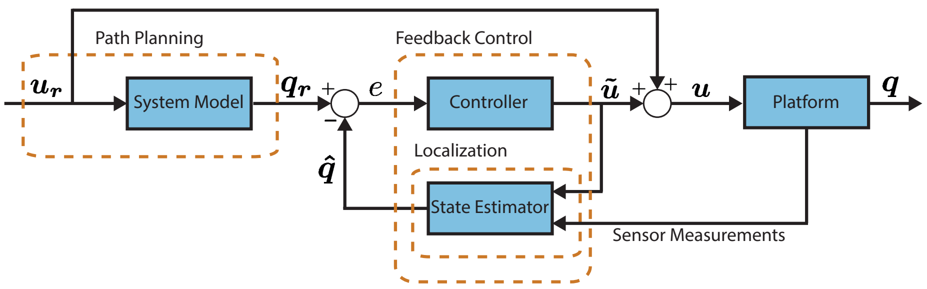


Fig. 10. Control diagram of the autonomous navigation algorithm for the INTERACT rover. The three main aspects of the algorithm are indicated. The feedforward input u_r is used when the four wheel steering controller is active. For the PID controllers, no feedforward command is used and the output of the controller is $\boldsymbol{u}$ directly.

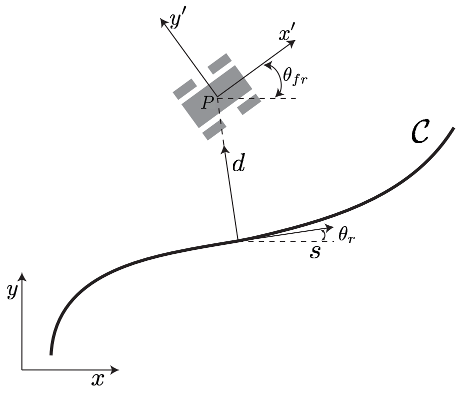


Fig. 11. Definition of the Frénet frame. The new position coordinates are s and d. The third state, heading, is defined by the difference between the platform heading and the reference heading: $\theta_e = \theta_r - \theta_{fr}$.

To simplify the definition of the two error signals for the controllers, the system pose is transformed to a so called "Frénet" frame. This frame is attached to the reference path $\mathcal{C}$ found by the planning algorithm as is shown in Figure 11. Because the reference trajectory has a time schedule, the origin of the Frénet frame can be attached to the reference point of the current time instant. The abscissa s of the frame points in the direction of the current reference heading $\theta_{fr,r}$. The ordinate d is perpendicular to this, defining a right-handed frame. The position coordinates of the platform in the Frénet frame can be found by applying a transformation matrix to the platform coordinates in world frame

$$\begin{bmatrix} s \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_r) & -\sin(\theta_r) & x_r \\ \sin(\theta_r) & \cos(\theta_r) & y_r \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{36}$$

The error signals used for the PID controllers are now defined as follows. The first error related to the platform velocity is given by the coordinate s. The second error, related to the steering input, is given by the lateral deviation d from the path. Based on these two errors, two PID controllers can be implemented

$$u_1 = -k_{p1}s - k_{i1}\sum_{i=1}^{k} s(t_i)\Delta t - k_{d1}\frac{s(t_k) - s(t_{k-1})}{\Delta t} \tag{37}$$

$$u_2 = -k_{p2}d - k_{i2}\sum_{i=1}^{k} d(t_i)\Delta t - k_{d2}\frac{d(t_k) - d(t_{k-1})}{\Delta t} \tag{38}$$

*3) **Rotation in place**:* During a rotation in place maneuver, only one input is available: the velocity command u_1. A PID controller is defined which uses θ_e as error input. This is the difference between the reference heading $\theta_{fr,r}$ and the actual heading estimated by the localization algorithm using (21)

$$\theta_e = \theta_r - \theta_{fr}. \tag{39}$$

The PID feedback law becomes

$$u_1 = -k_{p3}\theta_e - k_{i3}\sum_{i=1}^{k} \theta_e(t_i)\Delta t - k_{d3}\frac{\theta_e(t_k) - \theta_e(t_{k-1})}{\Delta t} \tag{40}$$

B. Method

For the controller for four wheel steering, not every choice of the gains k_i will result in a stable system. The gains are chosen after performing a stability analysis on the closed loop system using the Hurwitz stability theorem, and choosing the gains such that critical damping $\zeta = 1$ is achieved. These gains are tested during experiments with the platform.

The gains for the crab mode PID controller are tuned using the simulated system. The gains are chosen such that the response of the system on a diagonal path as shown in Figure 12 shows accurate tracking and the computed input to the platform shows no significant overshoot and settles within 2 seconds.

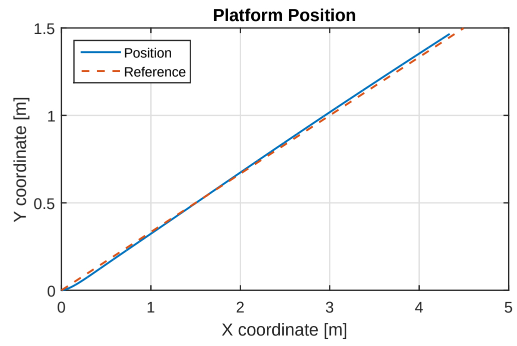
The gains for the turn in place controller are tuned based on the response of the system on a reference slope as shown in Figure 13 which is representative for the rate of change in heading expected during navigation. They are chosen such that the tracking error never exceeds 0.05 radians after 4 seconds. When the controller is used during reference trajectory tracking, it is always given this time to converge to the reference point.

C. Results

The stability analysis for the four wheel steering controller results in the following values for the k_i:

$$\begin{aligned} k_1 &= 5 \\ k_2 &= 15 \\ k_3 &= 75 \\ k_4 &= 125 \end{aligned}$$

Because the controller is based on linearization, it only provides local asymptotic stability. Accurate determination



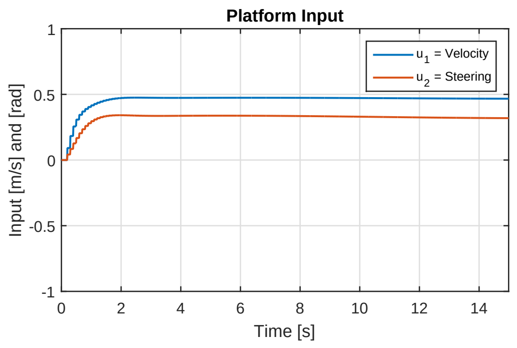


Fig. 12. Response of the controller for the crab mode on a diagonal reference path. The reference path is given by a dashed orange line. The actual position by the blue solid line. The velocity input u_1 is shown by the blue line in the lower graph. u_2, the steering command, is given by the orange line.

of the region of stability is difficult, but simulations and experiments show that it is quite large (see also [28]). This means that the system will converge to the reference trajectory even for quite large errors, and can thus handle the discontinuities resulting from the path planning. However, transient behaviour is unpredictable and might result in collisions, so the discontinuity should preferable be kept small.

For the crab mode controller, the gains are:

$$\begin{aligned} k_{p1} &= 3 \quad k_{i1} = 0 \quad k_{d1} = 0 \\ k_{p2} &= 5 \quad k_{i2} = 2 \quad k_{d2} = 0 \end{aligned}$$

The gains for the PID controller are:

$$k_{p3} = 5 \quad k_{i3} = 0.7 \quad k_{d3} = 0$$

The oscillation during the start visible in Figure 13 is caused by the fact that the platform will only rotate in place when the velocity input exceeds 0.3 m/s. This is likely caused by friction in the mechanical system and wheel-to-ground interaction forces. The limitation is incorporated in the controller by adding a viscous friction block in Simulink. However, as a result, a very small error at the start of the reference will make the controller compensate with an input of 0.3 m/s minimum, which causes the oscillation. The amplitude of the oscillation is within the 0.05 radian limit and hence acceptable.

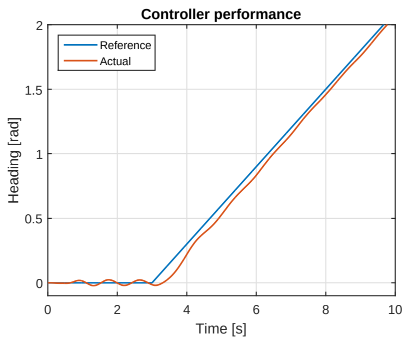


Fig. 13. Response of the controller for rotate in place on a reference slope. The reference heading is given in blue. The actual heading (measured by the gyroscope) is given in orange.

VII. Experimental Evaluation

A. Implementation

For the experiments with the complete autonomous navigation algorithm, the path planning requires specific settings for the *coarse approach* and the *alignment*. These settings follow from the simulations described in Section V and are listed below. The gains for the controllers were as reported in subsection VI-C and identical for both stages.

*1) **Coarse Approach Settings**:* For this stage, the single tree version of the RRT algorithm is used with a goal bias of 0.05. The similarity threshold ϵ is set to the distance at which the taskboard can be recognized, which is 5 meter. The position and heading gain for the distance function (22) are set to $k_{pos} = 1$ and $k_\theta = 2$. The integration time $\Delta t = 2$ and a total of 7 motion primitives are selected. As no accurate maneuvering is necessary, it was decided to not include rotation in place motions as to not excessively switch between control modes.

$$\begin{bmatrix} u_1 \\ u_2 \\ \text{DM} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.2 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.3 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.4 \\ 48 \end{bmatrix}$$

*2) **Alignment Settings**:* For the alignment, the bidirectional version is used. This was done because this stage needs to result in an exact alignment of the platform in front of the taskboard. As stated before, it is virtually impossible to exactly reach the goal location due to the nonholonomic constraints of the platform. The goal is only reached with an accuracy determined by the similarity threshold ϵ. Choosing the bidirectional algorithm will relocate this problem to the point where the trees meet (i.e. in the middle of the path). Although this causes a discontinuity, the end of the path is now exactly the goal position. The similarity threshold is set to $\epsilon = 0.1$ to keep the discontinuity in the path small, such that the controller described in Section VI can handle it. The position and heading gain for the distance function are unchanged. $\Delta t = 1$ and a total of 9 motion primitives are selected. Both crab mode and rotation in place are included in the set. The turn in place input allows rotation in both

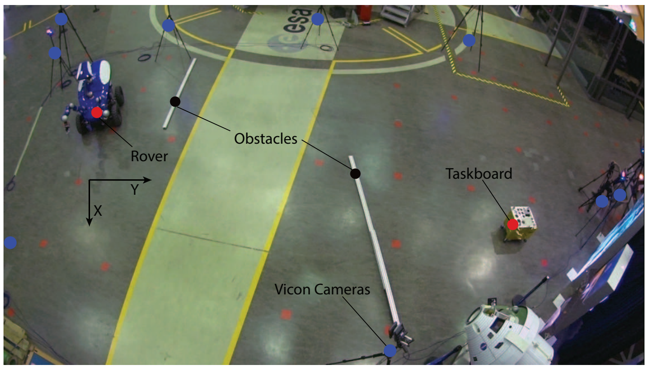


Fig. 14. Image of the workspace used during the navigation experiments. The rover and taskboard are both indicated with a red dot. Beams used as obstacles are indicated with black dots. 9 Vicon cameras, indicated in blue, are positioned around the perimeter of the workspace.

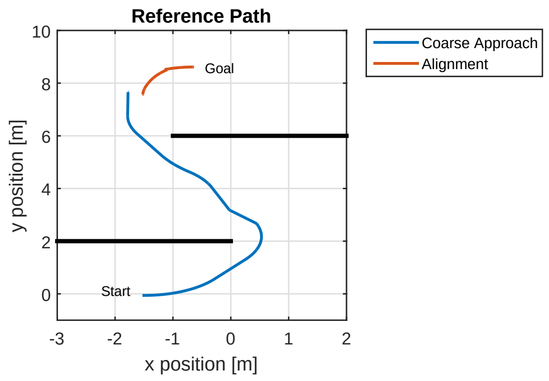


Fig. 15. Plot of a possible path from start to goal position. The path for the coarse approach is plotted in blue. The alignment path is plotted in orange. Because the position measurement w.r.t. the taskboard is available when planning for the alignment stage, the localization error gets reset to 0. This explains the small jump in the platform position.

CW and CCW direction.

$$\begin{bmatrix} u_1 \\ u_2 \\ \text{DM} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.2 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.25 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.4 \\ 48 \end{bmatrix}, \begin{bmatrix} \pm 0.5 \\ 0 \\ 64 \end{bmatrix}$$

B. Method

The autonomous navigation algorithm as described in this paper is implemented on the actual platform and tested. The workspace used during the experiments was roughly 10 by 6 meter. 9 Vicon cameras were positioned along the perimeter of the workspace. The platform and the taskboard were positioned as far as possible from each other within this workspace as is shown in Figure 14. Obstacles could be added to the planning algorithm to show obstacle avoidance capabilities and increase path length. The planning was done off-line and separated into the *coarse approach* and *alignment* stage. The detection of the taskboard in the *alignment* stage is simulated by using the Vicon system. The resulting reference trajectory and input from the planning were uploaded to an on-board computer on the platform. The Simulink model for platform control was run in real time on this on-board computer. The complete path from the initial position to taskboard was in the order of 20 meter.

During the *coarse approach*, the controller performance is separated from the localization performance by using two different error metrics. The *Localization error* $\boldsymbol{e}_{\text{loc}}$ is defined as the difference between the localization pose estimate and the actual pose as measured by the Vicon motion capture system.

$$\boldsymbol{e}_{\text{loc}} = \boldsymbol{q}_{\text{loc}} - \boldsymbol{q}_{\text{vicon}} \tag{41}$$

This error represents the performance of the localization algorithm, and is unknown to the system. The *tracking error* $\boldsymbol{e}_{\text{track}}$ is defined by the difference between the localization estimate and the reference trajectory and hence represents the controller performance

$$\boldsymbol{e}_{\text{track}} = \boldsymbol{q}_{\text{loc}} - \boldsymbol{q}_{r} \tag{42}$$

If the controller tracks the path perfectly, it is zero. This error is known to the system and hence the execution can be terminated if the error gets too large (i.e. exceeds the minimum turning radius).

The performance of the navigational algorithm as a whole is assessed using the *total error*. This error is computed by taking the difference between the actual location of the platform according to the Vicon system and the desired location given by the path planning algorithm. The final value of this error after execution of the *alignment* stage should be within the limits of 15 cm in position and 6° in heading.

Other important variables are the drive mode, which determines which controller is used to track the path, and the input to the platform computed by the controller.

During the experiments, 8 trials of the *coarse approach* and 10 trials of the *alignment* are executed. The initial position for the *coarse approach* is held constant to within a couple of centimeters and degrees. The path found by the planning algorithm is different for every trial but the configuration of the obstacles and taskboard is identical. An example path found by the planner is shown in Figure 15. The location of the platform at the start of the *alignment* stage is varied, to test if accurate alignment is possible from several angles and positions w.r.t. the taskboard.

C. Results

The final tracking error and localization error of the 8 *coarse approach* trials and the average over all trials are given in Table III. The tracking of the reference trajectory is always well within the minimum turning radius of 1 meter and the obstacles could be avoided safely. However, localization errors are quite high: in the order of 0.5 meter. Taking into account that the travelled distance is in the order of 20 meter, this error exceeds the 0.5% limit by far. The graphs of the development of the tracking and localization errors over time of one of the runs are given in Figure 16. This indeed shows that the tracking error is within limits at all times, but the localization error is diverging rapidly.

The final errors of the 10 *alignment* trials along with the average are given in Table IV. Bold entries show the values

TABLE III
RESULTS OF COARSE APPROACH STAGE

	Abs. Tracking Errors			Abs. Localization Errors		
Trial	x [m]	y [m]	θ_{fr} [°]	x [m]	y [m]	θ_{fr} [°]
1	0.384	0.302	2.550	0.082	0.574	0.745
2	0.113	0.134	0.544	0.010	0.443	0.476
3	0.155	0.265	9.546	0.332	0.424	0.974
4	0.185	0.055	2.074	0.009	0.415	1.089
5	0.133	0.126	2.922	0.057	0.358	0.854
6	0.165	0.434	21.76	0.020	0.545	1.874
7	0.093	0.267	2.487	0.205	0.395	1.742
8	0.102	0.655	0.355	0.278	0.418	2.521
Average	0.166	0.280	5.280	0.124	0.446	1.284

TABLE IV
RESULTS OF ALIGNMENT STAGE. BOLD ENTRIES ARE NOT WITHIN THE SPECIFIED ACCURACY OF 15 CM IN POSITION AND 6° IN HEADING

	Absolute Final Errors			
Trial	x [m]	y [m]	θ_{fr} [°]	Within limits
1	**0.162**	0.079	0.018	no
2	0.114	**0.212**	**7.523**	no
3	**0.152**	0.046	1.169	no
4	0.060	**0.359**	3.197	no
5	0.006	0.050	4.165	yes
6	**0.426**	**0.159**	5.139	no
7	0.112	0.098	2.236	yes
8	0.140	0.016	**11.61**	no
9	0.045	**0.152**	2.022	no
10	**0.221**	0.135	**9.848**	no
Average	0.140	0.132	4.693	yes

that are not within the specified bounds. Although many of the trials are close to the required bounds, only 2 out of 10 trials reach the final position with enough accuracy. On average, the accuracy is 14 cm in x, 13 cm in y, and 5° in heading, which is within the specified limits.

Figure 17 shows the comparison of the actual position with the reference, and the total error for one of the alignment trials. In 17a, a significant deviation from the reference can be observed in the middle segment of the path. This deviation coincides with a discontinuous jump in the heading error in Figure 17b around $t = 7$. The error is mostly compensated for during the remainder of the path, but an error in x of about 0.4 meter is still present after execution is done. This results in the total error being outside the required bounds for performing the manipulation tasks.

The set of results shown in Figure 18 is a special case of the final alignment stage where the platform is placed with a lateral offset from the platform. Figure 18a shows the this lateral offset w.r.t. the taskboard. The black arrows visualize the heading of the platform at equally spaced time intervals. The positioning is similar to a parallel parking maneuver, however, the platform is not capable of reversing its motion. Hence, the turn in place drive mode is used extensively, as can be seen from Figure 18d. The tracking error shown in Figure 18b has the largest magnitude (about 0.5 meter) around half way. The final error is in the order of 5 cm. This is well within the bounds required for manipulation without relocation.

VIII. DISCUSSION

From Table IV, it can be seen that the final position of the platform is not always within the specified bounds. This shows that the performance of the algorithm is not consistent enough and needs to be made more robust. The proposed algorithm is, however, a solid basis for the development of an improved version.

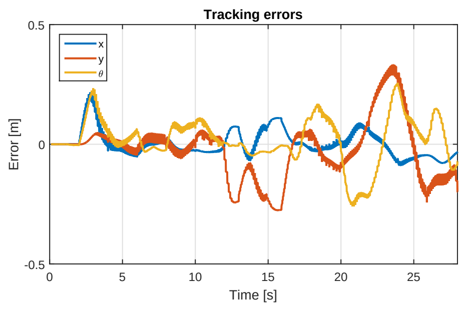
There can be many explanations for failure of the algorithm and many aspects of it can still be improved. However, three main causes that can explain the failure of reaching the goal are considered: (1) the discontinuity that is present in the path used for the final alignment. (2) Too much error accumulation in the localization estimate. (3) The fact that the path has a time schedule. These causes and possible solutions will be discussed next.

The discontinuity in the reference path caused by the connection of the two trees used in the bidirectional RRT algorithm can result in severe tracking errors. This is especially apparent in Figure 17a. In this particular case, the discontinuity is mainly present in the heading, which can be seen around $t = 7$ in Figure 17b. Although the controller does not become unstable, it needs quite some time to compensate and reduce the tracking error. It is not able to do this before the path ends and an error in x of about 0.5 meter remains.

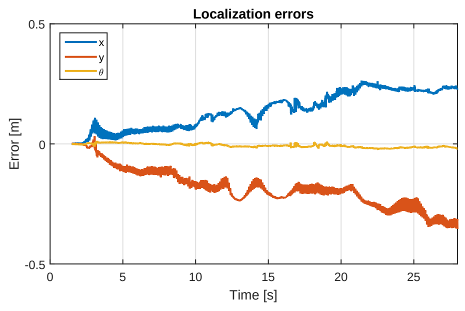
The issue can be solved by reducing or completely removing the discontinuity. Reduction can be achieved by making the bounds for connecting the trees more stringent (i.e. reduce ϵ). Alternatively, the cost function could be adapted such that only discontinuities that are "easy" to correct for with regard to the nonholonomic constraints are allowed. For example, a discontinuity of 20 cm in the direction of motion results in a much smaller tracking error than the same discontinuity in a lateral direction. To completely remove the discontinuity one would have to look to other planning algorithms or controllers that can stabilize a certain pose of the system. However, these controllers generally are not capable of incorporating obstacle avoidance.

Unbound error accumulation in the position estimate can only be solved by adding a sensor to the system that provides an absolute position measurement. This will likely also ensure sufficient accuracy on rough terrain, a scenario that was not experimentally validated in this study. In this case, the preferred solution would be a stereo camera pair or a LIDAR sensor. Besides reducing the localization error, these sensors also make obstacle detection possible. If the addition of such a sensor is not an option, increased accuracy could be achieved by reducing wheel slip through the use of Ackermann steering (currently the platform is not capable of this), or better wheel slip detection.

Finally, it seems that the tracking accuracy sometimes suffers from the fact that there is a time schedule attached to the reference path. If the platform somehow accumulates a delay during tracking, as is the case due to the excessive error in the tracking of the path of Figure 17, it will lag behind the reference. If the controller tries to compensate for this, strong and possibly erratic input is required, which sometimes results in saturated input to the platform. Even with this correction, the platform lags behind and although

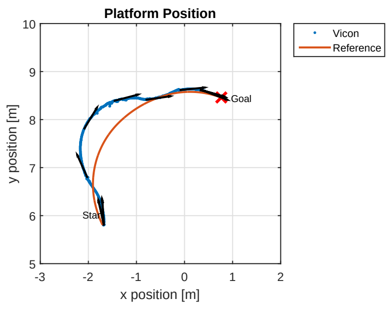


(a)

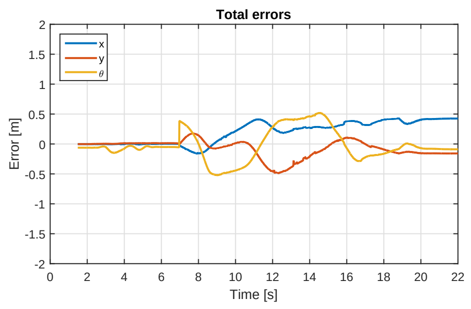


(b)

Fig. 16. Tracking (a) and localization error (b) of one of the coarse approach trials. The errors are separated in x (blue), y (orange) and θ_{fr} (yellow).

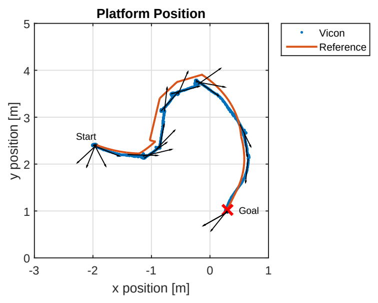


(a)

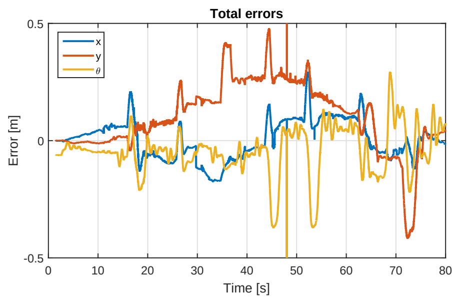


(b)

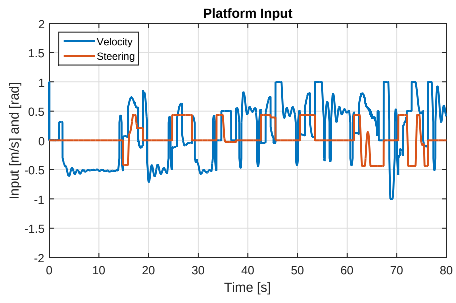
Fig. 17. Position comparison (a) and total error (b) of one of the alignment trials. In (a), the heading of the platform is indicated at equally spaced time intervals by the black arrows. The errors in (b) are separated in x (blue), y (orange) and θ_{fr} (yellow).



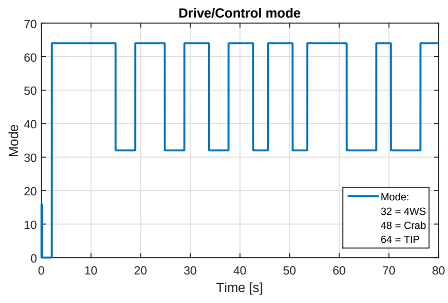
(a)



(b)



(c)



(d)

Fig. 18. Position comparison with heading indication (a), total error (b), platform input (c), and drive mode (d) of one of the alignment trials. In (a), the heading of the platform is indicated by black arrows at equally spaced time intervals. The peak in the total error graph (b) around $t = 48$ is because the platform was out of view of the Vicon setup for one time sample.

it converges to the reference, there is still an error visible. When the path comes to an end, the controller is stopped and is not given the time to compensate. It might be better if there would not be a time schedule. The reference point for the controller would then be established by finding the closest point on the path. This might solve the issue seen in the trial shown in Figure 17. However, it makes the use of a feedforward input more difficult.

IX. Conclusion

In this paper, a navigational algorithm was presented and experimental results of its implementation on the INTERACT platform were given. The goal was to position the platform in front of the taskboard such that manipulation tasks could be performed without relocating the platform.

From the experimental results, the following conclusions can be drawn:

- A algorithm capable of performing end-to-end navigation for a four wheel steering, four wheel drive platform has been developed.
- The UMBmark calibration method is successfully applied to a four wheel steering platform.
- The required accuracy on the final position can be reached. However, the algorithm is not consistent enough and fails in several occasions.
- The RRT path planner can incorporate all drive modes of the platform.
- The three path tracking controllers are capable of keeping the tracking error well within the maximum value of 1 meter (the minimum turning radius).

Nomenclature

A	Location of the steering axis of rotation.
B	Location of the center of the wheel, also the ground contact point.
C	Conversion factor between encoder count and distance [m/pulse].
L	Distance between the front and rear axle of the system [m].
N	Cumulative encoder count.
N_{rev}	Number of encoder counts/pulses per revolution.
P	Reference point on the chassis of the platform.
S	Flag for the RRT algorithm indicating *reached*, *trapped* or *advanced*.
ΔN	Incremental encoder count.
$\Delta\psi$	Incremental change in the side slip angle ψ during one odometry time sample [rad].
$\Delta\theta_{fr}$	Incremental heading change [rad].
Δd	Incremental distance [m].
Δd_L	Average incremental distance of left wheels [m].
Δd_R	Average incremental distance of right wheels [m].
Δt	Time step or sample time.
Δx	Incremental distance in x [m].
Δy	Incremental distance in y [m].
α	Wheel geometry parameter: the angle between the x'-axis and the line PA [rad].
β	Wheel geometry parameter indicating the angle between the extension of the line PA and the wheel plane [rad].
$\boldsymbol{C}$	Matrix containing the no side-slip constraints.
$\boldsymbol{J}$	Matrix containing the pure rolling constraints.
$\boldsymbol{R}(\theta)$	Rotation matrix representing a rotation of θ around the z-axis.
$\boldsymbol{S}$	Matrix containing the part of the equations of motion depending on the velocity.
$\boldsymbol{\Gamma}$	Matrix containing the part of the equations of motion depending on the rate of change of the steering angles.
$\boldsymbol{\Sigma}$	Matrix containing the allowed velocities of the nonholonomic system.
$\boldsymbol{\eta}$	System input vector for the approximate linearization controller.
$\boldsymbol{q}_{init}$	Initial state/vertex of for the RRT tree.
$\boldsymbol{q}_{near}$	Nearest state/vertex to $\boldsymbol{q}_{rand}$.
$\boldsymbol{q}_{new}$	New state/vertex that gets added to the RRT tree.
$\boldsymbol{q}_{rand}$	Random state/vertex used in the RRT algorithm.
$\boldsymbol{u}$	Platform input vector used in the mathematical model. u_1 is the velocity input, u_2 is the steering angle.
$\boldsymbol{v}$	System input after an input transformation.
$\boldsymbol{v}_r$	Reference or desired value for the transformed system input $\boldsymbol{v}$.
$\boldsymbol{z}$	System state after a state transformation.
$\boldsymbol{z}_r$	Reference or desired value for the transformed system state $\boldsymbol{z}$.
$\dot{\boldsymbol{\xi}}$	Time derivative of the extended system state.
$\dot{\boldsymbol{q}}$	Time derivative of system pose: generalized velocities.
$\boldsymbol{\xi}$	Extended system state containing system pose $\boldsymbol{q}$ and wheel rotation.
$\boldsymbol{q}'$	System pose including steering angle. Used for four wheel steering controller.
$\boldsymbol{q}$	System pose containing position and heading: Generalized coordinates.
$\dot{\phi}_w$	Angular rate of change of the wheel about its horizontal axis [rad/s].
$\dot{\theta}_w$	Rate of change of the steering angle [rad/s].
$\dot{\theta}_{fr}$	Angular rate of change of the systems heading [rad/s].
$\dot{x}$	Velocity in the x direction in world frame [m/s].
$\dot{y}$	Velocity in the y direction in world frame [m/s].
$\dot{\boldsymbol{v}}$	Derivative of the system input after transformation.
$\dot{\boldsymbol{z}}$	Derivative of the transformed system state.
$\hat{\boldsymbol{q}}$	Estimated system pose containing position and heading.
$\mathcal{T}$	Tree of the RRT algorithm containing the states as vertices and edges between them.
ϕ_w	Rotation angle of the wheel about its horizontal axis [rad].
ψ	Side slip angle of the platform [rad].
θ_e	Heading error signal for the rotation in place control.
θ_w	Steering angle of the wheel, i.e. rotation about its vertical axis measured from the x'-axis [rad].
θ_{fr}	Heading of the system, i.e. angle between the robot frame and the world frame [rad].
$\tilde{\boldsymbol{v}}$	Input error of the transformed system $\boldsymbol{z}$ with input $\boldsymbol{v}$.
$\tilde{\boldsymbol{z}}$	State error of the transformed system $\boldsymbol{z}$.
a	Distance from the reference point P to the front wheel pair axis [m].
b	Distance from the reference point P to the rear wheel pair axis [m].
d	Ordinate of the Frénet frame attached to the reference trajectory.
k_θ	Heading gain for the distance metric ρ.
k_d	Derivative gain for a PID controller.
k_i	Integral gain for a PID controller.
k_{pos}	Position gain for the distance metric ρ.
k_p	Proportional gain for a PID controller.
l	Wheel geometry parameter: distance between the reference point P and the point steering axis A [m].
r_w	Wheel radius [m].
s	Abscissa of the Frénet frame attached to the reference trajectory.
w	Wheel base; distance between the ground contact points of the left and right wheel of a pair [m].
w_A	Steer axis base; distance between the steering axes of the left and right wheel of a pair [m].
w_o	Wheel offset from steering axis [m].
x	x position coordinate in world frame [m].
x'	x position coordinate in the robot frame [m].
y	y position coordinate in world frame [m].
y'	y position coordinate in the robot frame [m].

Appendix I
Introduction to the Thesis Project

Figure 19 shows the navigational problem to be solved by the algorithm. The platform is initially positioned at an arbitrary location in the environment and needs to navigate to the taskboard. It is assumed that the platform cannot identify all obstacles autonomously, and it is left for the astronaut to exercise his human judgement on which areas of the environment are save to traverse. For this purpose, the astronaut receives a bird's-eye view similar to Figure 19a to indicate the obstacles and goal position. After this is done, a path planning algorithm is required to find a feasible path to the goal.

Path Planning

The path planning should take the nonholonomic constraints of the system into account and plan a path that can be tracked exactly by the platform (i.e. the path should be *feasible*). Furthermore, the planning should take the obstacles into account and plan the path around these as shown in Figure 19b. Path planning algorithms can be roughly divided into two groups: sampling based planners, which encompass probabilistic roadmap methods (PRM) and Rapid Exploration Random Trees (RRTs), and combinatorial planners, which do not discretize the configuration space and hence find an exact solution. The former group applies sampling of the configuration space and thus generates a discretized graph which can be searched for a solution using graph search algorithms such as Dijkstra's algorithm, A* and D*. Examples are Obstacle Based PRM (OBPRM) [2], lazy PRM [3], Medial Axis PRM (MAPRM) [4] and RRT algorithms [5], [6]. The rapid exploration random tree algorithm is especially suitable for nonholonomic planning problems [7] as it can use the equations of motion of the system to generate the path. A drawback of sample based planners is that they are not complete, i.e. they do not always find a solution if one exists (or require infinite time to find one), and are not able to report that no solution exists. On the contrary, combinatorial methods such as the potential field method [8] and the skeleton based method [9] are complete, but require more computation time. A thorough overview of planning methods is given in the book by La Valle [10].

Localization

When a path is found, the platform can start its motion, and the path should be tracked as accurately as possible. This will require a feedback controller to cope with inaccuracies in the system model and external disturbances. The feedback signal for the controller consists of the platform *pose* $\boldsymbol{q} = \begin{bmatrix} x & y & \theta_{fr} \end{bmatrix}$: two position coordinates x and y in the world frame and the platform heading θ_{fr} measured between the x-axis and the longitudinal axis of the platform. Hence, the platform pose needs to be measured or estimated, which is called localization. However, absolute position sensors such as GPS are not available in the extraterrestrial environment where an evolution of the INTERACT rover is supposed to operate. Several other sensors that may provide a solution in these environments such as sun sensors [11] or LIDAR are not available on this platform. Hence, localization for INTERACT can only rely on the on-board proprioceptive sensors such as wheel encoders and the inertial measurement unit (IMU). Because these measurements are relative, they suffer from unbounded error accumulation as indicated by the dashed red line in Figure 19c. The navigational algorithm should be able to cope with this increasing error in order to avoid obstacles safely.

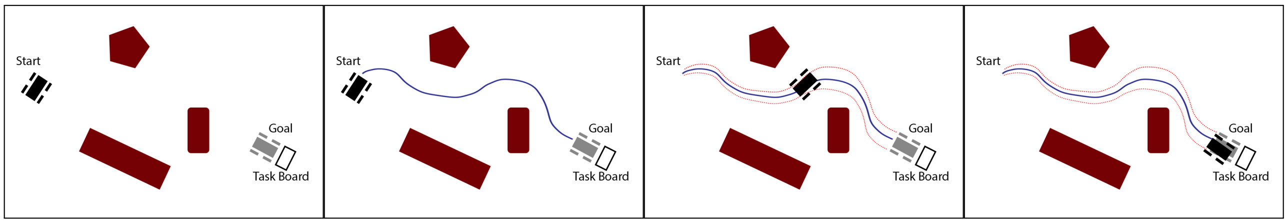
A detailed overview of localization techniques is given in [12]. Within the topic of localization, much attention is focussed on the fusion of multiple sensor signals into an accurate location estimate. The fusion of the measurements is done using a Kalman Filter, but other approaches such as fuzzy logic are also possible [13], [14]. The fusion techniques are mostly applied to fuse high frequency proprioceptive measurements from wheel encoders and IMUs, with lower frequency absolute position measurements from e.g. a GPS or camera system using active beacons. In [15], a method is described to aid IMU measurements with accurate GPS position and velocity measurements for the localization of high speed road vehicles. In [16], an autonomous cargo handling system is equipped with a RADAR system to aid the dead-reckoning position estimate.

Because dead-reckoning and odometry are the basis for every localization algorithm, much effort is put into reducing systematic and non-systematic errors present in this technique. Systematic errors can be reduced by platform parameter calibration. The University of Michigan Benchmark (UMBmark [17]) is a widely used platform calibration method targeting the two main contributions to systematic errors: unequal wheel diameter and uncertainty about the wheel base (see e.g. [18], [19], [20]). This method, however, has never been applied to a four wheel steering platform, but only differential drive robots and car-like systems. Non-systematic errors are mainly caused by wheel slip and can be minimized by detection and compensation thereof, which is described in [21] with the use of so called ”Expert Rules” and fuzzy logic, and in [22] by fusion of IMU and odometry data.

Control

Controlling nonholonomic systems is complicated by the non-linear system equations and by the fact that the platform is under-actuated: the three dimensional system pose needs to be controlled with only two input variables, velocity and steering angle. A suitable controller that is able to handle these difficulties is required to track the path.

Despite the complication due to the nonholonomic constraints, controllers can be developed that track reference trajectories and even stabilize the system at a certain pose by using linearization techniques and discontinuous or time-varying feedback [23]. In Chapter 34 of [24], several control algorithms are derived and applied to a unicycle and car-like system. This is first done through general methods not specific to nonholonomic systems (such as PID controllers), which are very limited and generally do not apply heading control. Several methods especially developed for nonholonomic systems are then introduced. These require the system to be in chained form [25], [26], [27] and apply



(a) Define obstacles, goal position (b) Path Planning (c) Path Tracking (d) Taskboard Reached

Fig. 19. Sequence showing the several steps of the navigational problem. 19a shows the starting conditions for the algorithm with obstacles (red) and goal position (grey) defined. 19b shows the planning of a path (blue) towards the goal. 19c shows execution of the paths with accumulating tracking/localization errors (dashed red). 19d shows the platform arriving at the taskboard.

some form of (feedback) linearization. Reference trajectory tracking including heading control can be achieved through approximate linearization around this trajectory, which is described in [28]. However, this is not done for four wheel steering platforms.

Stabilization of a certain pose can be done through dynamic feedback linearization [29], [30] and sinusoidal inputs [31], [32]. More advanced methods using Lie Groups [33] are capable of stabilizing both positions and trajectories. In this case, the trajectories do not have to be feasible, i.e. the controller can handle infeasible paths which cannot be exactly tracked. These methods are tested on simulated platforms, but are not experimentally validated.

To the authors knowledge, an algorithm governing every single step of the navigational problem and its implementation on an actual platform has not been presented in literature before. Furthermore, it is a challenge to achieve the necessary accuracy in localization based on proprioceptive sensors alone. Finally, a feedback controller based on approximate linearization has not been applied yet to a four wheel steering, four wheel drive system.

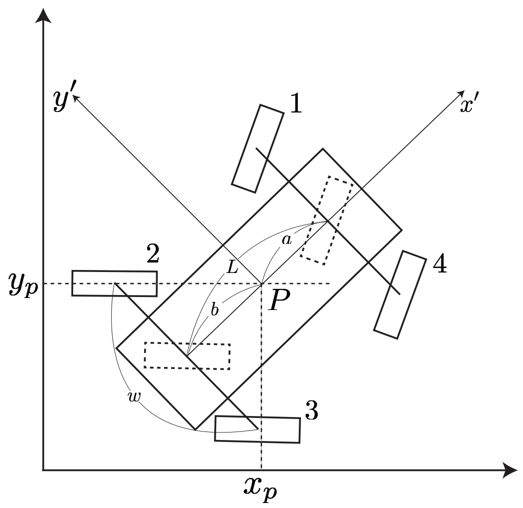


Fig. 20. Sketch of the simplification towards a bicycle model and the parameters that define the INTERACT system. The image shows the world frame with the coordinates (x_p,y_p) of the reference point P on the chassis. The robot coordinate frame is centered on P and indicated with primes. Both the two front wheels as well as the two rear wheels are collapsed into a virtual wheel centered on the respective axis (dashed wheels). The distance from P to the front and rear wheel are a and b respectively. The wheel base is given by w.

Appendix II
System Modelling

This appendix describes the derivation of the configuration kinematic model for the INTERACT platform manufactured by Ambot/Symbotics. It is deemed sufficiently accurate to only model the kinematics and leave out the dynamics of the system. Since, the speeds at which the platform will be operated are relatively low (in the order of 1 m/s), which will greatly reduce the contribution of dynamic effects. Furthermore, including the dynamics will greatly complicate the modelling and the computations herein. Finally, although including the dynamic effects might give a better correspondence with reality, it requires detailed knowledge of, for example, wheel-ground interaction and platform/motor parameters and constants. This information is not available and/or not accurately known, which counters the possible gain in accuracy by including dynamics.

The system model that is derived goes, in some aspects, beyond the current capabilities of the INTERACT platform. The model represents a system capable of both parallel and Ackermann steering, with independent steering angles of the front and rear wheel set. At this point, the platform itself is not capable of Ackermann steering and will always have equal but opposite steering angles on the front and rear wheel set in four wheel steer mode. The modelling is done in this way to keep the derivation more general and to make future improvements to the platform (which are already announced by the manufacturer) easier to incorporate.

A. Configuration Kinematic Model

The Ambot platform is a four wheel steer, four wheel drive system. The derivation of the model is mainly based on [34], [40], [24], [28]. The configuration kinematic model of the platform not only contains the state $\boldsymbol{q} = \begin{bmatrix} x & y & \theta_{fr} \end{bmatrix}^T$ describing the position and heading of the system, but is extended with the angular position ϕ of the wheels, which is needed to simulate odometry/dead-reckoning in a later stadium. This extended state is designated with $\boldsymbol{\xi}$, or the generalized coordinates of the system. The modelling and simulation are done in such a way that both Ackermann and parallel steering are possible.

Derivation of the model starts with the nonslip and pure rolling constraints. The nonslip condition is a nonholonomic constraint (see e.g. [28], [41], [42], [35]), which means it is a non integrable constraint that cannot be expressed as a relation between the generalized coordinates, but only between the generalized velocities. It states that the velocity vector of the wheel must be lying in the plane of the wheel, i.e. there is no sideslip allowed. The second constraint indicates that the wheel does not show any rotational slip, and the travelled distance is equal to the rotation of the wheel times its radius. Mathematically, these two constraints have the following form

$$\begin{aligned} \dot{x}\sin(\theta_w) - \dot{y}\cos(\theta_w) &= 0, && (43)\\ \dot{x}\cos(\theta_w) + \dot{y}\sin(\theta_w) &= r_w.\dot{\phi}_w && (44) \end{aligned}$$

where x and y are the position coordinates of the wheel, θ_w is the angle of the wheel with the x-axis (steering angle), r_w is the wheel radius, and ϕ_w is the rotation of the wheel around its horizontal axis. Each wheel of the platform is subject to these constraints.

The constraints (43) need to be applied using the specific parameters of the INTERACT platform and its wheel geometry. The important platform parameters are shown in Figure 20. A reference point P with coordinates x_p, y_p defines the origin of a reference frame that is attached to the platform. The distance from P to the front and rear axle is given by a and b respectively. The wheel axle distance is given by $L = a + b$. The wheel geometry is shown in more detail in Figure 21. The distance from P to the vertical (steering) axis A is l. The angle of PA with the x' axis is given by α. The wheel-to-ground contact point is indicated by B and the wheel offset between A and B is w_o, which is a constant. The angle between PA and AB is given by β. If the wheel is orientable, this angle is variable. Additional assumptions that are made to create the system model are:

- The robot frame is a rigid body
- Wheels are non-deformable vertical discs able to roll around their horizontal axis which are always parallel to the ground.
- There is a single point of contact between the ground and the wheel.

To derive the equations of motion for the system pose $\boldsymbol{q}$, the model is simplified by reducing it to the so called bicycle model. The front and rear wheel sets are merged into a virtual wheel located in the middle of the axis (see Figure 20). The resulting system has only two wheels which are both orientable. With this simplified geometry, the nonslip condition for each wheel becomes

$$\dot{x}\cos(\alpha+\beta+\theta_{fr}) + \dot{y}\sin(\alpha+\beta+\theta_{fr}) + l\dot{\theta}_{fr}\sin(\beta) = 0, \tag{45}$$

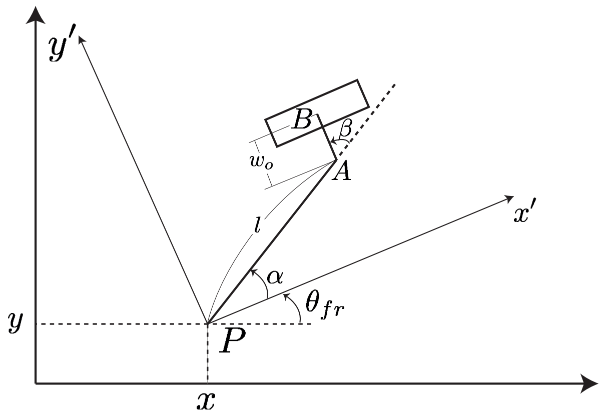


Fig. 21. Sketch of the wheel parameters. For each wheel, l is the distance from P to A, which is the rotation point of the steering axis. β is the angle between the line PA and the wheel its horizontal axis of rotation. β can be considered to be the steering angle if the wheel is orientable. α is the angle between x' and the line PA. The distance between A and the ground contact point B is designated w_o.

which can be rewritten into matrix form

$$\begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & L\sin(\beta) \end{bmatrix} \boldsymbol{R}(\theta_{fr})\dot{\boldsymbol{q}} = 0 \tag{46}$$

where the following trigonometric identity is used ($s = \sin$ and $c = \cos$):

$$s(\alpha+\beta+\theta_{fr}) = s(\alpha+\beta)c(\theta_{fr}) + c(\alpha+\beta)s(\theta_{fr}) \tag{47}$$

$$c(\alpha+\beta+\theta_{fr}) = c(\alpha+\beta)c(\theta_{fr}) - s(\alpha+\beta)s(\theta_{fr}) \tag{48}$$

In these equations, α indicates the angle between the x'-axis and the line PA as shown in Figure 21, with P being a reference point on the chassis. $\boldsymbol{R}(\theta_{fr})$ is the rotation matrix from the robot frame to the world frame:

$$\boldsymbol{R}(\theta_{fr}) = \begin{bmatrix} \cos(\theta_{fr}) & \sin(\theta_{fr}) & 0 \\ -\sin(\theta_{fr}) & \cos(\theta_{fr}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In a similar way, the pure rolling condition can be written as:

$$\dot{x}\sin(\alpha+\beta+\theta_{fr}) - \dot{y}\cos(\alpha+\beta+\theta_{fr}) - w_o\dot{\theta}_{fr} \tag{49}$$

$$-w_o\dot{\beta} - l\dot{\theta}_{fr}\cos(\beta) = r_w\dot{\phi}_w \tag{50}$$

$$\begin{bmatrix} -s(\alpha+\beta) & c(\alpha+\beta) & w_o + lc(\beta) \end{bmatrix} \boldsymbol{R}(\theta_{fr})\dot{\boldsymbol{\xi}} + w_o\dot{\beta} + r\dot{\phi}_w = 0 \tag{51}$$

The choice for β as the steering angle is not very intuitive when applying input to the system. Hence, the steering angle θ_w is defined as the angle between the x'-axis and the wheel plane. The relation between the two is: $\theta_w = \alpha + \beta - \pi/2$. The steering angle θ_w is the second control input to the platform: $\theta_w = u_2$.

The pure rolling condition (51) and nonslip condition (46) can be written down for all four wheels and the two virtual wheels (which receive the subscript f for front and r for rear). The values of the wheel parameters needed for the equations are listed in Table V.

Adopting the notation of [43], all constraints can be put in the following matrix form:

$$\boldsymbol{J_1}(\beta)\boldsymbol{R}(\theta_{fr})\dot{\boldsymbol{\xi}} + \boldsymbol{J_2}\dot{\beta} + \boldsymbol{J_3}\dot{\phi} = 0 \tag{52}$$

$$\boldsymbol{C}(\beta)\boldsymbol{R}(\theta_{fr})\dot{\boldsymbol{\xi}} = 0 \tag{53}$$

Where (52) contains the pure rolling constraints for the four real wheels and (53) contains the no slip conditions for the two virtual wheels. Hence, the matrices are the following:

$$\boldsymbol{J_1} = \begin{bmatrix} -s(\alpha_1+\beta_1) & c(\alpha_1+\beta_1) & w_o + l_1c(\beta_1) \\ -s(\alpha_2+\beta_2) & c(\alpha_2+\beta_2) & w_o + l_2c(\beta_2) \\ -s(\alpha_3+\beta_3) & c(\alpha_3+\beta_3) & -w_o + l_3c(\beta_3) \\ -s(\alpha_4+\beta_4) & c(\alpha_4+\beta_4) & -w_o + l_4c(\beta_4) \end{bmatrix} \tag{54}$$

$$\boldsymbol{J_2} = \operatorname{diag}\begin{bmatrix} r_{w,1} & r_{w,2} & r_{w,3} & r_{w,4} \end{bmatrix} \tag{55}$$

$$\boldsymbol{J_3} = \operatorname{diag}\begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix} \tag{56}$$

$$\boldsymbol{C} = \begin{bmatrix} c(\alpha_f+\beta_f) & s(\alpha_f+\beta_f) & l_fs(\beta_f) \\ c(\alpha_r+\beta_r) & s(\alpha_r+\beta_r) & l_rs(\beta_r) \end{bmatrix} \tag{57}$$

As discussed before, the nonslip constraint is a nonholonomic one that imposes limits on the generalized velocity of the system. For example, the platform is not able to move directly sideways, which is a problem that is often encountered while driving a regular car. The motions that are allowed by the constraints can be found by computing a basis of the null space of $\boldsymbol{C}$. This basis is contained in a matrix designated $\boldsymbol{\Sigma}$:

$$\boldsymbol{\Sigma} = Null\,[\boldsymbol{C}] = \begin{bmatrix} -(a+b)\sin(\beta_f)\sin(\beta_r) \\ \frac{a+b}{2}\sin(\beta_f+\beta_r) \\ -\sin(\beta_f-\beta_r) \end{bmatrix} \tag{58}$$

Which means the generalized velocities for the pose $\dot{\boldsymbol{q}}$ can be computed as:

$$\dot{\boldsymbol{q}} = \boldsymbol{R}(\theta_{fr})\boldsymbol{\Sigma}(\beta)u_1 \tag{59}$$

Here, u_1 represents the first input to the system which is equal to the velocity (in [m/s]) scaled by the platform length L. The rotation matrix is the transform from robot to world frame as encountered before.

The model can be extended to a configuration kinematic model by also including the wheel rotation angles $\boldsymbol{\phi_w}$ in the state. This can be done by rewriting (52) to

$$\dot{\boldsymbol{\phi}}_{\boldsymbol{w}} = -\boldsymbol{J_2}^{-1}\boldsymbol{J_1}\boldsymbol{R}(\theta_{fr})^T\dot{\boldsymbol{\xi}} - \boldsymbol{J_2}^{-1}\boldsymbol{J_3}\dot{\boldsymbol{\beta}}, \tag{60}$$

where $\dot{\boldsymbol{\phi}}_{\boldsymbol{w}}$ and $\dot{\boldsymbol{\beta}}$ are vectors containing the angular speed of all four wheels and the rate of change of the steering angles respectively.

The full state space model with extended state $\boldsymbol{\xi} = \begin{bmatrix} x & y & \theta_{fr} & \phi_1 & \phi_2 & \phi_3 & \phi_4 \end{bmatrix}^T$ becomes:

TABLE V
WHEEL PARAMETERS FOR THE FOUR REAL WHEELS AND THE TWO VIRTUAL WHEELS OF THE BICYCLE MODEL.

Wheels	Subscript	Abbreviation	α	β	l	offset
Imaginary Front	f	IF	0	β_f	a	-
Imaginary Rear	r	IR	π	β_r	b	-
Front Left	1	FL	$\tan^{-1}(w/2a)$	β_1	$\sqrt{(w^2/4+a^2)}$	w_o
Rear Left	2	RL	$\pi-\tan^{-1}(w/2b)$	β_2	$\sqrt{(w^2/4+b^2)}$	w_o
Rear right	3	RR	$\pi+\tan^{-1}(w/2b)$	β_3	$\sqrt{(w^2/4+b^2)}$	$-w_o$
Front right	4	FR	$-\tan^{-1}(w/2a)$	β_4	$\sqrt{(w^2/4+a^2)}$	$-w_o$

$$\dot{\boldsymbol{\xi}} = \boldsymbol{S}(\boldsymbol{q})u_1 + \boldsymbol{\Gamma} \tag{61}$$

$$\boldsymbol{S}(\boldsymbol{\xi}) = \begin{bmatrix} \boldsymbol{R}(\theta)\boldsymbol{\Sigma}(\beta) & 0 \\ -\boldsymbol{J_2}^{-1}\boldsymbol{J_1}\boldsymbol{R}(\theta)\boldsymbol{\Sigma}(\beta) & 0 \end{bmatrix} \tag{62}$$

$$\boldsymbol{\Gamma} = \begin{bmatrix} 0 \\ 0 \\ -\boldsymbol{J_2}^{-1}\boldsymbol{J_3}\dot{\boldsymbol{\beta}} \end{bmatrix} \tag{63}$$

B. Rotate In Place

The configuration kinematic model described above is able to represent the four wheel steer and crab mode of the platform. However, it contains a singularity for $\theta_f = \pi/2$ and $\theta_r = -\pi/2$ or, equivalently, $\beta_f = 0$ and $\beta_r = 0$. In a regular vehicle, these angles would not be actually possible which removes the need to consider the singularity. However, the INTERACT platform is capable of rotation in place, which requires the steering angles to be precisely this. Hence, a separate configuration kinematic model needs to be derived for this drive mode.

The model for rotation in place is much simpler, as the velocity input directly determines the angular velocity. The x and y coordinates of the point P are stationary or describe a circular motion if P is offset from the middle. It can be easily derived that the generalized velocities can be computed as follows:

$$\dot{\theta}_{fr} = w_o + \sqrt{\frac{L^2}{4} + \frac{w_A^2}{4}}u_1 \tag{64}$$

$$\dot{x} = -\left(\frac{L}{2} - a\right)\sin(\theta_{fr})\dot{\theta}_{fr} \tag{65}$$

$$\dot{y} = \left(\frac{L}{2} - a\right)\cos(\theta_{fr})\dot{\theta}_{fr} \tag{66}$$

And the wheel rotational speeds:

$$\dot{\phi}_{w,1} = \frac{u_1}{r_{w,1}} \tag{67}$$

$$\dot{\phi}_{w,2} = \frac{u_1}{r_{w,2}} \tag{68}$$

$$\dot{\phi}_{w,3} = \frac{u_1}{r_{w,3}} \tag{69}$$

$$\dot{\phi}_{w,4} = \frac{u_1}{r_{w,4}} \tag{70}$$

C. Ackermann Steering and Side Slip Angle

With a four wheeled vehicle, not all steering angles are fully independent. As soon as the angles of the virtual wheels are set, a so called instantaneous center of rotation (ICR) is defined by the intersection of the rotation axes of the two wheels. In many vehicular applications, the real wheels get the same steering angle as the virtual wheels, so called parallel steering, i.e. $\theta_f = \theta_{w,1} = \theta_{w,4}$ and $\theta_r = \theta_{w,2} = \theta_{w,3}$. However, this introduces wheel slip, as the rotation axes of the real wheels will not intersect at the ICR. A different steering geometry was introduced by Ackermann [44] which lets the axes of rotation intersect at the same point and hence eliminates slip. This means that the steering angle of the inner wheels are slightly larger than the angles of the outer wheels, see also Figure 22.

The steering angles of all four wheels can be computed by using simple trigonometry and the wheel geometry shown in Figure 22,

$$\begin{aligned} \tan(\theta_f) &= \tfrac{C_{front}}{R_1}, & \tan(\theta_r) &= \tfrac{C_{rear}}{R_1} \\ \tan(\theta_1) &= \tfrac{C_{front}}{R_1 - \frac{w_A}{2}}, & \tan(\theta_2) &= \tfrac{C_{rear}}{R_1 - \frac{w_A}{2}} \\ \tan(\theta_3) &= \tfrac{C_{rear}}{R_1 + \frac{w_A}{2}}, & \tan(\theta_4) &= \tfrac{C_{front}}{R_1 + \frac{w_A}{2}} \end{aligned} \tag{71}$$

Furthermore

$$L = a + b = C_f + C_r \tag{72}$$

In these equations, R_1 is the shortest distance from the ICR to the longitudinal axis of the platform, R is the distance from the ICR to P and is the radius of rotation for this reference point. This system can be solved for the real angles β_1 to β_4 (and thus for the steering angles θ_{w1-4}) and for the distances C_{rear}, C_{front} and R_1.

The so called side slip angle can also be computed from the geometry shown in Figure 22. The side slip is defined as the angle between the velocity vector at P, which is always perpendicular to the radius of rotation R, and the current heading (longitudinal axis) of the platform itself. The side slip angle is computed as

$$\tan(\psi) = \left(\frac{C_{rear} + b}{R_1}\right), \tag{73}$$

Using (73) and the expressions for θ_f and θ_r from (71) one can derive:

$$R_1 = \frac{C_{rear}}{\tan(\theta_r)} \tag{74}$$

$$\frac{\tan(\theta_f)}{\tan(\theta_r)} = \frac{a + b + C_{rear}}{C_{rear}} \tag{75}$$

resulting in:

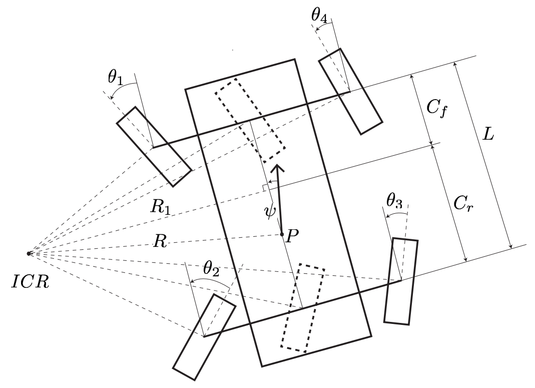


Fig. 22. Geometry of four wheel steer (Ackerman Steering). The horizontal axis of rotation of each wheel intersect in the same point, which is called the instantaneous center of rotation (ICR). The radius of rotation is indicated with R, which is the distance from the ICR to point P. When parallel steering is performed, there is no single instantaneous center of rotation, and hence there will be slip occurring at the wheels. The velocity vector at P, perpendicular to R, defines the side slip angle ψ as the angle between this velocity vector and the current heading of the platform.

$$C_{rear} = \frac{(a+b)\tan(\theta_r)}{\tan(\theta_f) - \tan(\theta_r)} \tag{76}$$

Substituting this in (73) gives:

$$\tan(\psi) = \frac{b\tan(\theta_f) + a\tan(\theta_r)}{a+b} \tag{77}$$

The radius of rotation then becomes:

$$ICR_{dist} = \sqrt{R_1^2 + (C_{front} - a)^2} \tag{78}$$

D. Implementation in Simulink

The mathematical model as described above is implemented in Matlab/Simulink. The inputs to the model are the following:

- For Ackermann Steering: (1) Velocity input which is the velocity of point P in the x' direction scaled by the platform length: $v_{input} = v_{x'}/L$. This means that if the desired velocity is 1 m/s, this should be divided by L_p to give the velocity input to the sim. (2) Front virtual wheel steering angle input in degrees. (3) Rear virtual wheel steering angle in degrees.
- For parallel steering: (1) Velocity input identical to the Ackermann steering case (2) Steering angle input for the front and rear virtual wheels (equal but opposite sign).
- For rotation in place: (1) Rotational velocity of the wheel scaled by platform length.

The output of the model is equal to (61): the posture including orientation and wheel rotation parameters in the world frame.

Appendix III
Model Validation

To assess the accuracy of kinematic model and the simulation, the output has to be compared with the output of the real platform. To be able to compare the two, the pose of the real platform has to be measured very accurately. This can be done by using the Vicon motion capture system available in the lab. This system uses cameras with IR emitters and reflective markers to estimate the 3D position and orientation of objects. The accuracy of this system is in the order of millimetres. The Vicon data is taken as a ground truth in all validation and calibration experiments.

Several experiments were performed where the platform was driven outside on tarmac/asphalt. The actual platform location was measured with 4 Vicon cameras (or 6 for later experiments). The input (velocity, steering angle) to the platform was logged along with several other important parameters such as:

- Motor velocity for each wheel
- Motor position for each wheel
- Steering encoder count for each wheel
- Acceleration, velocity and position according to accelerometers in the on-board IMU
- Attitude and angular velocities according to the gyroscopes in the on-board IMU
- Drive mode

The recorded input can be supplied to the Simulink model and the actual position as measured by the Vicon system can be compared to the model output. Discrepancies between the two can have several sources which can be categorized into systematic and non-systematic errors [37]. The systematic errors include unequal wheel diameters, wheel diameters that differ from the nominal/documented diameter, inaccuracies in the wheel base and length of the platform, and wheel misalignments. Many of these errors can be reduced or eliminated by performing platform calibration, which aims at empirically establishing the important platform parameters to better match reality. Details of this calibration method are discussed in subsection V-C of this Appendix.

A. *Error Metric for Performance Assessment*

To be able to quantify the performance of the model, some metric is needed that indicates how well the model represents reality. A metric often used in literature (see e.g. [45], [46], [11]) is the difference between the estimated/-modelled final position of the platform and the actual final position, represented as a percentage of the total travelled distance. Mathematically, this can be computed as:

$$e_d = \frac{\sqrt{(x_{f,m} - x_{f,v})^2 + (y_{f,m} - y_{f,v})^2}}{d_{\text{total}}} \cdot 100\% \qquad (79)$$

This metric is reported in the subsequent sections to assess the performance of the model (and later of the localization accuracy) as compared to the ground truth as measured by the Vicon system.

TABLE VI
Resulting errors between the final position of the platform according to the model and according to the ground truth.

	Errors	
Experiment	**Initial**	**Calibrated and velocity adj.**
Straight driving	1.50%	1.38%
Turn-in-place	9.47%	6.44%
$10°$ turn	20.15%	4.35%
$15°$ turn	10.49%	2.34%
$20°$ turn	2.52%	1.29%
1st run, $-10°$ turn	19.96%	0.60%
2nd run, $-10°$ turn	11.43%	5.02%
3rd run, $-10°$ turn	24.23%	7.96%
$-20°$ turn	15.30%	1.14%
$-25°$ turn	9.19%	0.86%

B. *Results*

The following results were obtained by running the actual platform and the simulation side by side while applying the same input to both. Besides the platform input, the simulation requires several relevant platform parameters that influence the model output. The parameters were measured on the platform or taken from documentation from the manufacturer. The values were as follows:

- Distance between front and rear axle: $L = 0.805$ meter
- Distance between left and right steering axle: $w_A = 0.787$ meter
- Distance between steering axle and ground contact point: $w_o = 0.071$ meter
- Wheel radius: $r_w = 0.241$ meter

Several maneuvers were performed which included straight driving and turns to the left and the right with steering angles ranging from 10 degrees up to 25 degrees. A turn-in-place maneuver was also performed. In Figures 23, 24 and 25 plots of the straight driving, turn-in-place and turning with an angle of -20 degrees are given. These three maneuvers were chosen because the results clearly show the discrepancies between model and reality. The plots contain the following information, starting with the top left graph in Figure 23: the x-position versus the y-position of both the simulation (orange) and Vicon measurement (blue). The final positions are indicated with a red square for the simulation and a blue cross for the Vicon data. The top right graph shows the platform heading error (blue) defined as the difference between the simulated heading (θ_{fr}) and the Vicon measurement. Bottom left shows the normalized absolute velocity of the platform ($\sqrt{\dot{x}^2 + \dot{y}^2}$) versus time along with the velocity from the Vicon. Finally, the bottom right figure shows the x-position and y-position errors of the platform in blue and orange respectively.

The error metric as defined in equation (79) was computed for every maneuver separately. These errors are listed in Table VI.

From these results it is evident that the actual platform velocity is slightly lower than the velocity used in the simulation. This can be seen especially well in the norm velocity graph of Figure 25. It also seems that the velocity difference is steering angle dependent, with increasing differences for higher steering angles. The cause of this discrepancy seems to lie with the platform itself, i.e. it cannot match the commanded input velocity for some reason. The velocity

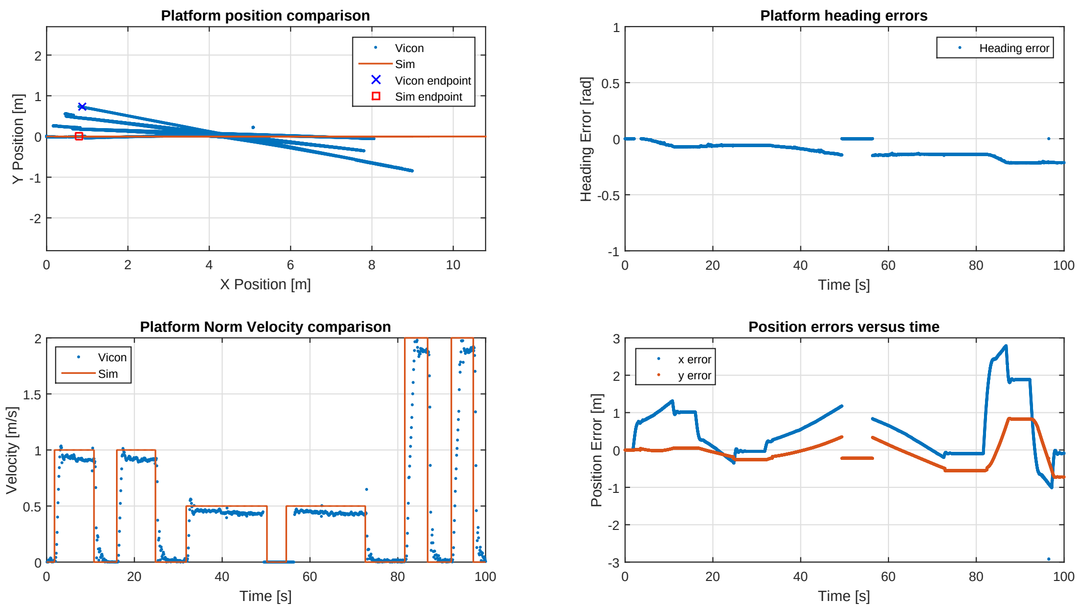


Fig. 23. Driving straight, both backwards and forwards. No calibration of platform parameters nor velocity.

input of the simulation is adjusted to match the actual platform. The values that have to best correspondence with the ground truth are: $0.92v_{input} - 0.3|\theta_f|$ for normal motion and $0.28v_{input}$ for rotations in place.

A large error between simulation and experiment can be seen in the results for the rotate in place maneuver in Figure 24. This error has two causes. The first is again related to the velocity, with the simulated platform rotating much faster than the actual platform. This can be seen from the platform heading error. The second cause is the fact that the simulation has a perfect reference point P located exactly in the middle of the platform. While rotating in place, this point does not move. Evidently, the reference point on the platform is not stationary, and there is an offset of P to the rear of the platform. Adjusting the parameters a and b accordingly in the simulation greatly improves the results as can be seen in Figure 27.

Another discrepancy is that the radius of rotation of the simulated platform is much smaller than that of the actual platform. This phenomenon can have several causes, for example inaccuracies in wheel radius, wheel base or wheel axle distance. To be able to calibrate the platform and discern between the different errors, a good calibration method is required. This method is described in more detail in Appendix VI. The distance between the front and rear axle is corrected to match the experimental results which requires the value: $L_p = 0.9232$ meter. The errors are computed again after adjusting the velocity and calibrating the platform parameters. The results are listed in the second column of Table VI and shown in Figure 26 to 28.

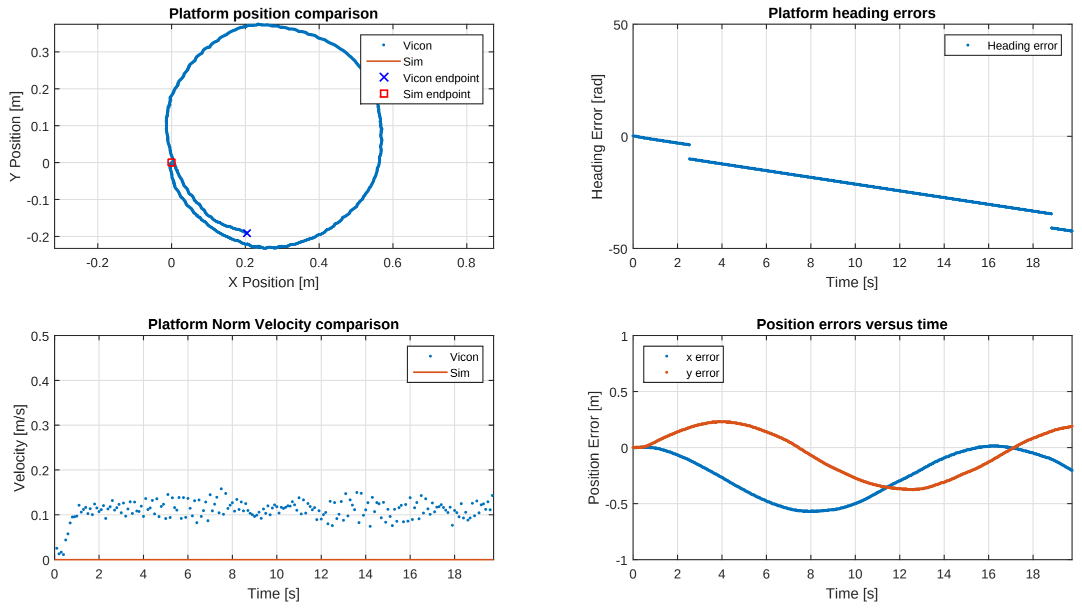


Fig. 24. Turn-in-place. No calibration of platform parameters nor velocity.

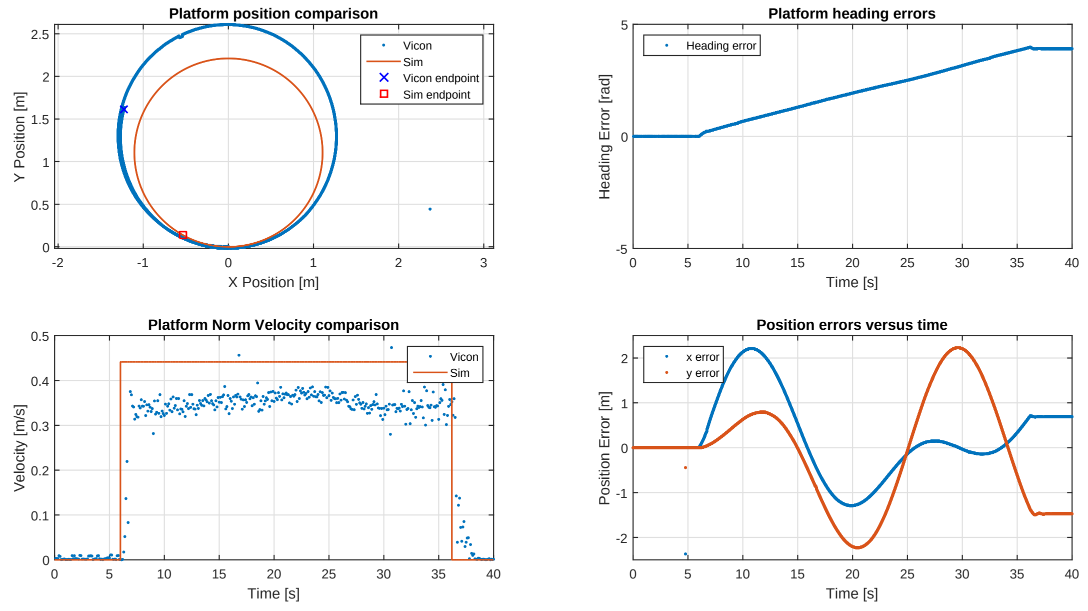


Fig. 25. -20° degree turn. No calibration of platform parameters nor velocity.

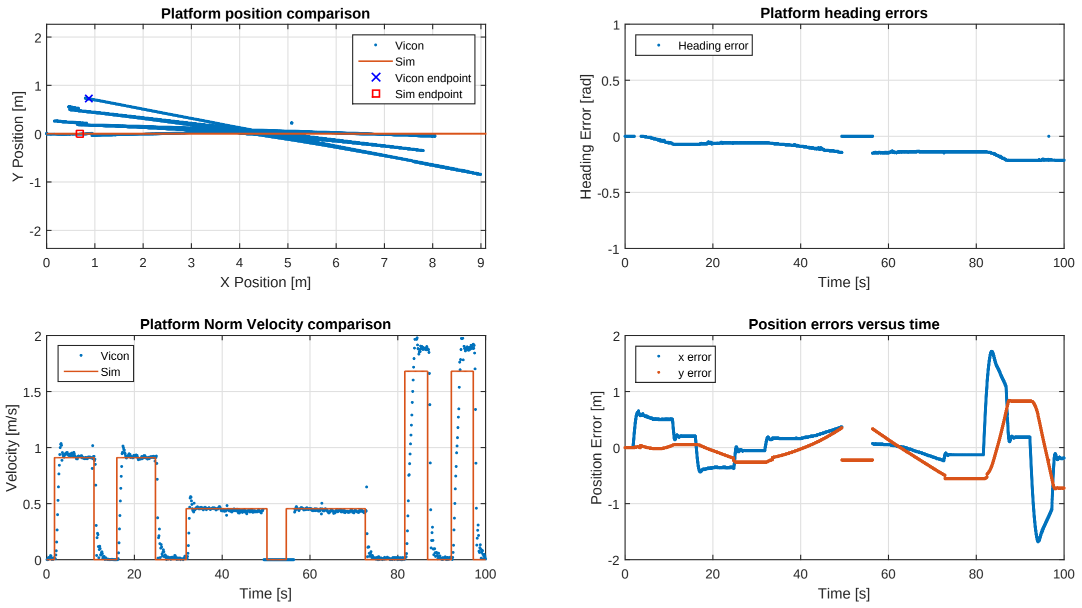


Fig. 26. Driving straight, both backwards and forwards. Parameter calibration and velocity correction.

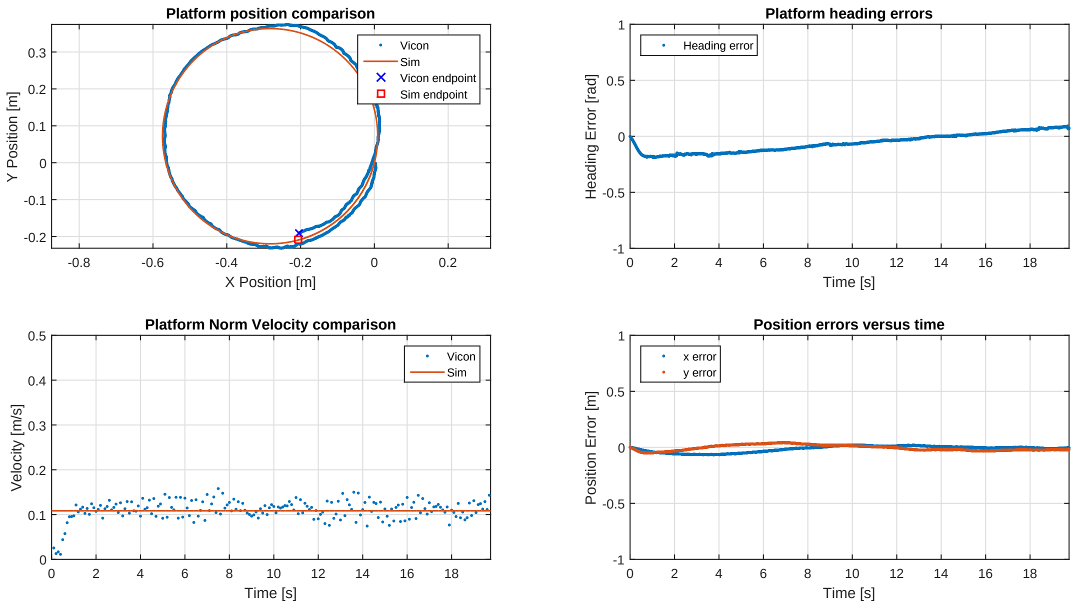


Fig. 27. Turn-in-place. Parameter calibration and velocity correction.

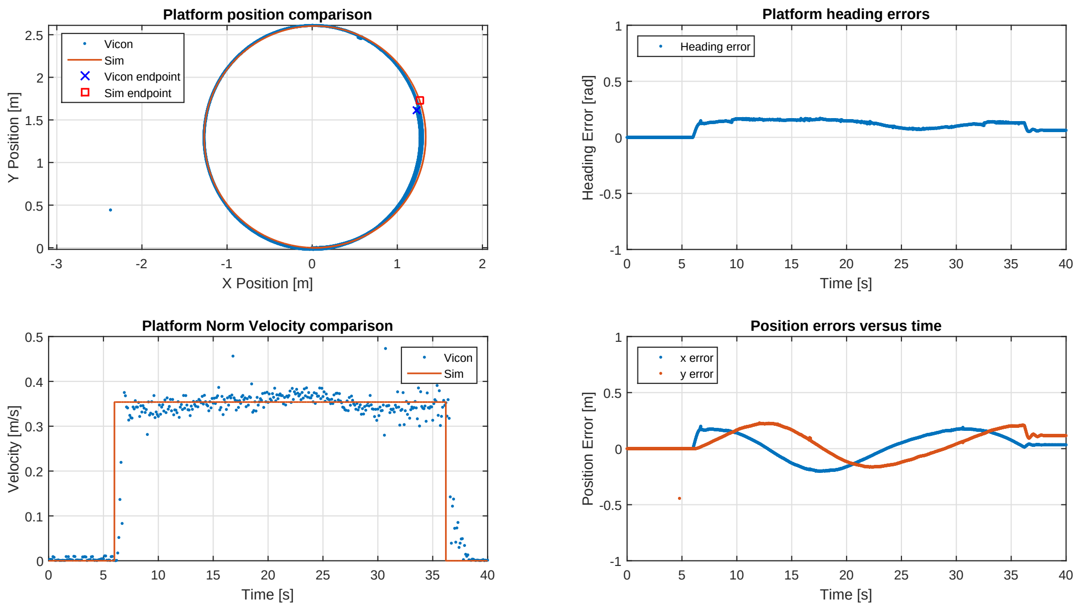


Fig. 28. $-20°$ degree turn. Parameter calibration and velocity correction.

Appendix IV
Model Simulation and Visualization

To be able to visualize the model described in Appendix II, the SPANviewer software is used, which is developed at the Haptics and Telerobotics Lab. The software is able to visualize and animate robotic systems based on CAD model files (.vrml) and input supplied by the user, such as the Denavit Hartenberg (DH) parameters of the system. Visualization can help to judge and verify the behaviour of the model and simulation. Unnatural behaviour of the platform that cannot always be seen from graphs and plots can be quickly identified in this way.

A. Simulation in SPANviewer

The .xml code needed to visualize the INTERACT system can be found on the server of the Haptics and Telerobotics lab in the miniproject *"AutoNavAmbot"* and in section IV-E of this Appendix. This file defines the exact locations of the separate components of the platform, which .vrml files to use, geometrical parameters of the system, and which parameters are variables that need to be supplied by the user.

To be able to smoothly simulate the system, the CAD model needs to be slightly simplified to reduce the number of components. There is no detail necessary in the platform chassis itself, as well as in the casing of the robot. Furthermore, internals are not visible and can be left out. Hence, the first component in SPANviewer is the "shell" of the platform including the casing.

To be able to show the motion of the wheels (both steering and rotation) they need to be separated from the body. The wheel hubs are attached to the platform shell at the location indicated in the documentation from the manufacturer and can, in this way, visualize the steering motion of the wheels. The wheels itself are attached to the hubs and are able to rotate around the horizontal axis.

The two Kuka LWR arms for manipulation as well as the Schunk arm that holds the head are then added to the model. Because these arms are attached to a custom made part of which no accurate technical drawing is available, the position of the arm base relative to the platform needs to be measured precisely on the actual platform. Details of this measurement can be found in subsection IV-C. The resulting model and visualization in SPANviewer can be seen in Figure 29

To be able to animate the model in SPANviewer, it needs to receive the extended configuration state $\boldsymbol{q}$ from the Matlab/Simulink model. Communication between Matlab and SPANviewer goes through the RTI Data Distribution Service (DDS). Data from the Simulink model is published on certain topics. SPANviewer can listen on these topics and take the data as input.

B. ICR Visualization and Predicted Position Overlay

The Instantaneous Center of Rotation (ICR) and the radius of rotation are also visualized in SPANviewer. The radius of rotation is represented by a transparent red disc centered on the ICR as illustrated in Figure 30.

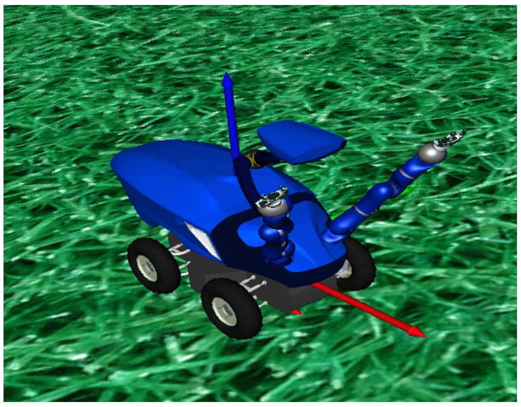


Fig. 29. Print screen of the SPANviewer GUI showing the INTERACT model and the scene around it. The arrows show the position and orientation of the world frame.

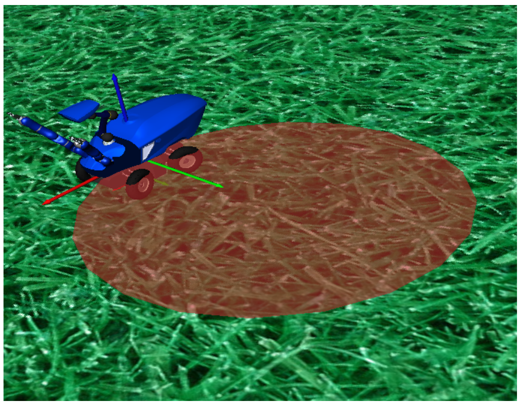


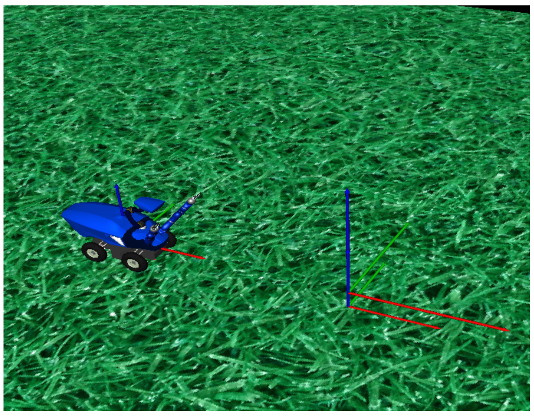
Fig. 30. Print screen showing the visualization of the ICR and the radius of rotation. The input steering angle is 0.2 radians for the front wheels and -0.2 radians for the rear wheels.

Furthermore, the means for the astronaut to control the position of the robot are simulated and visualized as well. The astronaut is able to incrementally increase the position set point of the platform. This set point can be visualized in two ways, namely relative to the world or relative to the platform. In the first case, the set point is fixed w.r.t. the world and hence indicates the goal position at the start of the motion. However, the final position of the platform might not coincide with this location due to wheel misalignments and other non-ideal mechanical issues and the lack of feedback control on the position. The second case updates the goal position during execution based on the current platform position and travelled distance. In this sense, the second visualization method better predicts the final position of the platform. Figure 31 shows print screens of both overlays.

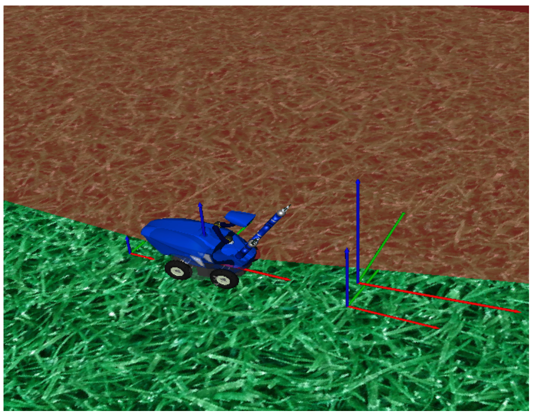
C. Arm Base Transform Measurement

This section will give details of the measurement performed with the VICON motion capture system to find the transformation between the AMBOT platform base frame and the mounting base frame of the three robotic arms (2x Kuka LWR and 1x Schunk).

Base frame of the AMBOT platform: Because the middle of the platform was not accessible due to the mounted components, another location for the base frame



(a) Start



(b) Execution

Fig. 31. Print screens showing both versions of the predicted goal position overlay. At the start (left image) both predictions overlap. The overlay relative to the robot is located slightly higher and with longer axes to be able to better see the difference. In the right image, a wheel misalignment is simulated by giving a very small steering angle. The robot relative prediction hence moves slightly during execution. The overlay that is fixed in the world frame is stationary.

had to be chosen. A location that is accessible at any time is the front (nose) of the platform. This place was chosen to define a temporary base frame (x pointing down, y pointing to the left, z pointing forward). See also Figure 32c.

Base frame of arm mounting: The frame on the mounting points for the arms needs to be in the middle of the aluminium cylinder. To find the middle, thread was put through the mounting holes and crossed over to the hole on the opposing side. The wand was then aligned with the middle of the cylinder, with x aligned with a set of holes pointing forward, y pointing to the middle of the platform and z pointing up. See also Figure 32.

D. Results

Measuring the rotation matrix and translation using the Vicon system resulted in the following transformation matrices:

Left Kuka arm:

$$T = \begin{bmatrix} -0.5319 & 0.0300 & -0.8463 & -210.5021 \\ -0.3127 & -0.9357 & 0.1634 & 291.3380 \\ -0.7870 & 0.3515 & 0.5071 & -134.1501 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Right Kuka arm:

$$T = \begin{bmatrix} 0.5295 & -0.0284 & 0.8478 & -212.0201 \\ -0.3078 & 0.9249 & 0.2233 & 54.6301 \\ -0.7905 & -0.3792 & 0.4810 & -137.8906 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Schunk arm:

$$T = \begin{bmatrix} -0.0141 & -0.0206 & 0.9997 & -405.2506 \\ -0.0051 & 0.9998 & 0.0205 & 173.5694 \\ -0.9999 & -0.0048 & -0.0142 & -281.7267 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

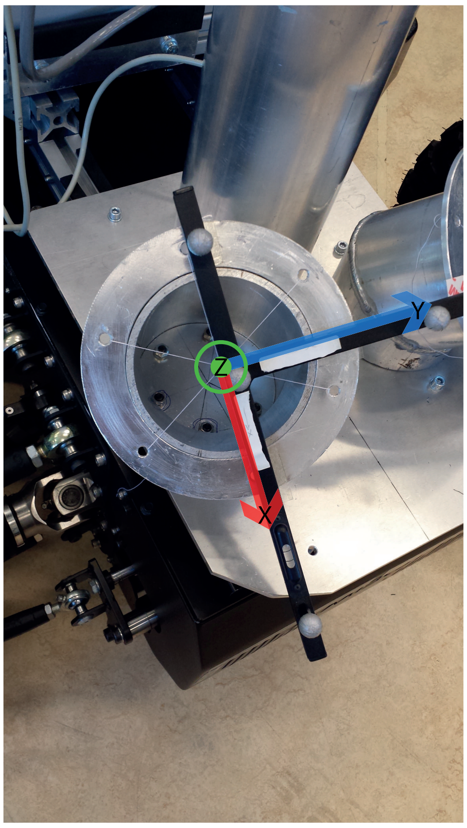
These values can also be found in the .mat files `LeftKukaBaseTransform.mat`, `RightKukaBaseTransform.mat` and `SchunkBaseTransform.mat`. Conversion to XYZ Euler angles, which is more convenient for SPANviewer, gives:

$$\begin{aligned} \text{Left Kuka: } & \alpha = 5.6770, \quad \beta = 5.3773, \quad \gamma = 2.6102 \\ \text{Right Kuka: } & \alpha = 0.6676, \quad \beta = 5.3716, \quad \gamma = 0.5266 \\ \text{Schunk: } & \alpha = 2.8134, \quad \beta = 4.7274, \quad \gamma = 2.7929 \end{aligned}$$

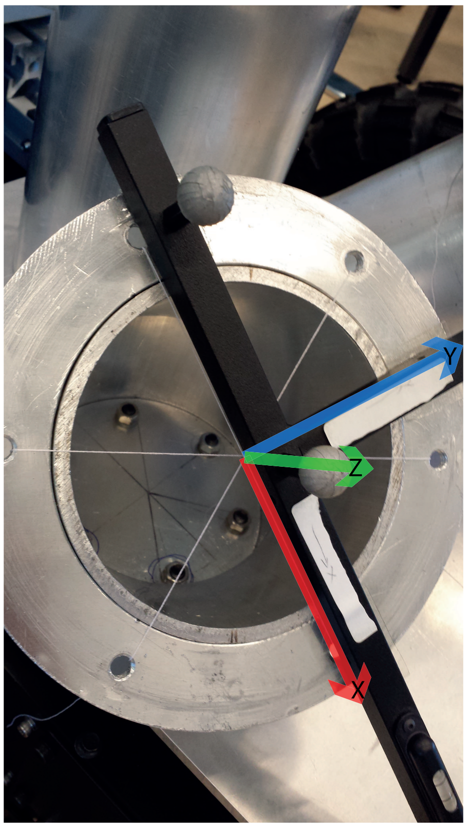
The final transformation of the platform base (middle of the platform's top plate) to the arm base is then a combination of the transform from the base to the frame established at the nose and the transform from the nose frame to the arm base as given above. The transform from the base frame to the nose frame was measured from the available CAD drawings and verified on the real platform:

Nose Frame:

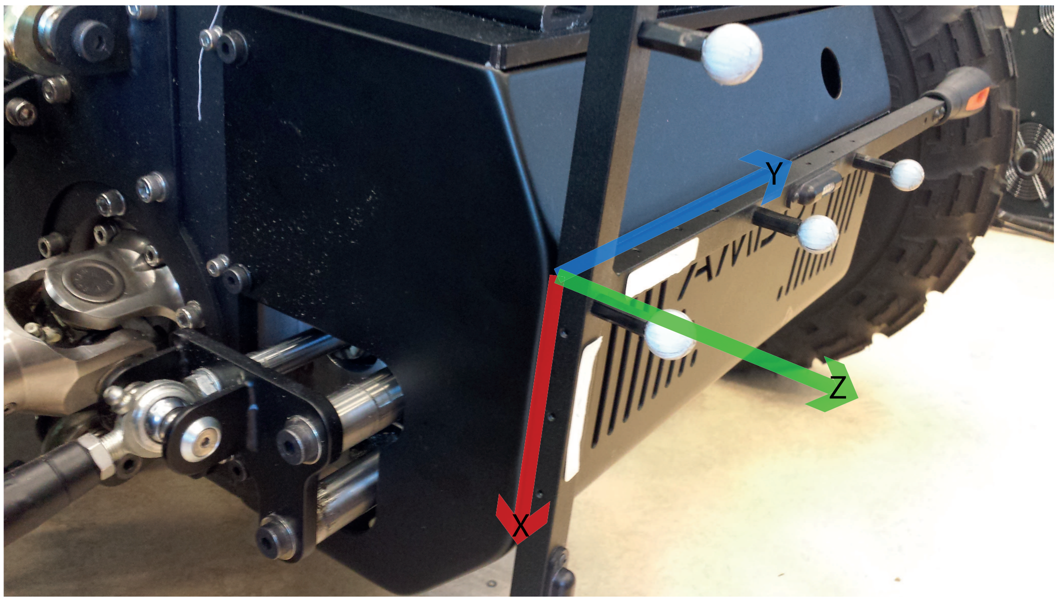
$$T = \begin{bmatrix} 0 & 0 & 1 & 619.04 \\ 0 & 1 & 0 & -173.23 \\ -1 & 0 & 0 & -52.98 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(a) Right Kuka arm



(b) Detail right Kuka arm



(c) AMBOT platform

Fig. 32. Images clarifying the definition of the frames on the AMBOT platform.

E. INTERACT .xml file for SPANviewer

```
 ,
<?xml version="1.0"?>
<span xmlns="ESA-Telerobotics-Lab-SPAN" version="0.3">
  <world>
    <scene axis="0.5">
      <image s="100" filename="asphalt.jpg" />
      <structure convention="dh" >
        <offset fixed="0" axis_scale="0" channel="1">
          <joint axis_scale="0" fixed="0">
            <vrml r="pi/2,pi/2,0" s="0.001" tx="0" ty="0" tz="0.0" name="INTERACT_spanviewer.wrl" />

            <!-- left front wheel -->
            <offset r="0" tx="0.40237" ty="0.3646625" tz="-0.124471">
              <joint a="0" alpha="pi/2" axis_scale="0.1" d="0" theta="0" fixed="0">
                <vrml r="-pi/2,0,pi" s="0.001" tx="0" ty="0" tz="0.0" name="wheelhub.wrl" />
                <joint a="0.0" alpha="pi/2" axis_scale="0" d="0" theta="0.0" fixed="0">
                  <vrml r="0,0,0" axis_scale="0" s="0.001" tx="0" ty="0" tz="0.0" name="wheel.wrl" />
                </joint>
              </joint>
            </offset>

            <!-- right front wheel -->
            <offset r="0" tx="0.40237" ty="-0.3646625" tz="-0.124471">
              <joint a="0" alpha="pi/2" axis_scale="0.1" d="0" theta="0" fixed="0">
                <vrml r="pi/2,0,pi" s="0.001" tx="0" ty="0" tz="0.0" name="wheelhub.wrl" />
                <joint a="0.0" alpha="pi/2" axis_scale="0" d="0" theta="0.0" fixed="0">
                  <vrml r="0,0,pi" axis_scale="0" s="0.001" tx="0" ty="0" tz="0.0" name="wheel.wrl" />
                </joint>
              </joint>
            </offset>

            <!-- left rear wheel -->
            <offset r="0" tx="-0.40237" ty="0.3646625" tz="-0.124471">
              <joint a="0" alpha="pi/2" axis_scale="0.1" d="0" theta="0" fixed="0">
                <vrml r="-pi/2,0,0" s="0.001" tx="0" ty="0" tz="0.0" name="wheelhub.wrl" />
                <joint a="0.0" alpha="pi/2" axis_scale="0" d="0" theta="0.0" fixed="0">
                  <vrml r="0,0,0" axis_scale="0" s="0.001" tx="0" ty="0" tz="0.0" name="wheel.wrl" />
                </joint>
              </joint>
            </offset>

            <!-- right rear wheel -->
            <offset r="0" tx="-0.40237" ty="-0.3646625" tz="-0.124471">
              <joint a="0" alpha="pi/2" axis_scale="0.1" d="0" theta="0" fixed="0">
                <vrml r="pi/2,0,pi" s="0.001" tx="0" ty="0" tz="0.0" name="wheelhub.wrl" />
                <joint a="0.0" alpha="pi/2" axis_scale="0" d="0" theta="0.0" fixed="0">
                  <vrml r="0,0,pi" axis_scale="0" s="0.001" tx="0" ty="0" tz="0.0" name="wheel.wrl" />
                </joint>
              </joint>
            </offset>

            <!-- Cylinder to show ICR and radius of rotation -->
            <offset r="0" t="0" axis_scale="0" fixed="0" channel="2">
              <cylinder c="255,50,50,150" fixed="0" />
            </offset>

            <!-- right kuka arm -->
            <offset r="0,pi/2,0" tx="0.61904" ty="-0.17323" tz="-0.05298"
axis_scale="0" channel="3">
              <offset r="0.6676,5.3716,0.5266" t="-0.2120201,0.0546301,-0.1378906" axis_scale="0">
                <vrml s="1" name="kuka-swivel.wrl" axis_scale="0"/>
                <joint alpha="0" d="0.310" label="2" axis_scale="0">
                  <vrml s="1" r1="0" name="kuka-link1.wrl"/>
                  <joint alpha="0" label="3" axis_scale="0">
                    <vrml s="1" r1="pi/2" name="kuka-link2.wrl"/>
                    <joint alpha="-pi/2" d="0.400" label="3" axis_scale="0">
                      <vrml s="1" r1="pi/2" name="kuka-link3.wrl"/>
                      <joint alpha="pi/2" label="4" axis_scale="0">
                        <vrml s="1" r1="-pi/2" name="kuka-link4.wrl"/>
            <joint alpha="pi/2" d="0.390" label="5" axis_scale="0">
             <vrml s="1" r1="-pi/2" name="kuka-link5.wrl"/>
             <joint alpha="-pi/2" label="6" axis_scale="0">
              <vrml s="1" r1="pi/2" name="kuka-link6.wrl"/>
              <joint d="0.078" label="7" axis_scale="0">
               <vrml s="1" tz="-0.078" name="kuka-link7.wrl"/>
               <offset tz="0.010" axis_scale="0">
                <vrml s="0.001" r="-pi/2,0,0" name="gripper.wrl"/>
               </offset>
              </joint>
             </joint>
            </joint>
           </joint>
          </joint>
         </joint>
        </joint>
       </offset>
      </offset>

      <!-- left kuka arm -->
      <offset r="0,pi/2,0" tx="0.61904" ty="-0.17323" tz="-0.05298"
axis_scale="0" channel="3">
       <offset r="5.6770,5.3773,5.7518" t="-0.2105021,0.2913380,-0.1341501" axis_scale="0">
        <vrml s="1" name="kuka-swivel.wrl" axis_scale="0"/>
        <joint alpha="0" d="0.310" theta="0" label="2" axis_scale="0">
         <vrml s="1" r1="0" name="kuka-link1.wrl"/>
         <joint alpha="0" theta="0" label="3" axis_scale="0">
          <vrml s="1" r1="pi/2" name="kuka-link2.wrl"/>
          <joint alpha="-pi/2" d="0.400" theta="0" label="3" axis_scale="0">
           <vrml s="1" r1="pi/2" name="kuka-link3.wrl"/>
           <joint alpha="pi/2" label="4" axis_scale="0">
            <vrml s="1" r1="-pi/2" name="kuka-link4.wrl"/>
            <joint alpha="pi/2" d="0.390" label="5" axis_scale="0">
             <vrml s="1" r1="-pi/2" name="kuka-link5.wrl"/>
             <joint alpha="-pi/2" label="6" axis_scale="0">
              <vrml s="1" r1="pi/2" name="kuka-link6.wrl"/>
              <joint d="0.078" label="7" axis_scale="0">
               <vrml s="1" tz="-0.078" name="kuka-link7.wrl"/>
               <offset tz="0.010">
                <vrml s="0.001" r1="-pi/2" name="gripper.wrl"/>
               </offset>
              </joint>
             </joint>
            </joint>
           </joint>
          </joint>
         </joint>
        </joint>
       </offset>
      </offset>

      <!-- Schunk arm -->
      <offset r="0,pi/2,0" tx="0.61904" ty="-0.17323" tz="-0.05298"
axis_scale="0" channel="3">
       <offset r="2.8134,4.7274,2.7929" t="-0.4052506,0.1735694,-0.2817267" axis_scale="0">
        <offset tz="0.205">
         <vrml s="1" tz="-0.060" tx="-0.090" r1="pi/2" r2="-pi/2" name="fuss_LWA_4P.wrl" />
         <joint alpha="pi/2" d="0" a="0" theta="0" label="1" axis_scale="0">
          <vrml s="1" tz="0" ty="0" r1="pi/2" r3="0" name="ERB_145.wrl" />
          <joint alpha="-pi" d="0" a="0.350" theta="2" label="2" axis_scale="0">
           <vrml s="1" tx="-0.175" ty="0" tz="-0.100" r1="pi" name="vbe_145_145.wrl" />
           <joint alpha="-pi/2" d="0" a="0" theta="0" label="3" axis_scale="0">
            <vrml s="1" tx="0" tz="0" ty="0" r1="0" r2="0" name="ERB_145.wrl" />
            <joint alpha="pi/2" d="0.305" a="0" theta="0" label="4" axis_scale="0">
             <vrml s="1" tx="0" ty="-0.205" tz="0" r1="0" r2="pi"
r3="pi/2" name="vbe_145_115.wrl" />
             <joint alpha="-pi/2" d="0" a="0" theta="-1" label="5" axis_scale="0">
              <vrml s="1" tz="0" ty="0" r1="-0.7854" r2="0" r3="pi" name="ERB_115.wrl" />
              <joint alpha="0" d="0.075" a="0" label="6" axis_scale="0">
               <vrml s="0.001" t="-0.246,0,0.008" r="pi/2,0,0" axis_scale="0" name="head.wr
              </joint>
```

```
                          </joint>
                        </joint>
                      </joint>
                    </joint>
                  </joint>
                </offset>
              </offset>
            </offset>
          </joint>
        </offset>

      </structure>
    </scene>

  </world>
</span>
```

Appendix V
Localization

Solving the problem of estimating the location of a robotic vehicle in a certain reference frame is very important. If the position and attitude of a system is not well know, it is nearly impossible to correctly perform (for example) manipulation tasks, path planning or object avoidance [47]. This is because these objects are always defined in the *world frame* and their position becomes uncertain in a robot centered frame when the location of the vehicle is not exactly known in this world frame. Without an accurate location estimate, the first phase of the navigational problem (path planning and tracking), as shown in Figure 1b and 1c in the paper, would be impossible to execute. The method of estimating the location of a system is called *localization* or *state estimation* and has been a topic gaining much interest in the robotic community because of its importance for task execution [12].

The variety and number of sensors available on the INTERACT platform is very limited, and the use of external sensors (e.g. GPS, radio or IR beacons or cameras) is not possible in the environments the platform will be used in. Using the on-board sensors such as wheel encoders for localization purposes is referred to as dead-reckoning. If an inertial measurement unit (IMU) is used as well, the procedure is called odometry. The IMU on the platform is a MicroStrain 3DM-GX3-15 attitude-heading reference system (AHRS). The following sensor data is available on the platform:

- Steering angle encoder data
- Wheel rotation encoder data
- Three axis gyroscope data (IMU)
- Three axis accelerometer data (IMU)

In the next section, the dead-reckoning update equations are derived. This is the basis for the localization estimate. The dead-reckoning can be extended with IMU measurements through sensor fusion. This can be done using a Kalman filter [48], [11] or Fuzzy Logic [21]. Results of the odometry estimate are presented as well.

A. Dead-Reckoning

Dead-reckoning refers to the use of wheel encoders (measuring the position/angle of the driving axle of the wheels) to estimate the position of a vehicle. It is a widely used and simple method (e.g. [37], [20], [49], [18]). The minimal working configuration to perform this measurement is with encoders on two kinematically unconnected wheels on both sides of the vehicle [12]. Computing a position and heading estimate from these readings can be done as follows. The rotation of the wheel can be used to compute an estimate of the distance travelled by using the wheel radius. A difference between the readings on the left and on the right suggests a change in heading $\Delta\theta_{fr}$ [37], [50].

The localization method used for INTERACT relies on the wheel and steering encoder counts. The sensors give an estimate of the amount of rotation of each wheel and the current steering angle. These measurements can be used to estimate the location of the platform. However, each measurement is prone to errors due to uncertainties in wheel parameters (e.g. radius) and wheel slip or skid. As each estimate of the position is based on the previous one, the error is unbound. Hence, localization based on dead-reckoning alone will get less accurate over time. The accuracy can be improved by applying certain rules to detect wheel slip and by using additional sensors such as an IMU. This last option will be discussed in Section V-E.

TABLE VII
Lookup table for the steering angles of the wheels. The reported values are for the Front Left and Rear Right wheel. For the Front Right and Rear Left wheels, the negative value of the reported values should be taken.

Encoder Count	**Steering Angle [rad]**
-28000	0.4210
-24000	0.3613
-20000	0.3019
-16000	0.2423
-12000	0.1824
-8000	0.1222
-4000	0.0614
0	0.000
4000	-0.0623
8000	-0.1255
12000	-0.1899
16000	-0.2557
20000	-0.3227
24000	-0.3913
28000	-0.4616

The update equations for dead-reckoning are based on [36]. These equations are for a four wheel steering vehicle, which is applicable to INTERACT. First, the cumulative encoder count N for the wheel rotation can be used to compute the incremental count ΔN:

$$\Delta N = N_i - N_{i-1} \tag{80}$$

From the incremental count, the incremental distance Δd can be computed using a conversion factor C based on the wheel diameter and the number of encoder pulses per revolution N_{rev}:

$$\begin{aligned} \Delta d &= C\Delta N && (81)\\ C &= \frac{2\pi r_w}{N_{rev}} && (82) \end{aligned}$$

The vehicle sideslip angle ψ needs to be computed based on measured data. This can be done directly through equation (77). The value of the front and rear imaginary wheel steering angles comes from the steering encoders. This computation is outlined in more detail in Section V-B. Using this, the incremental change of ψ is defined as

$$\Delta\psi = \psi_i - \psi_{i-1}. \tag{83}$$

Now, the update on the robot state can be computed by

$$\begin{aligned} \Delta\theta &= \frac{\Delta d}{a+b}\cos(\psi)(\tan(\theta_f) - \tan(\theta_r)) && (84)\\ \Delta x &= \Delta d\cos(\theta_{i-1} + \Delta\theta/2 + \psi + \Delta\psi/2) && (85)\\ \Delta y &= \Delta d\sin(\theta_{i-1} + \Delta\theta/2 + \psi + \Delta\psi/2) && (86) \end{aligned}$$

Alternatives for the updates of the heading are:

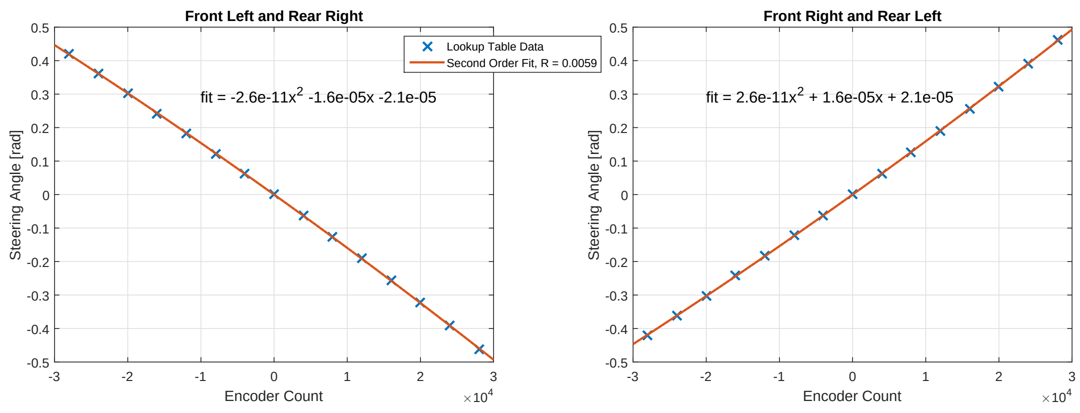


Fig. 33. Plots of the data points from the lookup table including the second order polynomial fit through the data.

$$\Delta\theta = -\frac{\Delta d}{a+b}\sin(\theta_f - \theta_r) \quad (87)$$

$$\Delta\theta = \frac{\Delta d_L \cos(\theta_L) - \Delta d_R \cos(\theta_R)}{w_B} \quad (88)$$

These are based on the equations of motion and taken from [21] respectively.

As can be seen from the equations, the actual steering angle needs to be derived from the encoder measurements. Details of this conversion are given in Subsection V-B.

Because the position estimate relies on the integration of incremental motion, the error on the position is unbounded and hence keeps increasing. Furthermore, the accuracy of these estimates suffers from different error sources, which makes the method unreliable. Two general types of errors can be distinguished, namely systematic and non-systematic. The sources that contribute most to these errors are listed in [37] and repeated here:

Systematic errors:

- Unequal wheel diameters
- Average of wheel diameters differs from nominal diameter
- Misalignment of wheels
- Uncertainty about the wheel base
- Limited encoder resolution
- Limited encoder sample rate

Non-systematic errors:

- Uneven floors
- Unexpected objects
- Wheel slip/skid

When the terrain is relatively smooth, the systematic errors are dominant. The contribution to this systematic localization error mainly comes from two sources: unequal wheel diameters and uncertainty about the wheel base [37]. Hence, the error can be reduced greatly when the wheel base and diameter are measured or estimated accurately. This can be done through a series of tests which is called *platform calibration*. A method is described in detail in Appendix VI. The results of calibrating the platform on the localization estimate is given in Subsection V-C.

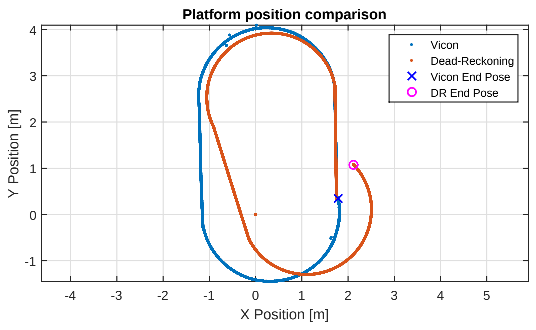


Fig. 34. Example run of the calibration experiment. The blue line shows the actual platform position measured with the Vicon system. The red line shows the platform location computed from localization. The final error which is used to compute the correction factors is the difference between the blue cross and the pink circle.

B. Steering Angle Encoders

The dead-reckoning equations introduced previously require measurement of the steering angles. The measurement is done through linear encoders on the steering actuators. The actual steering angles can be derived from the encoder counts by using a lookup table provided by the manufacturer. This table is given in Table V-A.

A second order polynomial is fitted to this data to be able to interpolate accurately between the given angles. A second order fit is chosen because this gives an exact fit, as shown in Figure 33

C. Odometry Calibration

The calibration method as described in Appendix VI is applied to the INTERACT platform. The reference trajectory of Figure 38 is driven by the robotic platform in an open loop fashion. The actual position of the platform is measured by the Vicon motion capture system. The location is also estimated using the dead-reckoning equations presented before. The resulting trajectory, as well as the dead-reckoning estimate, are shown in Figure 34 for one of the experiments. The parameters used in the equations are taken from documentation or CAD models and are:

- Distance between front and rear axle: $L = 0.8047$ meter
- Distance between left and right steering axle: $w_A = 0.78656$ meter

- Distance between steering axle and ground contact point: $w_o = 0.0711$ meter
- Wheel radius: $r_w = 0.24130$ meter

This experiment is repeated 5 times in clockwise direction and 5 times in counter-clockwise direction. The final x and y error in CW and CCW direction are computed from the difference of the localization estimate and the Vicon data. The result are plotted in Figure 35. The results of the experiments are used to solve te equations described in Appendix VI and find the correction factors for the platform parameters. With these corrections, the actual platform parameters can be estimated. These are found to be:

- $L = 0.9232$ meter
- $w_A = 0.8528$ meter
- $r_{wl} = 0.2408$ meter
- $r_{wr} = 0.2418$ meter

which show significant improvement in the localization errors, as is shown in Figure 35. Several results of localization experiments using different versions of the algorithm are given in Table VIII. The first column shows the results for dead-reckoning only, while the third column shows the results for a calibrated platform. Several other improvements on the algorithm are tested as well, which will be discussed next.

D. Improvements using Sensor Redundancy

In theory, the dead-reckoning equations given in (84) require that only one wheel is equipped with an encoder to compute Δd. In practice however, platforms usually come with encoders on at least one pair of wheels (left/right) if not all wheels. This sensor redundancy can be exploited to improve the localization estimate. This method is described in detail in [45] where a couple of so called Expert Rules are used to incorporate redundant sensors. The main benefit of redundancy and the expert rules that are used, is detection of wheel slip. The expert rule is as follows:

Expert Rule: Average + Smallest The difference between the incremental counts of both right and left side wheels is taken. If no slip occurs, this count should be closely matching. If slip occurs on one wheel, the encoder count will be significantly higher than that of the other wheel on the same side. In this case, the lower count is taken as correct.

Result of applying this rule to the localization estimate is given in Table VIII. As the surface on which the rover was driving during the experiments was very smooth, wheel slip was not an issue. Hence, the effect of applying this rule is rather small. Most likely the effect will be more visible on low traction, sandy terrain. This has, however, not been verified in an experiment as there is no proper location available.

Another type of redundancy that is present in the system is different sensors measuring the same quantity. For example, the accelerometers in the IMU can be used to compute an estimate of the current position, without using the wheel encoders at all. The use of an IMU in localization is called *odometry*. This technique, along with possible combination with dead-reckoning, is discussed in the next subsection.

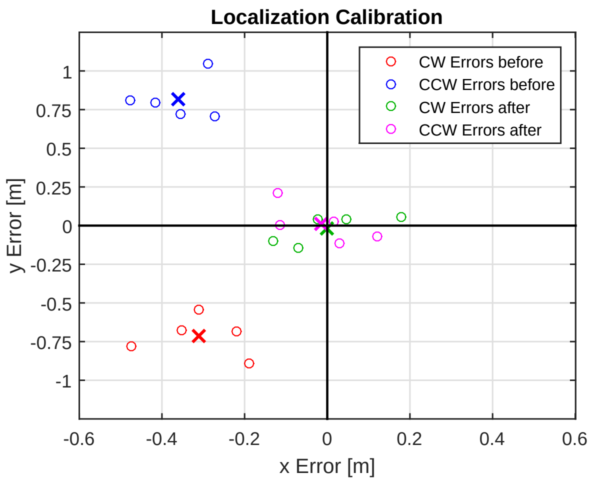


Fig. 35. Final position errors of the localization calibration experiment in CW and CCW direction *without* adjusting the platform parameters yet. The red circles show the CW results, with the red cross being the mean of the 5 runs. The blue circles represent the CCW results.

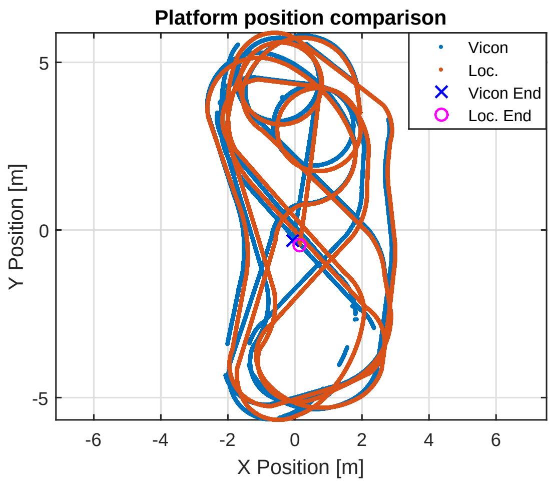
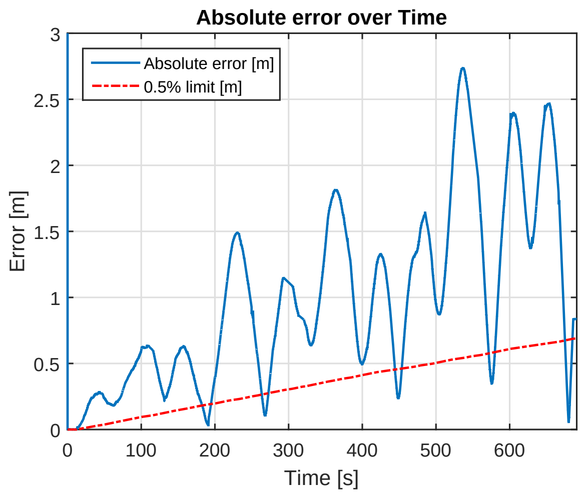


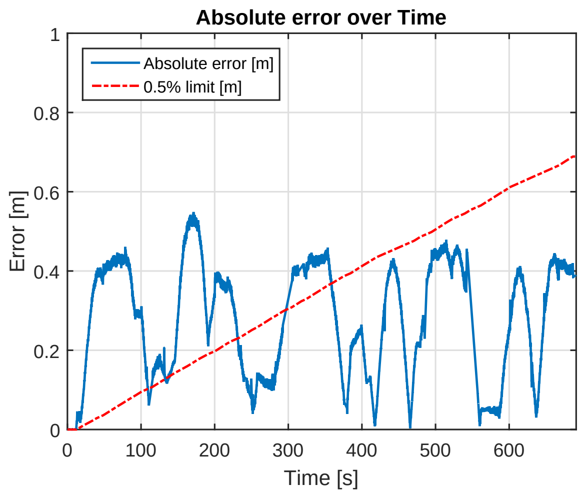
Fig. 36. Example of one of the localization results for a long distance run. The blue line shows the actual platform position as measured by the Vicon system. The red line shows the estimated position based on CDR with bias compensated odometry. The absolute position error over time is given in Figure 37.

E. Odometry

To improve the location estimate based on dead-reckoning alone, IMU data can be used. The simplest measure to improve localization is by taking the rate of change of the yaw angle from the gyroscopes and integrate the measurement to give a heading estimate directly. This replaces the heading update given in equation (84). However, the gyroscopes typically suffer from bias and drift. Because the measurement is integrated, these non-ideal sensor properties become especially apparent and degrade the results significantly. The bias and drift of the sensors can be estimated and compensated for. This can be done by using a Kalman filter. Sensor measurements from the IMU such as accelerometer data can also be fused with the dead-reckoning estimate using a Kalman filter. This technique has been attempted in the case of INTERACT. However, it turns out that the gyroscope bias can easily be estimated by averaging the sensor readings while the



(a) CDR and odometry



(b) CDR an bias compensated odometry

Fig. 37. Development of the error over time. The blue line shows the absolute error in meter. The dashed red line shows the acceptable limit of 0.5% of the travelled distance at that time instant.

platform is stationary, which once removed yields very accurate localization results. Incorporating accelerometer data using sensor fusion also turns out to be unnecessary, as the odometry estimate was always more accurate than the fused estimate. Nevertheless, fusion might have a more beneficial result on rougher terrain and this should be tested and verified in future experiments.

F. Results of Localization

The localization is implemented as described above. Several versions of the algorithm are tested and compared. The localization is tested on the motion primitives that were also used in the simulation validation described in Appendix III. Furthermore, the localization is tested on several longer distance runs. Four of these runs are performed with the following specifics:

- First run: only four wheel steering turns and straight driving.
- Second run: including turn in place maneuver.
- Third run: including reversals.
- Fourth run: Including crab motion.

The results of the comparison experiments are summarized in Table VIII. As an example, the results of the position measurement for one of the long distance runs are shown in Figure 36. The reported errors are the difference between the actual final position and the estimated final position expressed as a percentage of the travelled distance, as defined in equation (79) in Appendix III. The following versions have been tested:

- Simple dead-reckoning (DR).
- Dead-reckoning with platform calibration (CDR).
- CDR with expert rules.
- CDR with odometry; the gyroscope is used for the yaw measurement.
- CDR with odometry; bias compensated gyroscope yaw measurement.

From the error metric e_d of the final position it seems that bias compensation for the gyroscope does not greatly influence the results. However, if one looks at the error metric over time during the whole run, a great improvement can be seen as is shown in Figure 37). This indicates that only using the error in the final position can be misleading, and one should keep track of the development of this error over time.

Furthermore, the error exceeds the 0.5% limit in the beginning of the run. One possible explanation for this is that the reference point of the Vicon system does not exactly match the point P used in the planning algorithm. This will result in an error between the ground truth and the reference path that is not actually there. This becomes especially apparent while turning. However, this hypothesis has not been tested yet. One can see from Figure 37 that although the error oscillates, it does not seem to diverge and has an acceptable maximum value of 0.4 meter.

TABLE VIII
RESULTING ERRORS BETWEEN THE FINAL POSITION OF THE PLATFORM ACCORDING TO THE LOCALIZATION ALGORITHM AND ACCORDING TO THE GROUND TRUTH.

Experiment	**DR**	**DR expert rules**	**Calibrated DR (CDR)**	**CDR and Odo.**	**CDR and Bias Comp. Odo.**
Motion Primitives					
Straight driving	0.65%	0.93%	0.78%	0.30%	0.80%
Turn-in-place	5.96%	1.51%	3.36%	1.10%	1.05%
10° turn	5.14%	4.98%	3.42%	2.81%	2.84%
15° turn	3.78%	2.36%	2.10%	0.49%	0.59%
20° turn	1.00%	1.26%	1.23%	1.90%	2.47%
-10° turn	5.23%	3.58%	3.59%	3.64%	3.83%
-20° turn	5.62%	2.81%	3.57%	1.71%	1.75%
-25° turn	6.12%	6.67%	1.94%	2.07%	1.70%
Long distance runs					
Turns only	2.23%	1.48%	0.82%	0.61%	0.17%
Including turn in place	1.63%	3.71%	3.51%	0.24%	0.22%
Including reversals	6.54%	8.00%	1.96%	0.61%	0.74%
Including crab motion	4.55%	5.52%	1.17%	1.28%	0.42%

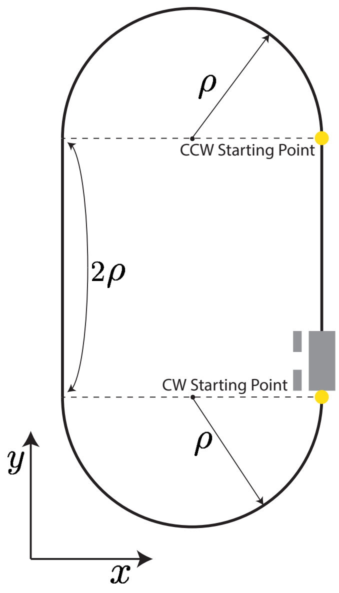


Fig. 38. Path used for calibration of the the platform. The parameter ρ describes the shape of the path, with the straight segments being 2ρ and the two semicircles having radius ρ. The path is executed in a feed-forward manner.

Appendix VI
Dead-Reckoning Calibration Method

As can be seen from (84), (87) and (88) derived above, the position estimate based on dead-reckoning relies on several platform parameters such as wheel base and wheel radius. Although the value of these parameters can be taken from the documentation that comes with the platform, the actual value might be slightly different due to e.g. tyre compression under load. Hence, the exact value of these parameters on the real platform need to be determined to reduce the effect of systematic errors and improve localization performance.

The process of accurately identifying these parameters is called platform calibration, and a systematic way to do this called UMBmark is explained in [37]. However, this method is designed for differential drive platforms with only two wheels (and a castor wheel for stability). In [51], the method is adapted for a four wheel car like robot with front wheel steering. Their version of the UMBmark algorithm requires only minor adaptations to suit the four wheel steering, four wheel drive platform used here. The method, including results, will be further described in this section.

To be able to calibrate the platform, a preprogrammed path is executed by the robot as is shown in Figure 38. The path consists of four segments: a straight path of length 2ρ, a semicircle with radius ρ, a straight path of length 2ρ and a semicircle with radius ρ. The final position of the platform is hence the same as the starting position. This path can be executed in a feedforward manner as a minor difference between initial and final positions are not a problem [37], as long as the two can be measured accurately. This measurement is done using the Vicon motion capture system that was also used during the model validation explained in Section III.

Two main sources of errors are targeted during this experiment, which are uncalibrated wheel base (designated Type A error) and unequal wheel diameter (designated Type B error). Both sources of error influence the localization estimate in a different way. The effect of the errors on the localization estimate will be evaluated separately. Later, the two contributions are superimposed to get the overall error on the localization.

A. Type A Errors

As can be seen from the dead-reckoning equations (84), an uncertainty in the wheel base will affect the estimated change in heading of the platform. It will not affect straight motions. Hence, the first segment of the path will be estimated correctly, but the first semicircle will be estimated with a different radius ρ_{odo} and will results in a heading change larger or smaller than $180°$ (depending on the sign of the wheel base error). The error in the heading angle is designated by α, as shown in Figure 39a. The final position estimated by the localization will hence be different from the actual final location. This effect is dependent on whether the trajectory is executed clockwise or counter-clockwise, so both cases are handled separately. By using the approximation for small angles:

$$\sin\alpha \approx \alpha \quad (89)$$
$$\sin 2\alpha \approx 2\alpha \quad (90)$$
$$\cos\alpha \approx 1 \quad (91)$$
$$\cos 2\alpha \approx 1 \quad (92)$$

the final error can be expressed as follows.

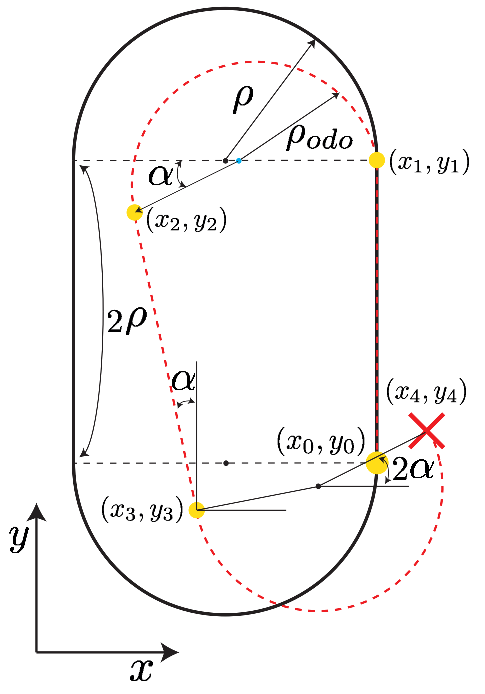
CW Direction:

$$y_1 = y_0 + 2\rho = 2\rho \quad (93)$$
$$x_1 = x_0 = 0 \quad (94)$$
$$y_2 = y_1 - \rho_{odo}\sin(\alpha) \approx 2\rho - \rho_{odo}\alpha \quad (95)$$
$$x_2 = x_1 - \rho_{odo} - \rho_{odo}\cos(\alpha) \approx -2\rho_{odo} \quad (96)$$
$$y_3 = y_2 - 2\rho\cos(\alpha) \approx -\rho_{odo}\alpha \quad (97)$$
$$x_3 = x_2 + 2\rho\sin(\alpha) \approx 2(\rho\alpha - \rho_{odo}) \quad (98)$$
$$y_4 = y_3 + \rho_{odo}\sin(\alpha) + \rho_{odo}\sin(2\alpha) \approx 2\rho_{odo}\alpha \quad (99)$$
$$x_4 = x_3 + \rho_{odo}\cos(\alpha) + \rho_{odo}\cos(2\alpha) \approx 2\rho\alpha \quad (100)$$

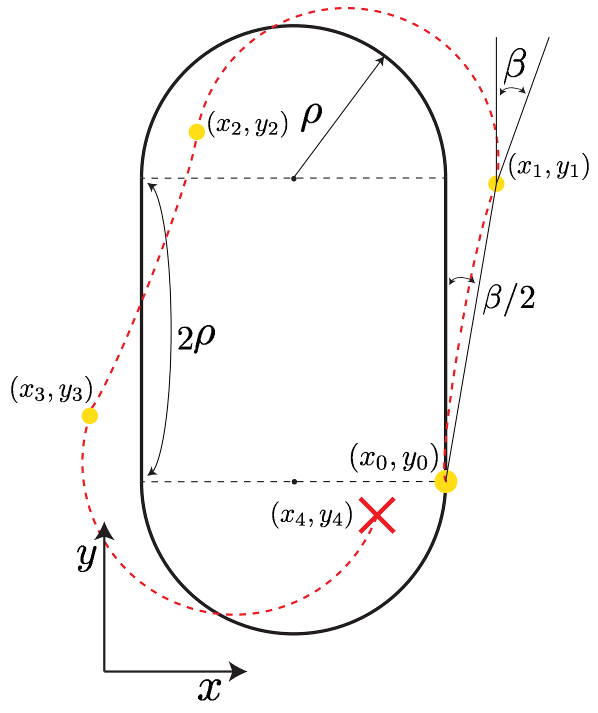
CCW Direction:

$$y_1 = y_0 - 2\rho = -2\rho \quad (101)$$
$$x_1 = x_0 = 0 \quad (102)$$
$$y_2 = y_1 + \rho_{odo}\sin(\alpha) \approx -2\rho + \rho_{odo}\alpha \quad (103)$$
$$x_2 = x_1 - \rho_{odo} - \rho_{odo}\cos(\alpha) \approx -2\rho_{odo} \quad (104)$$
$$y_3 = y_2 + 2\rho\cos(\alpha) \approx \rho_{odo}\alpha \quad (105)$$
$$x_3 = x_2 + 2\rho\sin(\alpha) \approx 2(\rho\alpha - \rho_{odo}) \quad (106)$$
$$y_4 = y_3 - \rho_{odo}\sin(\alpha) - \rho_{odo}\sin(2\alpha) \approx -2\rho_{odo}\alpha \quad (107)$$
$$x_4 = x_3 + \rho_{odo}\cos(\alpha) + \rho_{odo}\cos(2\alpha) \approx 2\rho\alpha \quad (108)$$

To eliminate ρ_{odo} from the equations, the fact that the travelled distance S along the semicircle is identical for both the reference path and the path containing the localization error is used. This is because only Type A errors are



(a) Type A



(b) Type B

Fig. 39. The effect of Type A errors (left) and Type B errors (right) on the localization estimate. For Type A errors, straight line segments are unaltered, while curves have a modified radius of rotation ρ_{odo}. This results in an extra heading change of angle α. For Type B errors, curves are unaltered and straight line segments show a slight curve which results in a heading angle error β

considered and hence the wheel radius is taken to be equal. This results in the following relation:

$$S_1 = \rho\pi \tag{109}$$
$$S_2 = \rho_{odo}(\pi + \alpha) \tag{110}$$
$$S_1 = S_2 \tag{111}$$
$$\rho_{odo} = \frac{\rho\pi}{\pi + \alpha} \tag{112}$$

The final position errors for Type A then become:

$$x_{4,cw} = 2\rho\alpha \tag{113}$$
$$y_{4,cw} = 2\frac{\rho\pi}{\pi + \alpha}\alpha \tag{114}$$
$$x_{4,ccw} = 2\rho\alpha \tag{115}$$
$$y_{4,ccw} = -2\frac{\rho\pi}{\pi + \alpha}\alpha \tag{116}$$

B. Type B Errors

The effect of unequal wheel diameters cannot directly be seen from the equations presented before. However, the effect on odometry is easily derived. When the diameter of an actuated wheel pair is not equal, the wheel with a larger diameter will travel a larger distance, and hence a path that would normally be straight, has now a curve away from the side of the larger wheel. As it is assumed that the wheel base is correctly estimated (we only consider Type A and B errors separately and apply superposition later), a curved path resulting from a certain steering angle will not show show any deviation from what is expected (i.e. radius of rotation ρ is as expected). The effect of Type B errors is shown in Figure 39b. The final error in the localization estimate can be expressed as follows:

CW Direction:

$$y_1 = y_0 + 2\rho\cos(\beta/2) \approx 2\rho \tag{117}$$
$$x_1 = x_0 + 2\rho\sin(\beta/2) \approx \rho\beta \tag{118}$$
$$y_2 = y_1 + 2\rho\sin(\beta) \approx 2\rho(1 + \beta) \tag{119}$$
$$x_2 = x_1 - 2\rho\cos(\beta) \approx \rho(\beta - 2) \tag{120}$$
$$y_3 = y_2 - 2\rho\cos(3\beta/2) \approx 2\rho\beta \tag{121}$$
$$x_3 = x_2 - 2\rho\sin(3\beta/2) \approx -2\rho(1 + \beta) \tag{122}$$
$$y_4 = y_3 - 2\rho\sin(2\beta) \approx -2\rho\beta \tag{123}$$
$$x_4 = x_3 + 2\rho\cos(2\beta) \approx -2\rho\beta \tag{124}$$

CCW Direction:

$$y_1 = y_0 - 2\rho\cos(\beta/2) \approx -2\rho \tag{125}$$
$$x_1 = x_0 - 2\rho\sin(\beta/2) \approx -\rho\beta \tag{126}$$
$$y_2 = y_1 + 2\rho\sin(\beta) \approx 2\rho(\beta - 1) \tag{127}$$
$$x_2 = x_1 - 2\rho\cos(\beta) \approx -\rho(2 + \beta) \tag{128}$$
$$y_3 = y_2 + 2\rho\cos(3\beta/2) \approx 2\rho\beta \tag{129}$$
$$x_3 = x_2 + 2\rho\sin(3\beta/2) \approx 2\rho(\beta - 1) \tag{130}$$
$$y_4 = y_3 - 2\rho\sin(2\beta) \approx -2\rho\beta \tag{131}$$
$$x_4 = x_3 + 2\rho\cos(\beta) \approx 2\rho\beta \tag{132}$$

To be able to quantify the difference in wheel diameter, a relation between β and the wheel diameter is needed. To derive this relation, the geometry of the straight line segment of the path is used. The geometry is shown in Figure 40. Note that this is highly exaggerated.

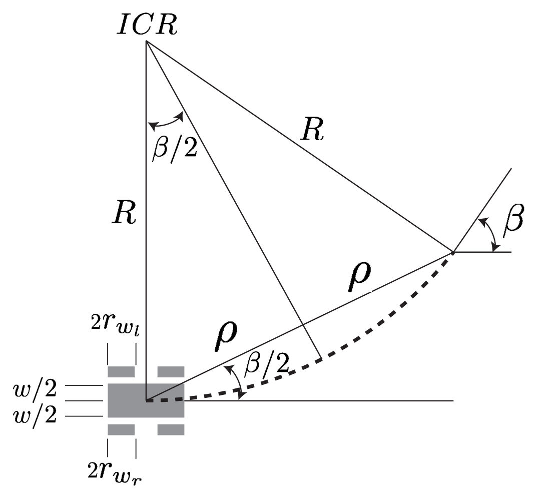


Fig. 40. Schematic representation of driving a straight segment with unequal wheel diameter. Note that the effect is greatly exaggerated. Because of the difference in diameter, the platform will curve and slightly change heading. The final heading change is β. The radius of the rotation is R.

From the geometry it follows that the radius of rotation R is related to ρ and β as

$$R = \frac{\rho}{\sin(\beta/2)}. \tag{133}$$

Furthermore, the ratio of the wheel diameters is related to the ratio of the radius of rotation for each wheel $R \pm w/2$

$$E_d = \frac{r_{w_l}}{r_{w_r}} = \frac{R - w/2}{R + w/2}. \tag{134}$$

Hence, if β is known, the ratio of the wheel diameters can be computed.

To find the combined total localization error, the Type A and Type B errors need to be superimposed. This means that (99) through (132) need to be combined. This gives:

$$\begin{aligned} x_{cw} &= 2\alpha\rho - 2\beta\rho && (135) \\ y_{cw} &= \frac{2\pi\alpha\rho}{\pi + \alpha} - 2\beta\rho && (136) \\ x_{ccw} &= 2\alpha\rho + 2\beta\rho && (137) \\ y_{ccw} &= -\frac{2\pi\alpha\rho}{\pi + \alpha} - 2\beta\rho && (138) \end{aligned}$$

Equations (135), (136), (137) and (138) can be used to solve for α and β:

$$\begin{aligned} \alpha_1 &= \frac{x_{cw} + x_{ccw}}{4\rho} && (139) \\ \alpha_2 &= \frac{\pi(y_{cw} - y_{ccw})}{y_{ccw} - y_{cw} + 4\pi\rho} && (140) \\ \beta_1 &= -\frac{x_{cw} - x_{ccw}}{4\rho} && (141) \\ \beta_2 &= -\frac{y_{cw} + y_{ccw}}{4\rho} && (142) \end{aligned}$$

Theoretically, (139) and (140) as well as (141) and (142) should yield an identical result. In practice the result will differ slightly due to the presence of other (systematic) errors, but this difference should be small. α and β are finally taken as the average of (139) and (140), and (141) and (142) respectively.

Now that α and β are estimated, the correction factors for the wheel base E_b and wheel diameters E_d can be computed. For the wheel base, this is relatively straightforward, as the ratio of the actual and the nominal wheel base are related to α, similar to (112):

$$E_b = \frac{w_{\text{actual}}}{w_{\text{nominal}}} = \frac{\pi}{\pi - \alpha} \tag{143}$$

The corrected wheel base simply becomes:

$$w_{\text{actual}} = E_b w_{\text{nominal}} \tag{144}$$

For the wheel diameter, there is a slight complication. Increasing one of the wheel diameters with the correction factor E_d will increase the *average nominal radius* r_a of the two wheels:

$$r_a = \frac{r_{w_r} + r_{w_l}}{2} \tag{145}$$

However, this quantity should not be altered as it was assumed to be correct during the experiments. Alternatively, one wheel diameter needs to be slightly lowered, while the other needs to be slightly increased, such that the average nominal diameter stays the same. This can be done by solving (134) and (145) for the wheel radii:

$$\begin{aligned} r_{w_l} &= \frac{2}{E_d + 1} r_a = c_l r_a && (146) \\ r_{w_r} &= \frac{2}{E_d^{-1} + 1} r_a = c_r r_a && (147) \end{aligned}$$

which results in the two correction factors c_l and c_r for the left and the right wheel respectively.

Appendix VII
Nonholonomic Path Planning using RRT

Rapid-exploration Random Tree (RRT) is a probabilistic path planning algorithm. It is very suitable for single query problems as it does not require a preprocessing step. Furthermore, because the algorithm does not contain many heuristics and tunable parameters, it is easy to implement and performance analysis is straightforward. The key feature of RRTs is that exploration of the state space is biased towards unexplored parts, hence the name. The algorithm generally has consistent behaviour, is probabilistically complete, and the tree is always connected although the number of edges in the tree is minimal [7]. The RRT algorithm does not optimize the path in any way, but instead finds a "good enough" solution between start and goal state. However, experiments by LaValle et al. [7], [52] show that the solution paths are near optimal (length wise), within a factor of 1.3 to 2.0.

The RRT algorithm is chosen because of the useful properties described above, which satisfy the requirements of the INTERACT project stated earlier. Furthermore, nonholonomic constraints are easily implemented in the algorithm, as will be described later in this section. The way the algorithm works as well as its specific implementation for INTERACT is described below.

A. The RRT Algorithm Implementation

As the name suggests, the RRT algorithm is based on the construction of a tree $\mathcal{T}$ in the state space. The pseudocode of the basic algorithm is shown in Algorithm 5. The tree is rooted at the start state of the system, $\boldsymbol{q}_{init}$. Iteratively, a random sample state $\boldsymbol{q}_{rand}$ is taken from the state space, towards which the tree is extended (see Algorithm 7). This is done by selecting a state $\boldsymbol{q}_{near}$ that is already present in the tree and which is closest to the random state. "Closest" is defined according to a distance metric ρ which can range from simple Euclidean distance to complex metrics that capture limitations of the system such as nonholonomic constraints. The nearest neighbour is found using the `nearestNeighbour` function shown in Algorithm 6. The distance metric ρ between two states is computed using the following function:

$$\begin{aligned}\rho(\boldsymbol{q}_1,\boldsymbol{q}_2) \quad = \quad & k_{pos}\sqrt{(x_{\boldsymbol{q}_1}-x_{\boldsymbol{q}_2})^2+(y_{\boldsymbol{q}_1}-y_{\boldsymbol{q}_2})^2} \\ & +k_\theta\left[1-\cos(\theta_{\boldsymbol{q}_1}-\theta_{\boldsymbol{q}_2})^2\right], \qquad (148)\end{aligned}$$

which weights the Euclidean distance between $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ with the difference in heading using a position gain k_{pos} and a heading gain k_θ.

A new state is computed by integrating the equations of motion $f(t,u)$ for the time increment Δt. This action is called incremental simulation. Note that this incremental simulation makes it possible to add a time schedule to the path, as each state receives a time stamp. This can be useful for the feedback control, which will become apparent in Section VIII.

The new state is stored in the tree along with the input (which is called an edge between states) that is required

Algorithm 5 General RRT Algorithm

```
1: $\mathcal{T}$.init($\boldsymbol{q}_{init}$)
2: while !goalReached do
3:     $\boldsymbol{q}_{rand}$ ← generateRandomState(bounds);
4:     $\boldsymbol{q}_{near}$ ← nearestNeighbour($\boldsymbol{q}_{rand}$,$\mathcal{T}$)
5:     $[S, \boldsymbol{q}_{new}]$ ← extendTree($\boldsymbol{q}_{rand}$,$\boldsymbol{q}_{near}$,$\mathcal{T}$)
6: end while
```

Algorithm 6 RRT Nearest Neighbour

```
 1: function NEARESTNEIGHBOUR($\boldsymbol{q}_1$,$\mathcal{T}$)
 2:     $d = \infty$
 3:     for all $\boldsymbol{q} \in \mathcal{T}$ do
 4:         if $\rho(\boldsymbol{q},\boldsymbol{q}_1) < d$ then
 5:             $\boldsymbol{q}_{near} = \boldsymbol{q}$
 6:             $d = \rho(\boldsymbol{q},\boldsymbol{q}_1)$
 7:         end if
 8:     end for
 9:     return $\boldsymbol{q}_{near}$
10: end function
```

to get from $\boldsymbol{q}_{near}$ to $\boldsymbol{q}_{new}$. Here it becomes apparent why it is useful to have the same input for the equations of motion and the actual platform: the edge that stores the inputs can directly be fed to the platform to have it execute the trajectory in a feedforward manner.

Also, an identifier for the parent state $\boldsymbol{q}_{near}$ is stored with the new state, which is needed to trace back the path from goal to start when a solution is found.

The tree extension process is repeated until the goal state $\boldsymbol{q}_{goal}$ or goal region is reached. The algorithm then stores the resulting path from $\boldsymbol{q}_{init}$ to $\boldsymbol{q}_{goal}$. The tree extension process is schematically clarified in Figure 41.

If obstacles are present in the workspace, collision detection should also be implemented. The checking is done when a new state is about to be added to the tree. The new state, including all the states that are part of the edge, are checked for collisions.

Application of the RRT algorithm to nonholonomic systems is very straightforward. By having the incremental simulator integrate the (nonholonomic) equations of motion $f(t,u)$, the resulting path will automatically satisfy the nonholonomic constraints and hence be feasible for the system to execute. However, it is close to impossible to exactly reach a (random) state when subject to the nonholonomic constraints. Hence, the algorithm will return *Reached* when the two states are within a certain threshold $\rho(\boldsymbol{q}_1,\boldsymbol{q}_2) < \epsilon$ from each other.

This threshold must be chosen small enough to have the algorithm reach the goal position with acceptable precision, but large enough such that convergence to the goal does not take too much time.

The algorithm explorers the state space relatively fast, however, convergence to the goal state or region might take a very long time as the chances that the goal state is taken as $\boldsymbol{q}_{rand}$ is infinitesimally small. There exist several improvements to the algorithm that reduce the convergence time. One of these solutions is goal bias, which means that, with a certain probability, the goal location is selected as a "random state". The bias towards the goal does not have to be large to significantly speed up convergence

Algorithm 7 RRT Extend

```
1:  function EXTENDTREE($q_{target}$,$q_{near}$,$\mathcal{T}$)
2:      $q_{new} \leftarrow$ generateState($q_{near}$,$q_{target}$,$\mathcal{T}$)
3:      if checkCollision($q_{new}$) then
4:          $\mathcal{T}$.addVertex($q_{new}$)
5:          $\mathcal{T}$.addEdge($u_{new}$)
6:          if $\rho(q_{new}, q_{target}) < \epsilon$ then
7:              $S$ = *Reached*
8:          else
9:              $S$ = *Advanced*
10:         end if
11:     end if
12:     $S$ = *Trapped*
13:     return $[S, q_{new}]$
14: end function
```

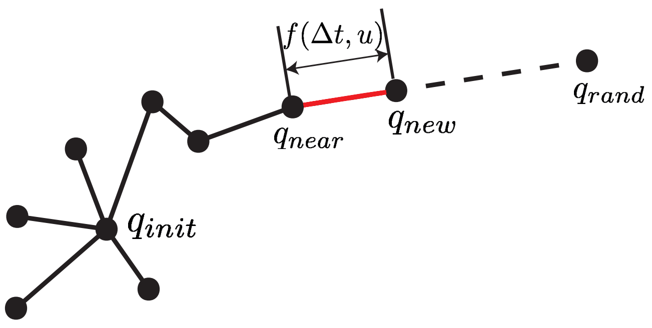


Fig. 41. Extension process of the RRT algorithm. Image adapted from [5].

(e.g. a bias of 0.05) [7]. This can also be seen from the experimental results given in Section VII-E. Even better results can be achieved by applying a method called goal zoom also introduced in [7]. The selection of the random state is again biased towards the goal. However, instead of taking the exact goal state a state is picked from a region around the goal. The size of this region is defined by the closest vertex to the goal currently in the tree. As a result, this will gradually focus state selection on the goal as the tree grows towards it. For performance analysis, also see Table X in Section VII-E.

A more aggressive version of the algorithm is RRT-Connect [5]. In this version, the *extend* operation is replaced by the *connect* operation, see Algorithm 8. The *connect* operation will iteratively apply the *extend* function until it returns either *Reached*, which means the tree was able to directly reach the random state q_{rand}, or *Trapped*, which means extension is not possible anymore and a new random state is chosen. Performance of both versions is analysed in Section VII-E.

Algorithm 8 RRT Connect

```
1: function CONNECTTREE($q$,$\mathcal{T}$)
2:     repeat
3:         $q_{near} \leftarrow$ nearestNeighbour($q$,$\mathcal{T}$)
4:         $[S, q_{new}] \leftarrow$ extendTree($q$,$q_{near}$,$\mathcal{T}$)
5:     until not ($S$ = *Advanced*)
6:     return $S$
7: end function
```

B. Bidirectional RRT

Another performance improvement can be expected by applying a bidirectional search, as this technique also improved results of other, more classical, search techniques [6]. With this technique, two trees are grown, one rooted at the start state and one rooted at the goal state. Besides extending the tree using random samples from the state space, as with the basic RRT algorithm, half of the computation time is spend on trying to grow the trees towards each other and connect them. This slightly adapted version of the RRT algorithm is given in Algorithm 9.

Algorithm 9 Bidirectional RRT Algorithm

```
1:  $\mathcal{T}_a$.init($q_{init}$)
2:  $\mathcal{T}_b$.init($q_{goal}$)
3:  while !treesConnected do
4:      $q_{rand} \leftarrow$ generateRandomState(bounds);
5:      $q_{near_a} \leftarrow$ nearestNeighbour($q_{rand}$,$\mathcal{T}$)
6:      $[S_a, q_{new_a}] \leftarrow$ extendTree($q$,$q_{near_a}$,$\mathcal{T}$)
7:      if $S_a$ != *Trapped* then
8:          $q_{near_b} \leftarrow$ nearestNeighbour($q_{new_a}$,$\mathcal{T}$)
9:          $[S_b, \sim] \leftarrow$ extendTree($q_{new_a}$,$q_{near_b}$,$\mathcal{T}$)
10:         if $S_b$ = *Reached* then
11:             return path($\mathcal{T}_a, \mathcal{T}_b$)
12:         end if
13:         swap($\mathcal{T}_a, \mathcal{T}_b$)
14:     end if
15: end while
```

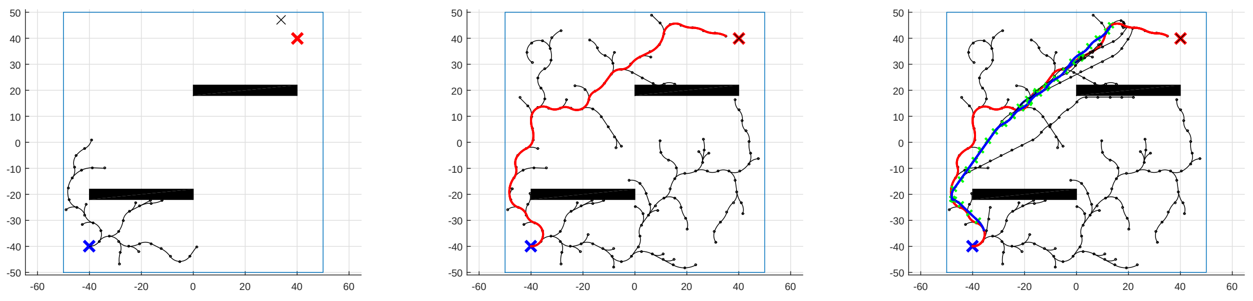
Just like with the general RRT algorithm, one or both of the *extend* operations in line 6 and 9 can be replaced with *connect*. This results in a much more ”aggressive” algorithm.

An issue arises when the bidirectional search is applied to nonholonomic systems. Because the algorithm returns *Reached* when a state from $\mathcal{T}_a$ is within ϵ from a state in $\mathcal{T}_b$ (or vice versa), there is a discontinuity in the path at the transition of the trees. An appropriate solution needs to be found to reduce the impact of this discontinuity and/or handle this transition properly. A planning solution that keeps the discontinuities as small as possible is discussed in Section VII-C2. Furthermore, a control algorithm that can handle (small) discontinuities can be chosen. The performance of two controllers w.r.t. discontinuities is discussed in Section VIII.

C. Several Improvements

Besides the different versions of the RRT algorithm that were presented in the previous section, a couple of other improvements can be thought of. Two improvements are particularly of interest for the INTERACT project. These improvements are discussed below.

1) Path Smoothing: Because of the random nature of the planning algorithm, the solution that is found will most likely contain many unnecessary bends and curves. These curves increase the path length and make it more complicated, which is not desirable. In an attempt to get rid of these curves, a preprocessing step called path smoothing is applied. During this step, two random vertices q_1 and q_2 belonging to the path are selected. Then, by using the



(a) Tree growing (b) Resulting Path (c) Path Smoothing

Fig. 42. Visualization of tree growing: showing $\boldsymbol{q}_{rand}$ with a black cross, $\boldsymbol{q}_{new}$ with a black circle, and the new edge with a black line. The resulting path is indicated in red. If path smoothing takes place, the smoothed segments are colored blue.

connect function, an attempt is made to find a better path between the two vertices. In this case, better is defined as being shorter and/or containing less reversals. If such a better path is found, the existing segment is replaced by the new one.

The *connect* function is chosen over the *extend* function because it will result in a more straight (and hence more desirable) path to $\boldsymbol{q}_2$. This is because there is no random state selected. Instead, $\boldsymbol{q}_2$ is always taken as $\boldsymbol{q}_{rand}$, which results in an aggressive attempt to connect the $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$.

The path smoothing can be applied multiple times. However, a balance needs to be found between increased computation time and increased path smoothness. The number of smoothing iterations is chosen as a fraction of the number of vertices of the initial path found. This fraction is a tuning parameter that can be chosen depending on path requirements and computation time available.

A major downside of path smoothing is that it increases the number of discontinuities in the path. Because the *connect* function returns *reached* when the end of the smoothed segment is within ϵ of $\boldsymbol{q}_2$, every smoothed segment adds a discontinuity to the path. If these are not handled properly, the benefit of smoothing is nullified. A method to cope with the discontinuities is discussed next.

2) Handling of Discontinuities: In the case of planning for nonholonomic systems, as is the case with the INTERACT platform, two states will never be completely identical, but a similarity threshold is used on the distance metric ρ (as described in Section VII-A). As discussed before, this will result in discontinuities when using the bidirectional algorithm and path smoothing. A (feedback) control algorithm might be able to handle these discontinuities correctly. Nevertheless, it is beneficial to keep them as small as possible to make the platform behaviour predictable. This can be achieved by selecting the similarity threshold ϵ relatively small. However, the algorithm might have trouble reaching this similarity threshold, which will greatly increase computation time of the bidirectional algorithm and will result in many failed attempts on path smoothing. This is especially evident when a relatively small set of inputs is available to the system.

To solve this issue, the set of inputs is extended when two vertices are close to each other and a connection is needed. This is the case when the two trees of the bidirectional algorithm are close to each other, or vertices in the path segment of a smoothing step are close to $\boldsymbol{q}_2$, the goal vertex. When a certain distance threshold is reached, the set of inputs gets extended with inputs containing lower speed and higher steering angles. This keeps the discontinuity small, while not significantly increasing the computation time. The exact input set that is used is described in Section VII-D.

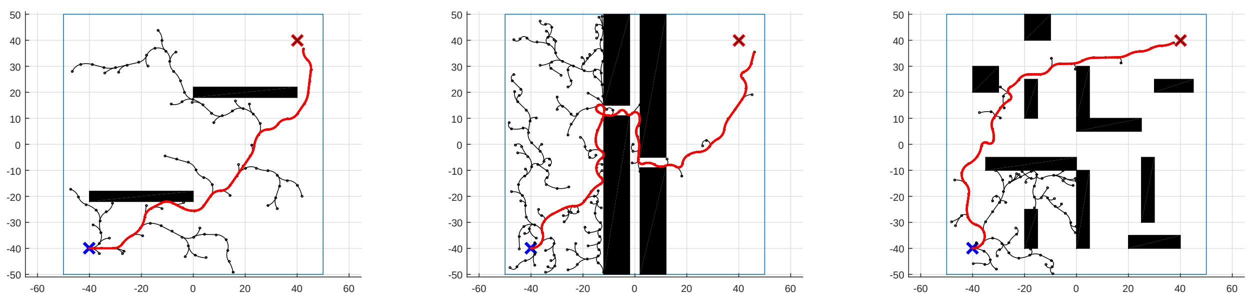
D. Implementation for INTERACT

The RRT algorithm (both general and bidirectional as well as *extend* and *connect*) was written in Matlab. For the incremental simulator, the equations of motions as derived in Appendix II are used. A discrete set of inputs is selected to generate new states in the state space. Three different test environments are defined to assess the performance of the algorithm.

To monitor the execution of the algorithm, the construction of the tree is visualized in Matlab. The random state $\boldsymbol{q}_{rand}$ is indicated by a thin black cross. The new state that gets added to the tree (either by minimizing the distance or by random selection) is shown by a small black circle. The path (or edge) to the new state is visualized by a thin black line. When the path to the goal is found, vertices and edges belonging to the path are coloured red. If path smoothing takes place, path segments that are smoothed are added in blue. Obstacles are visualized by black rectangles as is shown in Figure 42.

When the solution path is found, the platform can execute the path by applying the input stored in the edges that belong to the path. While executing, the platform is visualized in SPANviewer (main details can be found in Section IV). That path is added by showing waypoints (spheres) along the path with a small reference frame indicating the heading of the platform. See also Figure 44.

The RRT algorithm generates new states by applying a certain input to the system and integrating the equations of motion. The input can be selected from a pre defined discrete set of inputs both randomly or by finding the one that minimizes the distance to $\boldsymbol{q}_{rand}$ according to the metric ρ. A second option is to generate a random input within the systems limitations/boundaries every time the EOM are integrated. The performance of both techniques is investigated and results are given in Section VII-E, Table X.



(a) Simple (b) Narrow Corridor (c) Cluttered

Fig. 43. Three environments used to test the performance of the RRT planning algorithm.

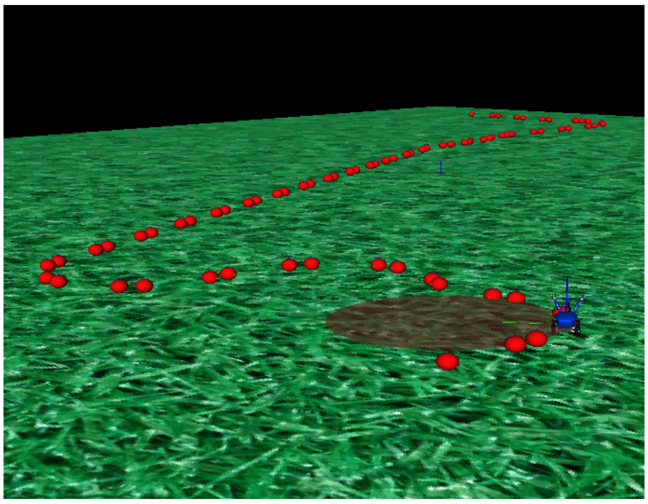


Fig. 44. Visualization of the solution path and path execution by the (simulated) platform in SPANviewer.

TABLE IX
INPUT SETS USED FOR THE PERFORMANCE ASSESSMENT OF THE RRT ALGORITHM. THE INPUT VECTORS CONTAIN THE VELOCITY INPUT AND THE STEERING ANGLE INPUT.

Input Set #	**Available inputs**
Set 1	$\begin{bmatrix}1\\0\end{bmatrix}$ $\begin{bmatrix}1\\\pm0.05\end{bmatrix}\begin{bmatrix}1\\\pm0.1\end{bmatrix}$
Set 2	Set 1 $\begin{bmatrix}0.8\\\pm0.2\end{bmatrix}\begin{bmatrix}0.6\\\pm0.3\end{bmatrix}\begin{bmatrix}0.4\\\pm0.4\end{bmatrix}$

When a discrete set of inputs is used, several possibilities need to be considered. An extensive set will make many maneuvers and motions possible, but will also increase the computation time of the algorithm if every input of the set is tried. Furthermore, it is not guaranteed that the resulting path will be better than when a limited set of inputs is used. To investigate this, several sets are defined and applied to the same environment. The input sets are given in Table IX.

The two inputs to the platform and algorithm are the velocity u_1 and the steering angle in radians u_2. The inputs contained in the sets are represented by a vector $\begin{bmatrix} u_1 & u_2 \end{bmatrix}^T$. If both steering to the left and to the right is allowed, a $\pm$ is added in front of u_2. If also a reversed velocity is allowed, a $\pm$ is added in front of the input vector.

The three test environments that are chosen to assess the performance are described below and shown in Figure 43. Each two dimensional environment is bounded to be within $[-50, 50]$ in both x and y. The starting state, indicated by a blue cross, is at $[-40, -40]$, while the goal position, indicated by a red cross, is at $[40, 40]$.

- Simple: See Figure 43a. Environment that should be easily solvable containing only two obstacles. There is a lot of free space.
- Narrow Corridor: See Figure 43b. Environment containing a narrow path that has to be taken to reach the goal. Probabilistic planners generally have difficulties finding this narrow path.
- Cluttered: See Figure 43c. Environment containing a high obstacle density.

E. Algorithm Performance Analysis

Several simulations have been performed using the RRT algorithm described in this Section. The performance of each version of the algorithm is judged based on the time needed for planning, the time needed for smoothing and the path length.

This is done for each of the three environments described above. Results of the performance analysis are given in Table X. The maximum number of iterations for tree extensions was set to 500. If there was no solution found within this number of iterations, the algorithm is considered to have failed. The number of failures is given in parentheses in the column for maximum plan time.

F. Path Planning Experiments

Several preliminary experiments on path planning have been performed on the real platform to test the performance. The platform was positioned in a workspace of roughly 10 by 5 meter. The initial pose of the platform was taken from the Vicon motion capture system and used as the initial position $\boldsymbol{q}_{init}$ for the RRT algorithm. The goal location $\boldsymbol{q}_{goal}$ was selected to be as far from the platform as possible. Virtual objects could be added to the planner to increase the difficulty and extend the path length. The path was planned off-line and the solution path including the feedforward command was sent to the platform. During execution, important platform parameters were logged and the platform position and heading was measured using the

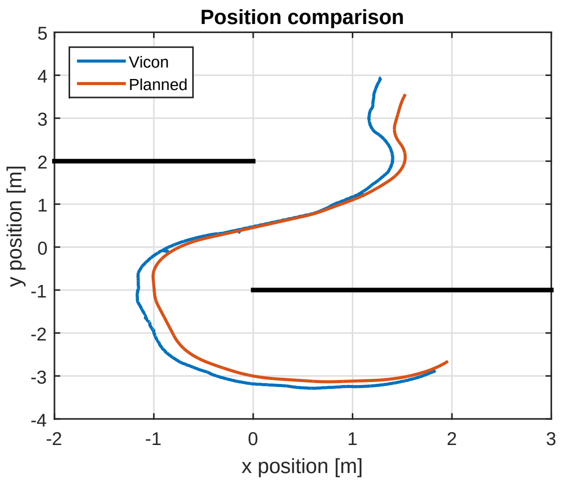


Fig. 45. Example run of the path planning experiments.

Vicon setup. A sample run showing the planned path and the actual path executed using the feedforward command is shown in Figure 45. Despite the lack of feedback control, the resulting path is quite close to the planned path.

TABLE X
PERFORMANCE ANALYSIS OF DIFFERENT VERSIONS OF THE RRT ALGORITHM. THE NUMBER BETWEEN PARENTHESES IN THE COLUMN LABELLED "MAX. PLAN TIME" INDICATES THE NUMBER OF FAILED PLANNING ATTEMPTS, I.E. WHERE MORE THAN THE MAXIMUM NUMBER OF ITERATIONS WAS REQUIRED. BOLD ITEMS ARE THE BEST RESULT FOR THAT SPECIFIC COLUMN.

		Simple				Narrow corridor				Cluttered			
Simulation	**Input set**	Avg. plan time [s]	Max. plan time [s][1]	Avg. smooth time [s]	Avg. Length [m]	Avg. plan time [s]	Max. plan time [s]	Avg. smooth time [s]	Avg. Lengt [m]h	Avg. plan time [s]	Max. plan time [s]	Avg. smooth time [s]	Avg. Length [m]
Single tree	Set 1	32.64	63.39 (0)	9.89	145.26	467.23	865.32 (44)	**8.69**	173.23	74.65	172.32 (0)	21.01	159.95
Single tree goal bias (0.05)	Set 1	23.13	85.85 (0)	10.45	141.10	271.05	792.12 (37)	10.09	179.38	56.23	234.21 (0)	19.30	158.12
Single tree goal bias (0.2)	Set 1	17.98	49.59 (0)	9.75	136.52	443.31	1247 (37)	9.56	172.45	61.78	286.95 (0)	**19.22**	158.42
Single tree goal zoom (0.05)	Set 1	20.54	51.37 (0)	**9.37**	140.99	424.02	772.91 (42)	10.39	189.58	58.35	146.27 (0)	21.16	161.67
Single tree goal zoom (0.2)	Set 1	19.49	47.48 (0)	10.09	144.73	337.57	908.05 (41)	9.26	**170.81**	47.66	136.51 (0)	20.89	159.67
Bidirectional	Set 1	**11.99**	**22.76 (0)**	10.40	158.53	318.94	497.07 (40)	8.88	176.86	**47.17**	**110.48 (0)**	23.06	162.50
Single tree including reversals	±Set 1	26.78	67.99 (0)	11.88	147.39	350.50	673.67 (39)	13.00	203.88	67.47	155.96 (0)	25.30	160.34
Bidirectional including reversals	±Set 1	12.85	27.41 (0)	13.01	176.41	168.32	252.71 (38)	14.06	183.94	66.54	150.22 (0)	28.93	173.52
Single tree random input	5 Random	45.77	71.48 (28)	24.63	156.24	**81.20**	**99.81** (48)	26.03	181.99	102.80	166.70 (15)	53.63	170.93
Bidirectional random input	5 Random	30.30	107.94 (0)	29.27	191.38	91.16	137.02 (31)	30.18	208.78	117.36	258.37 (5)	63.58	188.00
Single tree extended input set	Set 2	29.58	94.02 (0)	18.88	**129.64**	107.10	291.64 (21)	18.81	174.40	84.28	200.66 (0)	38.01	**152.33**
Bidirectional extended input set	Set 2	19.86	46.61 (0)	19.89	165.22	96.83	291.62 **(4)**	21.54	180.39	88.71	165.10 (0)	44.10	162.92

Appendix VIII
Feedback Control of Nonholonomic Systems

The path planner presented in Appendix VII generates both a reference trajectory (full pose consisting of x, y and θ) and the platform input required to track this path. In principle, the path can be executed by using only the generated input, i.e. in a feedforward manner. However, due to inaccuracies in the model and possible disturbances during execution, the platform might deviate from the path. To compensate for these errors, a feedback control algorithm needs to be implemented. However, because of the nonholonomic nature of the system, feedback control is not so straightforward. A theorem by Brockett [23] states that asymptotic stabilization of WMRs is not achievable by continuous time-invariant state feedback. The intuitive explanation is that three states need to be controlled (stabilized) by using only 2 control inputs. For holonomic systems this is not the case; each element of the state can be controlled by its own corresponding input, i.e. there are as many inputs as controlled states. Hence, to stabilize the full state of a WMR, discontinuous and/or time-variant feedback is necessary. If one, however, only wants to stabilize the position and not the heading as well (as in the path following with no orientation control, [24]), control becomes much simpler.

For the case of INTERACT, feedback control needs to be developed for every possible drive mode (4WS, crab, rotate in place). The controllers are presented in this section. The controller for four wheel steering is the most complex. In this mode, heading control is not necessary but it might increase the accuracy of the final position. Hence, two different control methods are applied, one with and one without heading control. The former is a very simple technique which decouples steering and velocity control and applies two PID controllers. This controller works for both four wheel steering as well as crab mode. The latter is slightly more advanced, incorporating the heading control by applying approximate linearization around the reference trajectory. For rotation in place, a PID controller is used as well. A useful tool that is applied in the PID control method is introduced first. To simplify the expressions of errors it is common to express the state of the system in *path coordinates* (see e.g. [53], [28], [54]), which is a reference frame attached to the desired path. This frame is also called *Frénet frame* (see e.g. [24]).

A. Modelling in a Frénet Frame

A Frénet frame is a coordinate system that is related to the path or curve that needs to be followed. Let $\mathcal{C}$ be the desired trajectory consisting of x_r and y_r reference positions and reference heading θ_r. Now let $\mathcal{F}_s$ be the Frénet frame attached to the path. Because the reference path generated by the path planner has a time schedule, the origin of the Frénet frame is attached to the reference point at that particular time instant. The rotation of this frame with respect to the world is θ_r. The new position coordinates are the abscissa s, and d, the corresponding ordinate, perpendicular the direction defined by θ_r, as shown in Figure 46.

The position of the INTERACT system can now be expressed with respect to the Frénet frame. A detailed derivation of this transformation can be found in [24] and [55]. In the case of INTERACT, it can easily be verified that the transformation to the path coordinates is given by:

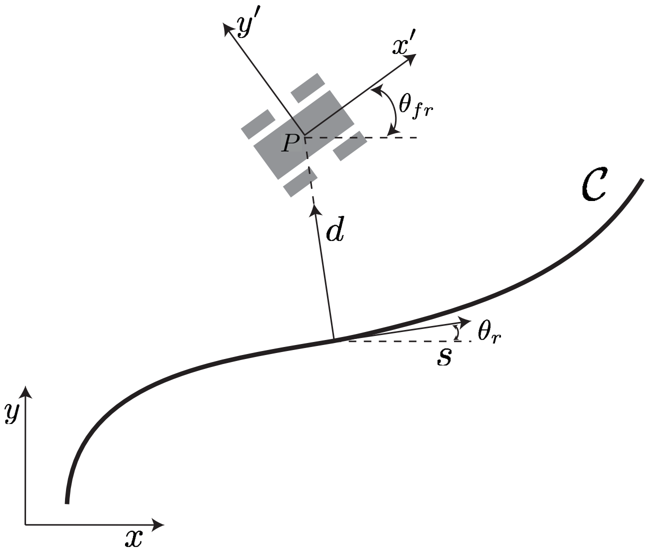


Fig. 46. Definition of the path coordinates or Frénet frame. The new coordinates are s and d, representing the distance travelled along the path $\mathcal{C}$ and the lateral deviation from it respectively.

$$\begin{bmatrix} s \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_r) & -\sin(\theta_r) & x_r \\ \sin(\theta_r) & \cos(\theta_r) & y_r \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{149}$$

B. PID Control

By using the system expressed in path coordinates, a separate PID controller for the steering and the velocity can be implemented. The reference trajectory $q_r = \begin{bmatrix} x_r & y_r & \theta_{fr} \end{bmatrix}^T$ which is generated by the RRT planning algorithm is converted to the path coordinates. Now, two errors can be defined. Because of the transformation towards path coordinates, the error related to the execution speed of the path is s itself, and the lateral error is equal to the coordinate d. The main benefit from a control point of view of expressing the system in path coordinates is that both s and d need to be regulated to 0.

The discrete time PID law used to compute the velocity and steering input of the system requires the discrete time derivative and integral of the error:

$$\begin{aligned} \frac{de(t_k)}{dt} &= \frac{e(t_k) - e(t_{k-1})}{\Delta t} \\ \int_0^{t_k} e(t)dt &= \sum_{i=1}^{k} e(t_i)\Delta t \end{aligned} \tag{150}$$

The input to the platform can then be computed by:

$$u_1 = -k_{p1}s - k_{i1}\sum_{i=1}^{k} s(t_i)\Delta t - k_{d1}\frac{s(t_k) - s(t_{k-1})}{\Delta t} \tag{151}$$

$$u_2 = -k_{p2}d - k_{i2}\sum_{i=1}^{k} d(t_i)\Delta t - k_{d2}\frac{d(t_k) - d(t_{k-1})}{\Delta t} \tag{152}$$

The controller gains are tuned using the simulated system. Several reference trajectories are tried and the gains are tuned until the tracking error is within the minimum turning radius of the platform (about 1 meter) and the controller output does not show high frequency oscillations. This results in the following values for k:

$$\begin{array}{lll} k_{p1}=3 & k_{i1}=0 & k_{d1}=0 \\ k_{p2}=3 & k_{i2}=0.5 & k_{d2}=0.1 \end{array}$$

The benefit of a PID controller is that it does not requires a model of the system. However, the method does not make use of the fact that a feedforward input is available, and heading control is not available. A more advanced method that does apply heading control for nonholonomic systems is control through approximate linearization of the system.

C. Approximate Linearization

To apply the method of approximate linearization as described in [28], the system needs to be put in so called *chained form*. It is a canonical form that is often used for controlling nonlinear systems because it shows a linear structure of the system equations (see e.g. [28], [25], [38]). The equations of motions that were found in Appendix II are slightly simplified before they are put in the chained form. The platform parameters a and b (see Figure 20) are assumed to be equal. Furthermore, the front steering angle is always equal but opposite to the rear steering angle, $\theta_f = -\theta_r$. The equations of motion for the INTERACT platform then become:

$$\begin{aligned} \dot{x} &= \eta_1 \cos(\theta_{fr}) \sin^2(\theta_w) && (153) \\ \dot{y} &= \eta_1 \sin(\theta_{fr}) \sin^2(\theta_w) && (154) \\ \dot{\theta}_{fr} &= \frac{\eta_1 \sin(2\theta_w)}{L} && (155) \\ \dot{\theta}_w &= \eta_2 && (156) \end{aligned}$$

Or equivalently:

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta}_{fr} \\ \dot{\theta}_w \end{bmatrix} &= \begin{bmatrix} \cos(\theta_{fr})\sin^2(\theta_w) \\ \sin(\theta_{fr})\sin^2(\theta_w) \\ \frac{\sin(2\theta_w)}{L} \\ 0 \end{bmatrix} \eta_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \eta_2 \\ \dot{\boldsymbol{q}}' &= \boldsymbol{g_1}\eta_1 + \boldsymbol{g_2}\eta_2 && (157) \end{aligned}$$

The general velocity vector $\dot{\boldsymbol{q}}$ is extended with the rate of change of the steering angle $\dot{\theta}_w$. This extended velocity vector is designated with $\dot{\boldsymbol{q}}'$. The first input to the extended system is η_1, which is equal to u_1: the platform velocity. η_2 is the rate of change of the steering angle, and hence relates to u_2 through:

$$\eta_2 = \dot{u}_2 \qquad (158)$$

The chained form can be achieved through an input and state transformation, which will transform the system into the following so called 4 state, 2 input chained form

$$\begin{aligned} \dot{z}_1 &= v_1 \\ \dot{z}_2 &= v_2 \\ \dot{z}_3 &= q_2' v_1 \\ \dot{z}_n &= q_3' v_1 \end{aligned} \qquad (159)$$

This two input case represents many kinematic models of wheeled mobile robots, such as the INTERACT platform. The linear structure becomes especially apparent when v_1 is assigned a predetermined function of time and is not considered as an actual control input. The system (159) then becomes a linear time-variant system.

A systematic way to transform a system to the (2,4) chained form can be found in literature (e.g. [26]). Necessary conditions to do this are given as well.

Now, according to [26], the system can be put into chained form if $\boldsymbol{g_1}$ and $\boldsymbol{g_2}$ are linearly independent and the following distributions are of constant rank and involutive:

$$\begin{aligned} \Delta_0 &= \text{span}\left\{\boldsymbol{g_1}, \boldsymbol{g_2}, \text{ad}_{\boldsymbol{g_1}}\boldsymbol{g_2}, ..., \text{ad}_{\boldsymbol{g_1}}^{n-2}\boldsymbol{g_1}\right\} && (160) \\ \Delta_1 &= \text{span}\left\{\boldsymbol{g_2}, \text{ad}_{\boldsymbol{g_1}}\boldsymbol{g_2}, ..., \text{ad}_{\boldsymbol{g_1}}^{n-2}\boldsymbol{g_2}\right\} && (161) \\ \Delta_2 &= \text{span}\left\{\boldsymbol{g_2}, \text{ad}_{\boldsymbol{g_1}}\boldsymbol{g_2}, ..., \text{ad}_{\boldsymbol{g_1}}^{n-3}\boldsymbol{g_2}\right\} && (162) \end{aligned}$$

Furthermore, there exists a function $h_1(q)$ such that:

$$dh_1 \cdot \Delta_1 = 0 \quad \text{and} \quad dh_1 \cdot \boldsymbol{g_1} = 1 \qquad (163)$$

If these conditions are met, a second function h_2 can be chosen such that

$$dh_2 \cdot \Delta_2 = 0 \qquad (164)$$

Once the functions h_1 and h_2 are found, the system can be put in chained form by the state transformation given by:

$$\begin{aligned} z_1 &= h_1 && (165) \\ z_2 &= L_{\boldsymbol{g_1}}^{n-2} h_2 && (166) \\ &\vdots && (167) \\ z_{n-1} &= L_{\boldsymbol{g_1}} h_2 && (168) \\ z_n &= h_2 && (169) \end{aligned}$$

and input transformation:

$$\begin{aligned} v_1 &= u_1 && (170) \\ v_2 &= (L_{\boldsymbol{g_1}}^{n-1} h_2) u_1 + (L_{\boldsymbol{g_2}} L_{\boldsymbol{g_1}}^{n-2} h_2) u_2 && (171) \end{aligned}$$

In these equations, L_f^k designates the Lie derivative:

$$\begin{aligned} L_f^0 h(x) &= h(x) && (172) \\ L_f^1 h(x) &= \frac{\partial h(x)}{\partial x} f(x) && (173) \\ &\vdots && (174) \\ L_f^k h(x) &= \frac{\partial}{\partial x}\left[L_f^{k-1} h(x)\right] f(x) && (175) \end{aligned}$$

and the $\text{ad}_f^k g(x)$ operator, which uses the *Lie bracket* $[f(x), g(x)]$, is defined by:

$$\begin{aligned} [f(x), g(x)] &= \frac{\partial g(x)}{\partial x} f(x) - \frac{\partial f(x)}{\partial x} g(x) && (176) \\ \text{ad}_f^0 g(x) &= g(x) && (177) \\ \text{ad}_f^1 g(x) &= [f(x), g(x)] && (178) \\ &\vdots && (179) \\ \text{ad}_f^0 g(x) &= \left[f(x), \text{ad}_f^{k-1} g(x)\right] && (180) \end{aligned}$$

This procedure can be applied to system (157) by first applying the transformation $v_1 = \eta_1 \cos(\theta_{fr}) \sin^2(\theta_w)$, which transforms $\boldsymbol{g_1}$ into:

$$\begin{bmatrix} 1 & \tan(\theta_{fr}) & \frac{\sin(2\theta_w)}{L\cos(\theta_{fr})\sin^2(\theta_w)} & 0 \end{bmatrix}^T \tag{181}$$

Then, the functions $h_1 = x$ and $h_2 = y$ can be chosen. It is easily verified that these functions satisfy the necessary conditions (163). This choice results in the following state and input transformations that give the single-chain form:

$$\begin{aligned} z_1 &= x \\ z_2 &= \frac{\sin(2\theta_w)\sec(\theta_{fr})^2}{L} \\ z_3 &= \tan(\theta_{fr}) \\ z_4 &= y \end{aligned} \tag{182}$$

$$v_1 = \eta_1 \cos(\theta_{fr}) \sin^2(\theta_w) \tag{183}$$

$$\begin{aligned} v_2 = {} & \frac{2\eta_1 \sin(\theta_{fr}) - 2\eta_1 \cos(2\theta_w)^2 \sin(\theta)}{L^2 \cos(\theta_{fr})^3} \\ & + \frac{2L\eta_2 \cos(2\theta_w)\cos(\theta_{fr})}{L^2\cos(\theta_{fr})^3} \end{aligned} \tag{184}$$

Now that the system is in chained form, the control by approximate linearization as presented in [28] can be applied. First, the state and input errors are defined as:

$$\tilde{z}_i = z_i - z_{ir} \tag{185}$$

$$\tilde{v}_j = v_j - v_{jr} \tag{186}$$

Where the subscript r denotes the reference value related to the path that needs to be followed. This reference value follows from the path planning stage.

The system's error equations become:

$$\begin{aligned} \dot{\tilde{z}}_1 &= \tilde{v}_1 \\ \dot{\tilde{z}}_2 &= \tilde{v}_2 \\ \dot{\tilde{z}}_3 &= z_2 v_1 - z_{2r} v_{1r} \\ \dot{\tilde{z}}_4 &= z_3 v_1 - z_{3r} v_{1r} \end{aligned} \tag{187}$$

Linearizing this system about the reference trajectory results in a time-varying linear system:

$$\begin{aligned} \dot{\tilde{z}} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & v_{1r} & 0 & 0 \\ 0 & 0 & v_{1r} & 0 \end{bmatrix} \tilde{z} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ z_{2r} & 0 \\ z_{3r} & 0 \end{bmatrix} \tilde{v} \\ \dot{\tilde{z}} &= A\tilde{z} + B\tilde{v} \end{aligned} \tag{188}$$

Now, regular control laws for time-varying linear systems can be used. The following feedback law is established:

$$\tilde{v}_1 = -k_1 \tilde{z}_1 \tag{189}$$

$$\tilde{v}_2 = -k_2 \tilde{z}_2 - \frac{k_3}{v_{1r}} \tilde{z}_3 - \frac{k_4}{v_{1r}^2} \tilde{z}_4 \tag{190}$$

The complete control input is then given by:

$$v = v_r + \tilde{v} \tag{191}$$

To find the actual input to the platform, input transformation (183) can be used to find η_1 and thus u_1. Furthermore, (184) can be inverted to find η_2. Together with (158), this gives u_2. However, the choice of (183) shows that the input transformation is only defined for the platform heading $\theta_{fr} \neq \pi/2 \pm k\pi$ with $k \in \mathbb{N}$. This means the controller will not work for platform headings close to $\pi/2 \pm k\pi$: the velocity will go to zero. To solve this, the controller is disabled and only the feedforward command is used for $\theta_{fr} = \pi/2 \pm 0.1$. Furthermore, note that the choice of (190) implies that $v_{1r} \neq 0$ and thus $u_{1r} \neq 0$.

Not all choices of k_i will results in a stable controller for every path that can be generated by the path planner. Hence, the choice of the gains k_i is very important. A short stability analysis similar to what is done in [56] can be performed, to find the gains that will stabilize the system. The open loop transfer function of the time-varying linear system becomes:

$$\begin{aligned} L(s) &= K(sI - A)^{-1}B && (192) \\ &= \begin{bmatrix} k_1/s & k_2/s + k_3/s^2 + k_4/s^3 \end{bmatrix} && (193) \end{aligned}$$

with $K = \begin{bmatrix} k_1 & k_2 & k_3/v_{1r} & k_4/v_{1r}^2 \end{bmatrix}$.

Now the stability of the system is determined by the denominator $T(s)$ of the closed loop transfer function:

$$\begin{aligned} T(s) &= 1 + L(s) && (194) \\ T(s) &= \begin{bmatrix} k_1/s + 1 & k_2/s + k_3/s^2 + k_4/s^3 + 1 \end{bmatrix} && (195) \\ & && (196) \end{aligned}$$

This gives two separate polynomials, one for k_1 and one for k_2, k_3 and k_4. Now the Hurwitz stability theorem can be applied to both. This theorem states that the system is stable iff all sub-determinants of the Hurwitz matrix are positive. The Hurwitz matrix can be constructed from the coefficients of the Hurwitz polynomials given by (194):

$$H_1 = \begin{bmatrix} k_1 & 0 \\ 0 & 1 \end{bmatrix} \tag{197}$$

$$H_2 = \begin{bmatrix} k_2 & 1 & 0 \\ k_4 & k_3 & k_2 \\ 0 & 0 & k_4 \end{bmatrix} \tag{198}$$

This results in the following constraints for the choice of the gains:

$$\begin{aligned} k_1 &> 0 \\ k_2 &> 0 \\ k_2 k_3 - k_4 &> 0 \\ k_2 k_3 k_4 - k_4^2 &> 0 \end{aligned} \tag{199}$$

These constraints leave many choices for the gains. In [28] it is suggested to choose the gains such that four coincident closed-loop eigenvalues at -5 are established and a damping coefficient of $\zeta = 1$. This results in:

$$\begin{aligned} k_1 &= 5 \\ k_2 &= 15 \\ k_3 &= 75 \\ k_4 &= 125 \end{aligned}$$

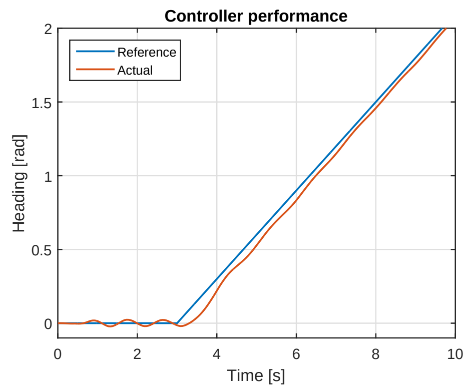


Fig. 47. Response of the controller for rotate in place on a slope. The reference heading is given in blue. The actual heading (measured by the gyroscope) is given in orange.

D. Feedback Control for Rotate in Place

Feedback control of the rotation in place drive mode is more simple than the other modes, as only one input is available: the velocity command. A PID controller is defined which uses θ_e as error input. This is the difference between the reference heading θ_r and the actual heading estimated by the localization algorithm (i.e. the integrated gyroscope measurements θ_{fr})

$$\theta_e = \theta_r - \theta_{fr} \tag{200}$$

and the PID feedback law becomes

$$u_1 = -k_{p3}\theta_e - k_{i3}\sum_{i=1}^{k}\theta_e(t_i)\Delta t - k_{d3}\frac{\theta_e(t_k) - \theta_e(t_{k-1})}{\Delta t} \tag{201}$$

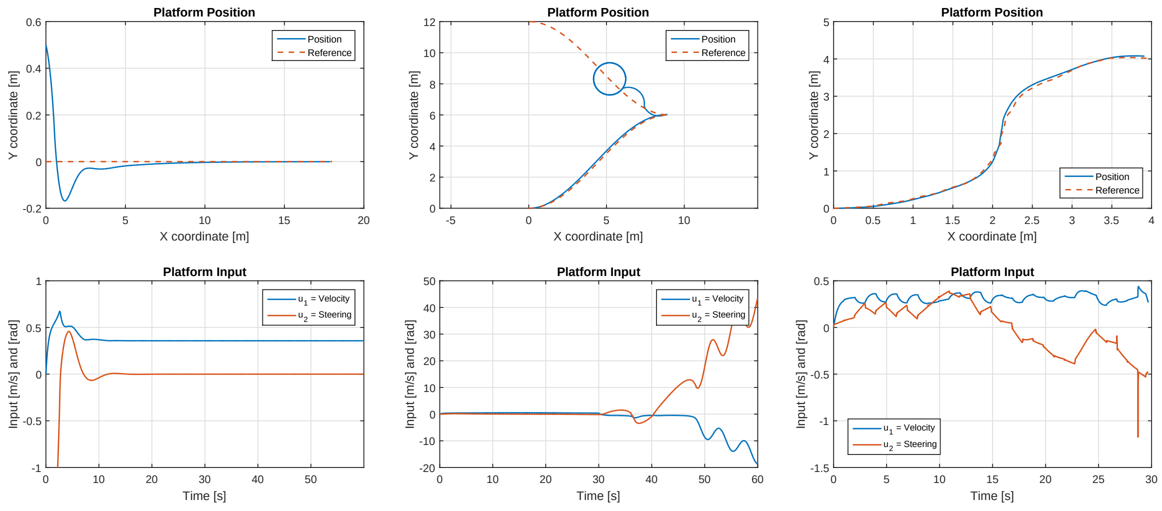
The gains are tuned based on the response of the system on a reference slope as shown in Figure 47 which is representative for the rate of change in heading expected during navigation. They are chosen such that the tracking error never exceeds 0.05 radians after four seconds. When the controller is used during reference trajectory tracking, it is always given this time to converge to the reference point. The oscillation during the start is caused by the fact that the platform will only move when the velocity input exceeds 0.3 m/s. This is likely caused by friction in the mechanical system and wheel-to-ground interaction forces. The limitation is incorporated in the controller by adding a viscous friction block in Simulink. However, as a result of this viscous friction, a very small error at the start of the reference will make the controller compensate with an input of 0.3 m/s minimum, which causes the oscillation. The amplitude of the oscillation is within the 0.05 radian limit and hence acceptable. The gains for the PID controller are:

$$k_{p3} = 5 \quad k_{i3} = 0.7 \quad k_{d3} = 0$$

E. Simulation Results

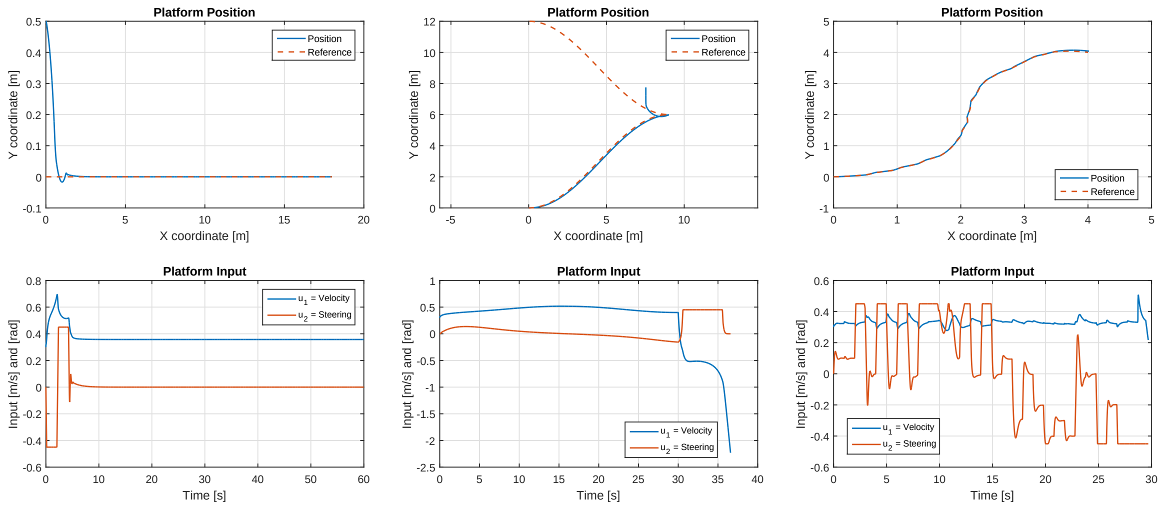
Both the PID controller as well as the controller based on approximate linearization are simulated using the mathematical model of the platform described before. The reference trajectory is manually generated for a straight reference along the x axis and for a maneuver containing a reversal (parallel parking). A more complex trajectory is generated using the RRT planner explained in Appendix VII. The bidirectional mode is used here to create a discontinuity roughly halfway the path. This is to assess whether the controller can handle such discontinuities, which can arise by using the bidirectional planning mode and/or path smoothing. The results of the simulations are given in Figure 48 for the PID controller and in Figure 49 for the approximate linearization. In each set of figures, the reference is given by a dashed orange line, while the simulated platform response is given by the solid blue line. In the lower graphs, the controller output (which is the platform input) is given.

It is evident that the tracking accuracy for the controller based on approximate linearization is generally better. It shows less overshoot and a faster settling time. Both controllers are not able to handle the reversal in the second path, but can cope with the discontinuity in the third path.



(a) Straight line reference (b) Reversal maneuver (c) RRT plan with discontinuity

Fig. 48. Results of simulations applying platform control using the PID controller. The top figure show the reference trajectory (red) along with the actual platform location (blue). Bottom figures show the input computed by the controller.



(a) Straight line reference (b) Reversal maneuver (c) RRT plan with discontinuity

Fig. 49. Results of simulations applying platform control using the controller based on approximate linearization. The top figure show the reference trajectory (red) along with the actual platform location (blue). Bottom figures show the inpute computed by the controller.

Appendix IX
Experimental Evaluation of the Navigation Algorithm

A. Experimental Setup

The feedback controllers are tested along with the full testing of the autonomous navigation algorithm as described in these appendices. The algorithm is implemented on the INTERACT platform and experimentally tested. The workspace used during the experiments was roughly 10 by 6 meter. 9 Vicon cameras were positioned along the perimeter of the workspace. The platform and the taskboard were positioned as far as possible from each other within this workspace. An image of the workspace and set-up is shown in Figure 51. Objects could be added to the planning algorithm, to make the task more difficult and increase path length. The planning was done off-line and separated into the coarse approach and alignment stage discussed before. During the first stage, the localization is based on the wheel encoders and IMU. During the alignment stage, the localization is improved by the detection of the taskboard. However, as this is not implemented yet, it is simulated by using the Vicon measurement as a location estimate during the alignment stage. The resulting reference trajectory and input from the planning were uploaded to the platform. The complete path from initial position to taskboard was in the order of 20 meter.

The reference point used for the Vicon system should be as close as possible to the reference point P used for the mathematical model and planning algorithm. If these do not match, the Vicon ground truth will show a deviation from the reference path, although the platform might be following it exactly. The problem is, however, that due to the bodywork and components on the platform, the exact reference point P is not reachable, and hence no markers can be attached. As the height of the reference does not matter since the navigation problem is considered in 2D only, the reference point P could be projected on the ground. However, this location is not visible to the Vicon cameras, because the robot itself is covering it. As a solution, markers are attached to the head of the robot which define the head reference frame H as in Figure 50. The calibration wand of the Vicon system is placed under the rover at the position of the projection of P on the ground. The location of the head T_W^H in the world frame W (as measured by the Vicon system) is recorded. Then, the platform is driven away and the wand becomes visible to the Vicon system. This makes measurement of the location of the reference point P in the world frame T_W^P possible. The transformation T_H^P between the head position in the world frame and P in the world frame can now be found through:

$$T_H^P = T_H^W T_W^P = \left(T_W^H\right)^{-1} T_W^P \tag{202}$$

When the experiments are executed, the head position is measured by the Vicon system, and transformed to the reference point P by using T_H^P

$$T_W^P = T_W^H T_H^P \tag{203}$$

The path planning requires specific settings for the *coarse approach* and the *alignment*. These settings follow from the simulations described in Appendix VII and are listed below. The gains for the controllers were as reported in Appendix VIII and identical for both stages.

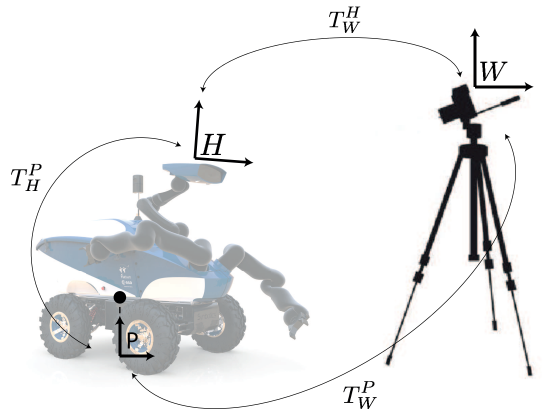


Fig. 50. Definition of the frames and transformations used to be able to measure the position and heading of the reference point P in the world frame.

1) ***Coarse Approach Settings:*** For this stage, the single tree version of the RRT algorithm is used with a goal bias of 0.05. The similarity threshold ϵ is set to the distance at which the taskboard can be recognized, which is 5 meter. The position and heading gain for the distance function (22) are set to $k_{pos} = 1$ and $k_\theta = 2$. The integration time $\Delta t = 2$ and a total of 7 motion primitives are selected. As no accurate maneuvering is necessary, it was decided to not include rotation in place motions as to not excessively switch between control modes.

$$\begin{bmatrix} u_1 \\ u_2 \\ \text{DM} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.2 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.3 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.4 \\ 48 \end{bmatrix}$$

2) ***Alignment Settings:*** For the alignment, the bidirectional version is used. This was done because this stage needs to result in an exact alignment of the platform in front of the taskboard. As stated before, it is virtually impossible to exactly reach the goal location due to the nonholonomic constraints of the platform. The goal is only reached with an accuracy determined by the similarity threshold ϵ. Choosing the bidirectional algorithm will relocate this problem to the point where the trees meet (i.e. in the middle of the path). Although this causes a discontinuity, the end of the path is now exactly the goal position. The similarity threshold is set to $\epsilon = 0.1$ to keep the discontinuity in the path small, such that the controller described in Section VI can handle it. The position and heading gain for the distance function are unchanged. $\Delta t = 1$ and a total of 9 motion primitives are selected. Both crab mode and rotation in place are included in the set. The turn in place input allows rotation in both CW and CCW direction.

$$\begin{bmatrix} u_1 \\ u_2 \\ \text{DM} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.2 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.25 \\ 32 \end{bmatrix}, \begin{bmatrix} 0.5 \\ \pm 0.4 \\ 48 \end{bmatrix}, \begin{bmatrix} \pm 0.5 \\ 0 \\ 64 \end{bmatrix}$$

During the *coarse approach*, the controller performance is separated from the localization performance by using two different error metrics. The *Localization error* is defined

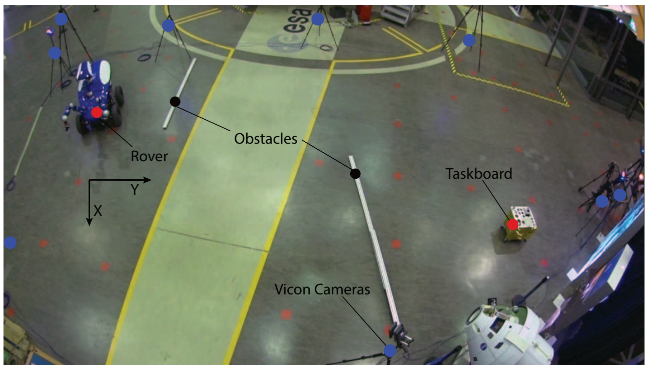


Fig. 51. Image of the workspace used during the navigation experiments. The rover and taskboard are both indicated with a red dot. Beams used as obstacles are indicated with black dots. 9 Vicon cameras, indicated in blue, are positioned around the perimeter of the workspace.

TABLE XI
RESULTS OF ALIGNMENT STAGE. BOLD ENTRIES ARE NOT WITHIN THE SPECIFIED ACCURACY OF 15 CM IN POSITION AND 6° IN HEADING

	Absolute Total Errors			
Trial	x [m]	y [m]	θ_{fr} [°]	Within limits
1	**0.162**	0.079	0.018	no
2	0.114	**0.212**	**7.523**	no
3	**0.152**	0.046	1.169	no
4	0.060	**0.359**	3.197	no
5	0.006	0.050	4.165	yes
6	**0.426**	**0.159**	5.139	no
7	0.112	0.098	2.236	yes
8	0.140	0.016	**11.61**	no
9	0.045	**0.152**	2.022	no
10	**0.221**	0.135	**9.848**	no
Average	0.140	0.132	4.693	yes

as the difference between the localization position estimate and the actual position as measured by the Vicon motion capture system. This error represents the performance of the localization algorithm, and is unknown to the system. The *tracking error* is defined by the difference between the localization estimate and the reference trajectory and hence represents the controller performance. If the controller tracks the path perfectly, it is zero. This error is known to the system and hence the execution can be terminated if the error gets too large (i.e. exceeds the minimum turning radius).

The performance of the navigational algorithm as a whole is assessed using the *total error*. This error is computed by taking the difference between the actual location of the platform according to the Vicon system and the desired location given by the path planning algorithm. The final value of this error after execution of the *alignment* stage should be within the limits of 15 cm in position and 6° in heading.

Other important variables are the drive mode, which determines which controller is used to track the path, and the input to the platform computed by the controller.

During the experiments, 8 trials of the *coarse approach* and 10 trials of the *alignment* are executed. The initial position for the *coarse approach* is held constant to within a couple of centimeters and degrees. The path found by the planning algorithm is different for every trial but the configuration of the obstacles and taskboard is identical. An example path found by the planner is shown in Figure 15. The location of the platform at the start of the *alignment* stage is varied, to test if accurate alignment is possible from several angles and positions w.r.t. the taskboard.

B. Results

The error metrics for the trials of both stages are listed in Table XII and Table XI. For the alignment stage, only the absolute total error after execution is reported. These values must be within te limits specified. The last column of the table shows if the final position was within these limits. For the coarse approach, all three error metrics are reported.

Several plots of the error metrics and a position comparison are given as well. The first set of graphs in Figure 52 is from the coarse approach trial. In the top left graph of 52a, the position of the platform from the Vicon system (blue) is compared to the localization estimate (yellow) and the reference trajectory (orange). The remaining figures show the *tracking* and *localization* error, and finally the combined *total error*. In each graph, the error is subdivided into the error in x position (blue), y position (orange) and heading θ (yellow). The set of graphs in Figure 52b shows the control input and the drive model respectively.

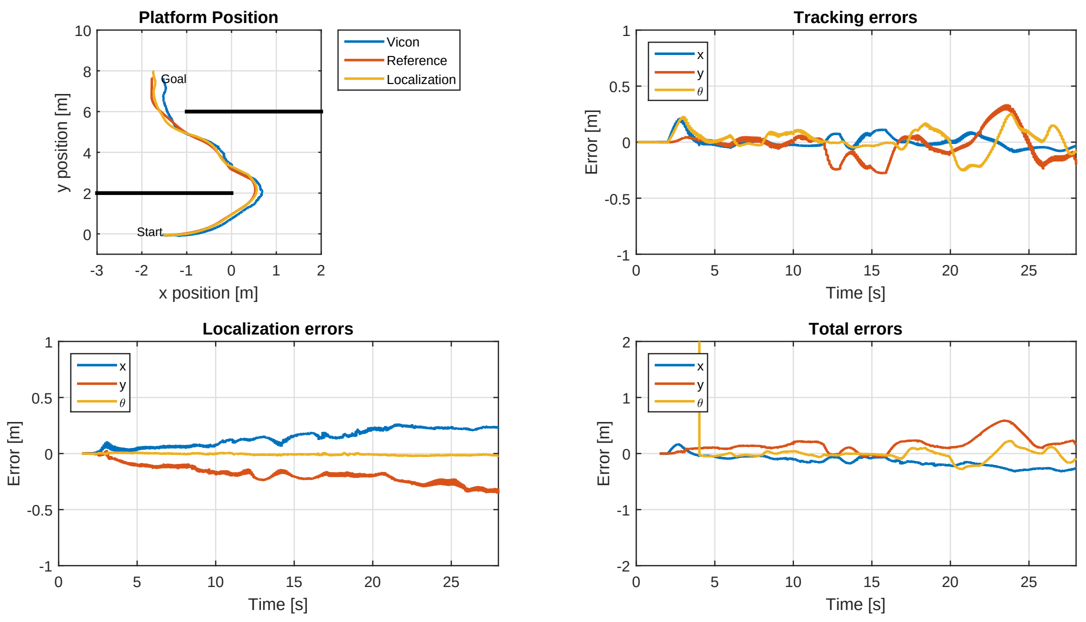
The tracking of the reference trajectory is always within 0.3 meter. The error when path execution is finished is around 0.1 meter. The localization accuracy, however, is diverging quite rapidly, with a final error of about 0.3 meter in both x and y.

Figure 53 shows the performance of the navigation algorithm for the alignment stage. During this experiment, the localization based on wheel encoders is replace by the Vicon position measurement to simulate the recognition and feedback of the taskboard and its pose. The tracking shows a large deviation in the middle segment of the path, which coincides with a jump in the heading error as can be seen in the top right figure. The tracking error is mostly compensated for during the remainder of the path, but an error in x of about 0.4 meter is still present after execution is done. The results in the total error being outside the required bounds for performing the manipulation tasks at the taskboard.

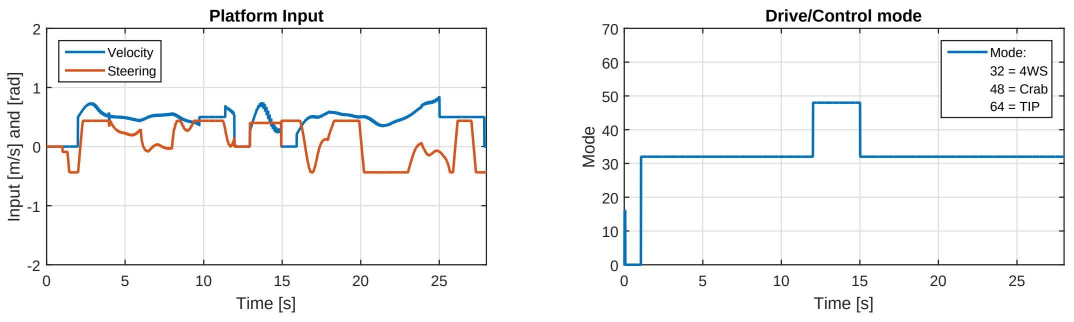
The final set of results is shown in Figure 54. This is a special case of the final alignment stage where the platform is placed with a lateral offset from the platform but with the right final heading. This is similar to a parallel parking maneuver except that this platform (or actually the controller) is not capable of a reversal. Hence, the resulting path requires extensive use of the rotation in place mode. The tracking error has the largest magnitude (about 0.5 meter) around half way. The final error is in the order of 5 cm. This is well within the bounds required for manipulation without relocation.

TABLE XII
RESULTS OF COARSE APPROACH STAGE

Trial	Abs. Tracking Errors			Abs. Localization Errors			Abs. Total Errors		
	x [m]	y [m]	θ_{fr} [°]	x [m]	y [m]	θ_{fr} [°]	x [m]	y [m]	θ_{fr} [°]
1	0.384	0.302	2.550	0.082	0.574	0.745	0.285	0.153	1.037
2	0.113	0.134	0.544	0.010	0.443	0.476	0.117	0.301	3.673
3	0.155	0.265	9.546	0.332	0.424	0.974	0.164	0.143	7.735
4	0.185	0.055	2.074	0.009	0.415	1.089	0.194	0.359	0.712
5	0.133	0.126	2.922	0.057	0.358	0.854	0.074	0.235	1.283
6	0.165	0.434	21.76	0.020	0.545	1.874	0.182	0.128	19.35
7	0.093	0.267	2.487	0.205	0.395	1.742	0.297	0.674	5.764
8	0.102	0.655	0.355	0.278	0.418	2.521	0.177	1.073	2.246
Average	0.166	0.280	5.280	0.124	0.446	1.284	0.186	0.383	5.223



(a) Errors



(b) Control Input

Fig. 52. Experimental results of the first planning and execution stage. The top left graph in 52a contains the platform position comparison between Vicon (blue), localization (yellow) and the reference (orange). The other three graphs contain the errors in x (blue), y (orange) and θ (yellow). 52b includes the platform input computed by the controller and the drive/control mode determined by the path planning algorithm. In this stage, only four wheel steer and crab mode are used.

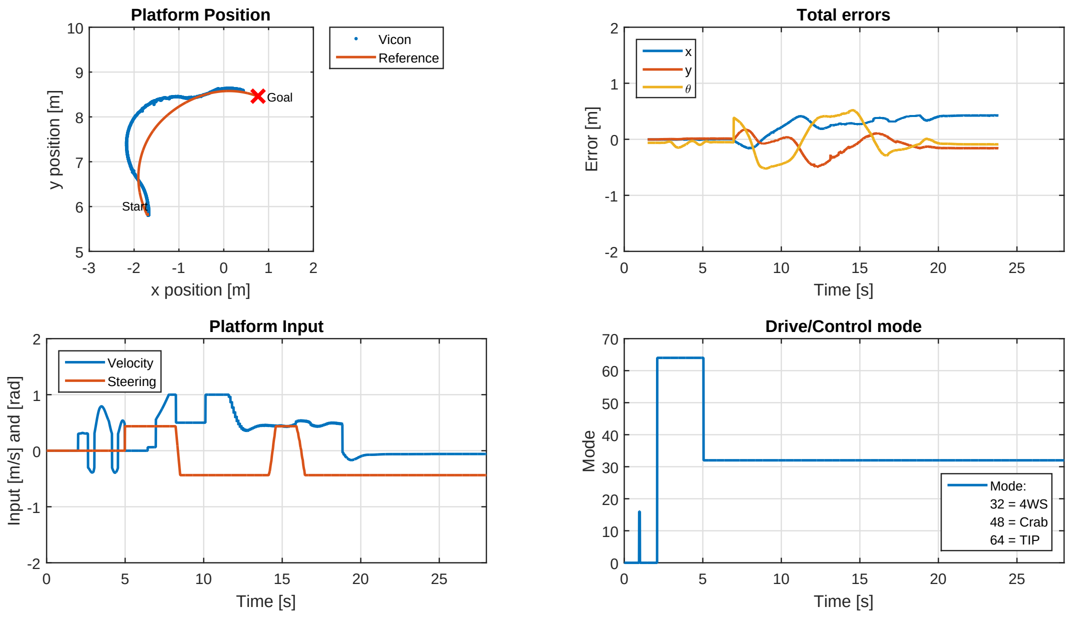


Fig. 53. Experimental results of the alignment stage of the navigation algorithm. The top left figure contains the platform position reference (orange) and the position according to the Vicon system (blue), which is also the localization estimate during this stage. The top right graph shows the *total error*. The lower two graphs show the input tot the platform computed by the controller and the drive/control mode respectively.

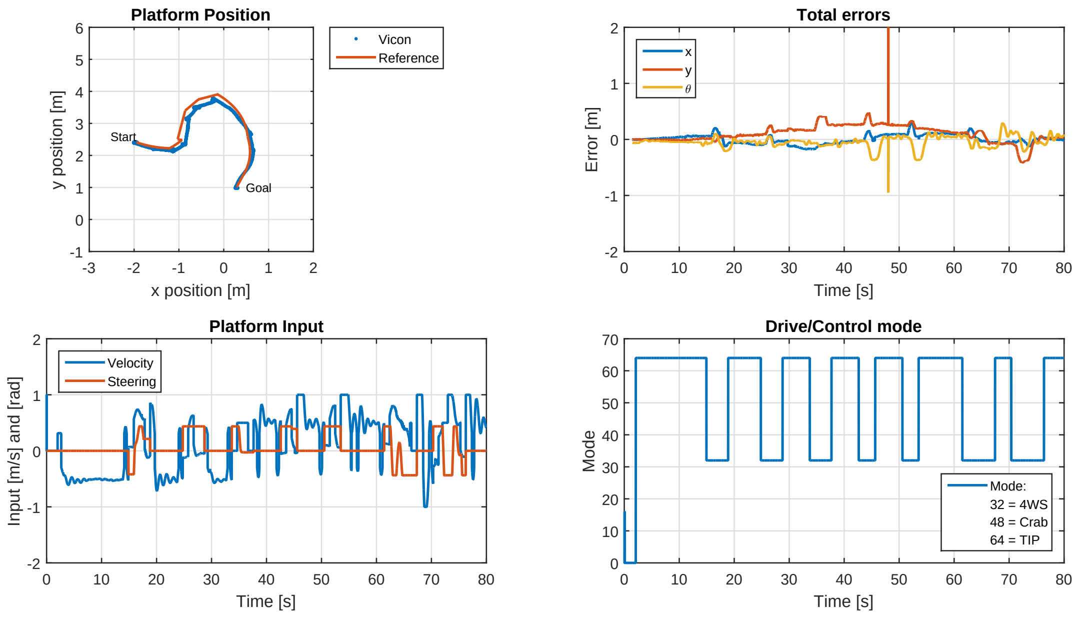


Fig. 54. Experimental results of the alignment stage where a ”parallel parking” maneuver needs to be executed. The content of the graphs is the same as in Figure 17. Note that reversals are not possible and hence the rotation in place mode needs to be used extensively.

Appendix X
Discussion

This section provides a discussion of the work done during the Msc. project. As the goal of the project was to develop and implement a working navigation algorithm, but not necessarily an optimal one, there are numerous improvements possible in various aspect of the algorithm. This section gives an overview of these improvements and references if available. Furthermore, it will list the situations that were not (thoroughly) tested during the Msc. project and that require more attention and/or future research.

A. Localization

The localization algorithm based on dead-reckoning has been shown to provide a position estimate within 0.5% of the travelled distance. However, this was only tested on very smooth terrain (no slopes) with high traction (no sandy or slippery surface). The performance of the localization will most likely deteriorate when the platform is driving on more difficult terrain. For the Msc. project it was not necessary to test the localization on very rough terrain. Besides, a suitable location that was large enough, indoors, and had the necessary equipment (such as Vicon motion capture system) available, could not be found.

If the platform, in its current state, needs to perform on more rough terrain, the following improvements could be implemented:

- Take the whole attitude of the rover into account while performing dead-reckoning (See for example: [57] [58] [59]), i.e. move towards 3D. This will increase the dead-reckoning accuracy when the environment contains slopes.
- Use accelerometer data to detect all wheel slip. This will give the possibility to reduce localization errors when all wheels are slipping, as the wheel encoder count can be ignored when the accelerometer does not show movement [60], [61], [14].
- Add Ackermann steering to the platform. This is a hardware improvement that would reduce localization errors. If the platform is capable of Ackermann steering, wheel slip would be reduced [44].

Additionally, it would be very beneficial to add a sensor to the platform that would give an absolute position measurement. This could be a stereo camera pair or a LIDAR. Not only would this greatly increase the accuracy of the localization, it would also add the capability of obstacle detection.

B. Path Planning

The path planning algorithm based on RRT is capable of planning a path using all drive modes of the platform. The RRT algorithm is generally used to find a "good enough" path, which is not optimized for a certain cost function (based on e.g. length or control effort). However, the distance function $\rho(q_1, q_2)$ used in the algorithm can still be tuned to influence the resulting path and adapt it to the needs of the specific situation. For example, the cost function could be defined such that the nonholonomic constraints are taken into account. This means that the same distance in the Euclidean sense would be valued higher if it is a lateral offset as compared to longitudinal. This can improve the accuracy of reaching the goal vertex q_{goal} and help match the vertices if the bidirectional version or path smoothing is used.

The algorithm parameter Δt influences the length of the motion primitives and how often the platform can change its drive mode. In wide workspaces containing few obstacles, it is beneficial to choose a high value for this parameter. It will make the planning faster, as the space will be traversed quicker using less vertices, and the behaviour of the platform will be more smooth as it switches less between drive modes. In more confined spaces, where many obstacles are present, it is necessary to choose Δt low to be able to find a solution. An unwanted side effect is that the platform can switch the drive mode every time a new motion primitive is selected. This does not benefit the localization and tracking accuracy. One solution could be to penalize the switch of drive modes, such that the behaviour becomes more smooth. Another solution could be to make Δt variable, and let the algorithm reduce its value if the specific environment requires this.

Finally, the same argumentation could be used for the number of motion primitives available to the planning algorithm. If the workspace is large and relatively free of obstacles, only a few motion primitives are necessary to find a path. If the platform is required to be very versatile, more motion primitives are necessary. If this number could be set dynamically during path planning, this would benefit the performance of the algorithm.

C. Control

Three different controllers, one for each drive mode, were implemented on the platform. The final experiments using the navigation algorithm show the performance of these controllers. Despite the difficulty of controlling nonholonomic systems, the controllers are able to keep the tracking error well within the minimum turning radius. Nevertheless, several improvements might be possible to increase the tracking accuracy, and make the system more robust to external disturbances.

Currently, the main limitation of the controllers is that they are not able to handle reversals of the platform. This greatly limits its versatility and results in more complicated paths than necessary. In [28], several techniques are presented that should solve this problem.

A hardware limitation of the platform is that velocity inputs lower than 0.3 m/s will not result in any motion. Apparently this input is too low for the internal controllers of the platform to overcome friction in the mechanical system. As a result, the platform has to drive at relatively high speeds. Especially at the end of the path, when a certain position needs to be reached, this results in overshoot.

The controller for the crab drive mode is currently a simple PID one. Although the controller is able to track the reference, performance is not as good as with the controller based on approximate linearization and the feed-forward command is not used. Furthermore, because the PID feedback law for the steering is based on an error signal that is not directly controlled by the steering angle (a forward velocity is always needed), the control is not ideal. By modifying the mathematical equations of the system, it should be possible to also apply the control through approximate linearization on the crab mode. This is expected to increase the tracking performance.

Regarding the choice of the gains k_i of the approximate linearization method, the following needs to be considered.

Tuning the gains to give better tracking could be possible. However, because the steering and velocity input becomes more aggressive and rapidly changing, this will deteriorate the performance of the localization estimate based on dead-reckoning. A balance needs to be found between tracking accuracy and localization accuracy. If the localization algorithm is not improved by the addition of an extra sensor providing an (absolute) position measurement, the controller should be tuned to give acceptable tracking while the localization estimate is not affected.

From the control effort that was necessary during the experiments, it seems that the controller needs to give quite some correction on the feedforward command. Furthermore, the steering input is often saturated at the bounding values, both positive and negative. This could suggest that the model used in the planner does not represent reality particularly well, or the control gains could be tuned more to reduce the control effort. This should be investigated.

Finally, the path planning algorithm results in a trajectory with a time schedule. The reference point for the controllers is hence the point on the trajectory which has the current time stamp. Although this makes it very straightforward to use feedforward control, it has some drawbacks. If the platform somehow accumulates a delay w.r.t. the path (e.g. because of a large external disturbance), the controller needs great effort to compensate for this, which results in extreme controller outputs for both velocity and steering. This is again not beneficial for the localization estimate, as wheel slip becomes more probable. Instead, the closest point on the path to the platform could be chosen as the reference point. This might make the path tracking more smooth and one does not have to worry about accumulating delays.

References

[1] A. Schiele, T. Krueger, J. Smisek, E. Mattheson, J. Rebelo, D. E. E., F. Van der Hulst, and S. Kimmer, "Towards the Interact Space Experiment: Controlling an Outdoor Robot on Earth's Surface from Space," *Proc. 13th ASTRA Conference on Space Robotics*, 2015.

[2] N. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An Obstacle-based PRM for 3D Workspaces," *Proc. Third Work. Algorithmic Found. Robot. Robot. Algorithmic Perspect. Algorithmic Perspect.*, pp. 155–168, 1998.

[3] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," *Proc. 2000 ICRA. Millenn. Conf. IEEE Int. Conf. Robot. Autom. Symp. Proc. (Cat. No.00CH37065)*, vol. 1, no. April, 2000.

[4] S. Wilmarth, N. Amato, and P. Stiller, "MAPRM: a probabilistic roadmap planner with sampling on the medial axis of the free space," *Proc. 1999 IEEE Int. Conf. Robot. Autom. (Cat. No.99CH36288C)*, vol. 2, no. May, pp. 1024–1031, 1999.

[5] J. Kuffner, J.J. and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," *Proc. 2000 ICRA. Millenn. Conf. IEEE Int. Conf. Robot. Autom. Symp. Proc. (Cat. No.00CH37065)*, vol. 2, no. April, pp. 995–1001, 2000.

[6] S. LaValle and Kuffner, "Rapidly-Exploring Random trees: Progress and Prospects," 2000.

[7] S. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

[8] J. Barraquand and J. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *Int. J. Rob. Res.*, vol. 10, no. 6, pp. 628–649, 1991.

[9] B. Mirtich and J. Canny, "Using skeletons for nonholonomic path planning among obstacles," *Proc. 1992 IEEE Int. Conf. Robot. Autom.*, 1992.

[10] S. LaValle, "Planning Algorithms," *Methods*, vol. 2006, p. 842, 2006. [Online]. Available: http://ebooks.cambridge.org/ref/id/CBO9780511546877

[11] E. Baumgartner, H. Aghazarian, a. Trebi-Ollennu, T. L. Huntsberger, and M. S. Garrett, "Rover Localization Results for the FIDO Rover," *Spie*, vol. 4571, pp. 34–44, 2001.

[12] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe, "Mobile robot positioning: Sensors and techniques," *J. Robot. Syst.*, vol. 14, no. 4, pp. 231–249, 1997.

[13] L. Ojeda and J. Borenstein, "FLEXnav: fuzzy logic expert rule-based position estimation for mobile robots on rugged terrain," *Proc. 2002 IEEE Int. Conf. Robot. Autom. (Cat. No.02CH37292)*, vol. 1, no. May, 2002.

[14] L. Ojeda, G. Reina, and J. Borenstein, "Experimental results from FLEXnav: An expert rule-based dead-reckoning system for Mars rovers," *IEEE Aerosp. Conf. Proc.*, vol. 2, pp. 816–825, 2004.

[15] G. Dissanayake, S. Sukkarieh, E. Nebot, and H. F. Durrant-Whyte, "The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications," *IEEE Trans. Robot. Autom.*, vol. 17, no. 5, pp. 731–747, 2001.

[16] H. F. Durrant-Whyte, "An Autonomous Guided Vehicle for Cargo Handling Applications," *Int. J. Rob. Res.*, vol. 15, pp. 407–440, 1996.

[17] J. Borenstein and L. Feng, "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," *IEEE Trans. Robot.*, vol. 12, no. 6, 1996.

[18] K. S. Chong and L. Kleeman, "Accurate odometry and error modelling for a mobile robot," *Proc. Int. Conf. Robot. Autom.*, vol. 4, no. April, pp. 2783–2788, 1997.

[19] A. Martinelli, "The accuracy on the parameter estimation of an odometry system of a mobile robot," *Proc. 2002 IEEE Int. Conf. Robot. Autom. (Cat. No.02CH37292)*, vol. 2, no. May, pp. 1378–1383, 2002.

[20] J. Borenstein, "Experimental Results from Internal Odometry Error Correction with the OmniMate Mobile Robot," *IEEE Trans. Robot.*, vol. 14, no. 6, pp. 963–969, 1998.

[21] L. Ojeda, G. Reina, D. Cruz, and J. Borenstein, "The FLEXnav precision dead-reckoning system," *Int. J. Veh. Auton. Syst.*, vol. 4, no. 2/3/4, p. 173, 2006.

[22] C. C. Ward and K. Iagnemma, "Classification-based wheel slip detection and detector fusion for mobile robots on outdoor terrain," *IEEE Int. Conf. Robot. Autom.*, vol. 26, no. April, pp. 33–46, 2007.

[23] R. W. Brockett, "Asymptotic stability and feedback stabilization," *Differ. Geom. Control Theory*, pp. 181–191, 1983. [Online]. Available: http://hrl.harvard.edu/publications/brockett83asymptotic.pdf

[24] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[25] R. Murray and S. Sastry, "Steering nonholonomic systems in chained form," *[1991] Proc. 30th IEEE Conf. Decis. Control*, no. December, pp. 1121–1126, 1991.

[26] R. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, 1994, vol. 29.

[27] P. Morin and C. Samson, "Time-varying exponential stabilization of chained form systems based on a backstepping technique," *Proc. 35th IEEE Conf. Decis. Control*, vol. 2, no. December 1996, 1996.

[28] A. DeLuca, G. Oriolo, and C. Samson, *Feedback control of a nonholonomic car-like robot*, 1998. [Online]. Available: http://link.springer.com/content/pdf/10.1007/BFb0036073.pdf

[29] B. D'Andréa-Novel and G. Campion, "Dynamic feedback linearization," *29th IEEE Conf. Decis. Control*, pp. 1–6, 1992.

[30] G. Oriolo, A. DeLuca, and M. Vendittelli, "WMR control via dynamic feedback linearization: Design, implementation, and experimental validation," *IEEE Trans. Control Syst. Technol.*, vol. 10, no. 6, pp. 835–852, 2002.

[31] D. Tilbury and Y. France, "Steering a Three-Input Nonholonomic System using Multi-rate Controls," *Proc. Eur. Control Conf.*, pp. 1–4, 1993.

[32] R. Murray and S. Sastry, "Nonholonomic motion planning. Steering using sinusoids," *IEEE Trans. Automat. Contr.*, vol. 38, no. 5, pp. 700–716, 1993.

[33] P. Morin and C. Samson, "Control of Nonholonomic Mobile Robots Based on the Transverse Function Approach," *IEEE Trans. Robot.*, vol. 25, no. 5, pp. 1058–1073, 2009.

[34] K. N. Spentzas, I. Alkhazali, and M. Demic, "Kinematics of four-wheel-steering vehicles," vol. 66, 2001.

[35] G. Campion, B. D'Andréa-Novel, and G. Bastin, "Modelling and state feedback control of nonholonomic mechanical systems," *IEEE Conf. Decis. Control*, pp. 1184–1189, 1991. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=261553

[36] K. Bohlmann, H. Marks, and A. Zell, "Automated Odometry Self-Calibration for Car-Like Robots with Four-Wheel-Steering," pp. 6–11.

[37] J. Borenstein and L. Feng, "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," *IEEE Trans. Robot.*, vol. 12, no. 6, pp. 869–880, 1996.

[38] C. Samson, "Control of chained systems application to path following and time-varying point-stabilization of mobile robots," *IEEE Trans. Automat. Contr.*, vol. 40, no. 1, pp. 64–77, 1995.

[39] P. Morin and C. Samson, "Control of nonlinear chained systems: from the Routh-Hurwitz stability criterion to time-varying exponential stabilizers," *IEEE Trans. Automat. Contr.*, vol. 45, no. 1, pp. 141–146, 2000.

[40] E. N. Moret, "Dynamic Modeling and Control of a Car-Like Robot," 2003.

[41] A. DeLuca and M. Benedetto, "Control of nonholonomic systems via dynamic compensation," *Kybernetika*, vol. 29, no. 6, pp. 593–608, 1993.

[42] P. R. Giordano, M. Fuchs, and G. Hirzinger, "On the Kinematic Modeling and Control of a Mobile Platform Equipped with Steering Wheels and Movable Legs."

[43] G. Campion, G. Bastin, and B. D'Andréa-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," pp. 47–62, 1996.

[44] J. Ackermann, "Robust decoupling, ideal steering dynamics and yaw stabilization of 4WS cars," *Automatica*, vol. 30, no. 11, pp. 1761–1768, 1994.

[45] L. Ojeda and J. Borenstein, "Methods for the reduction of odometry errors in over-constrained mobile robots," *Auton. Robots*, vol. 16, no. 3, pp. 273–286, 2004.

[46] J. Borenstein, "The CLAPPER : A Dual-drive Mobile Robot With Internal Correction of Dead-reckoning Errors," pp. 3085–3090, 1994.

[47] M. W. Powell, T. Crockett, J. M. Fox, J. C. Joswig, J. S. Norris, K. J. Rabe, M. McCurdy, and G. Pyrzak, "Targeting and localization for Mars rover operations," *Proc. 2006 IEEE Int. Conf. Inf. Reuse Integr. IRI-2006*, pp. 23–27, 2006.

[48] B. Barshan and H. F. Durrant-Whyte, "Inertial navigation systems for mobile robots," *IEEE Trans. Robot. Autom.*, vol. 11, no. 3, pp. 328–342, 1995.

[49] A. Kelly, "General solution for linearized systematic error propagation in vehicle odometry," *Proc. 2001 IEEE/RSJ Int. Conf. Intell. Robot. Syst. Expand. Soc. Role Robot. Next Millenn. (Cat. No.01CH37180)*, vol. 4, pp. 1938–1945, 2001.

[50] J. Crowley, "Asynchronous Control of Translation and Rotation in a Robot Vehicle," *Proceedings. IEEE/RSJ Int. Work. Intell. Robot. Syst. '. (IROS '89) 'The Auton. Mob. Robot. Its Appl.*, 1989.

[51] K. Lee, W. Chung, H. W. Chang, and P. Yoon, "Odometry calibration of a car-like mobile robot," *ICCAS 2007 - Int. Conf. Control. Autom. Syst.*, pp. 684–689, 2007.

[52] S. LaValle and J. Kuffner, J.J., "Randomized kinodynamic planning," *Proc. 1999 IEEE Int. Conf. Robot. Autom. (Cat. No.99CH36288C)*, vol. 1, 1999.

[53] C. Cariou, R. Lenain, B. Thuilot, and P. Martinet, "Adaptive control of four-wheel-steering off-road mobile robots: Application to path tracking and heading control in presence of sliding," *2008 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, no. OCTOBER 2008, pp. 1759–1764, 2008.

[54] C. Samson, "Path following and time-varying feedback stabilization of a wheeled mobile robot," in *Int. Conf. Autom. Robot. Comput.*, vol. 9, no. 0, 1992.

[55] C. Cariou, R. Lenain, and B. Thuilot, "High accuracy path tracking of a four-wheel-steering all-terrain vehicle on a slippery slope," *Eng. a*, no. September 2015, 2008.

[56] M. Rufli, "Driver-in-the-Loop Path Control for a Non-Holonomic Vehicle," *Bachelor Thesis, ETH Zurich*, no. July, 2006.

[57] J. Vaganay and M. Aldon, "Attitude estimation for a vehicle using inertial sensors," *Control Eng. Pract.*, vol. 2, no. 2, pp. 281–287, 1994.

[58] Y. Fuke and E. Krotkov, "Dead reckoning for a lunar rover on uneven terrain," pp. 411–416, 1996.

[59] L. Ojeda and J. Borenstein, "Improved Position Estimation for Mobile Robots on Rough Terrain Using Attitude Information," no. August, pp. 1–14, 2001.

[60] C. C. Ward and K. Iagnemma, "Model-based wheel slip detection for outdoor mobile robots," *Proc. - IEEE Int. Conf. Robot. Autom.*, no. April, pp. 2724–2729, 2007.

[61] J. Yi, J. Zhang, D. Song, and S. Jayasuriya, "IMU-based localization and slip estimation for skid-steered mobile robots," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 2845–2850, 2007.